55－R015

第５世代のコンピュータ

基礎理論研究分科会報告書

昭和５６年３月

財団法人　日本情報処理開発協会

この報告書は，日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和５５年度に実施した「第５世代の電子計算機に関する調査研究」の成果をとりまとめたものであります。

序

わが国における社会経済は，資源，エネルギー問題を始めとして国際的な変動と，不確実性の流れのなかにある。同時に，的確な情報の加工利用が重要視される情報化社会の形成が指向されている。

コンピュータは，われわれの情報活用においてすでに不可欠なツールとなっているが，今後１０年間には多くの諸問題を解決するため，更に高度な技術が要求され，新たな理論・技術にもとづくコンピュータ・システムの実現が望まれるであろう。

このため，当協会では「第５世代コンピュータ調査研究委員会」を設置し，１９９０年代に実用化されるべきコンピュータ・システム（第５世代コンピュータ）はどのようなものになるか，またその開発プロジェクトはどのように進めていくべきかについての調査研究を，昭和５４年度から２ヶ年の予定で開始した。

昭和５５年度は，本委員会のもとに設置した３分科会（システム化技術，基礎理論，アーキテクチャ）および多数のワーキング・グループによる調査研究活動，内外の大学等への研究委託，米国への技術調査などにより，第５世代コンピュータのイメージおよび研究開発課題を明確化し，さらに，その研究開発計画・体制について検討した。

本報告書は，これらの調査研究結果のうち，基礎理論研究分科会の活動成果をとりまとめたものである。

最後に調査研究にご協力いただいた第５世代コンピュータ調査研究委員会委員を始め，関係各位に厚く御礼申し上げる次第である。

昭和５６年３月

財団法人　日本情報処理開発協会
会長　上野幸七

基礎理論研究分科会（Ｔ分科会）

	氏　名	所　属
主　査	渕　一博	電子技術総合研究所パターン情報部長
委　員	伊藤貴康	東北大学工学部通信工学科教授
〃	大須賀節雄	東京大学宇宙航空研究所助教授
〃	田中穂積	電子技術総合研究所パターン情報部推論機構研究室長
〃	田村浩一郎	電子技術総合研究所制御部論理システム研究室長
〃	長尾　真	京都大学工学部電気工学第２学科教授
〃	広瀬　健	早稲田大学理工学部数学科教授
〃	横井俊夫	電子技術総合研究所パターン情報部推論機構研究室主任研究官
〃	米沢明憲	東京工業大学理学部情報科学科助手
〃	雨宮真人	日本電信電話公社武蔵野電気通信研究所基礎研究部第一研究室調査役
オブザーバ	古川康一	電子技術総合研究所ソフトウエア部情報システム研究室主任研究官
〃	石崎　俊	電子技術総合研究所パターン情報部音声認識研究室主任研究官
〃	諏訪　基	電子技術総合研究所パターン情報部視覚情報研究室主任研究官

基礎理論研究分科会自然言語処理ワーキング・グループ

	氏　名	所　属
委　員	田中穂積	電子技術総合研究所パターン情報部推論機構研究室室長
〃	安西祐一郎	慶応義塾大学工学部管理工学科講師
〃	稲田俊明	山口大学教養部英語科講師
〃	白井英俊	東京大学工学部計数工学科助手
〃	鈴木英一	筑波大学現代語現代文化系助教授
〃	寺津典子	富山大学人文学部講師
〃	原口庄輔	筑波大学現代語現代文化系助教授
〃	溝口文雄	東京理科大学経営工学科講師
〃	山梨正明	京都大学教養部英語研究室助教授
〃	池田尚志	電子技術総合研究所制御部情報制御研究室主任研究官
〃	井佐原均	電子技術総合研究所パターン情報部推論機構研究室技官
〃	石崎俊	電子技術総合研究所パターン情報部音声認識研究室主任研究官
〃	内田ユリ子	電子技術総合研究所パターン情報部推論機構研究室主任研究官
〃	大谷木重夫	電子技術総合研究所制御部論理システム研究室技官
〃	板橋秀一	電子技術総合研究所パターン情報部音声認識研究室主任研究官
〃	佐藤泰介	電子技術総合研究所パターン情報部推論機構研究室技官
〃	松本裕治	電子技術総合研究所パターン情報部推論機構研究室技官
〃	三国一郎	電子技術総合研究所パターン情報部音声認識研究室技官
〃	山口喜教	電子技術総合研究所電子計算部計算機方式研究室技官
〃	横山晶一	電子技術総合研究所パターン情報部推論機構研究室技官

基礎理論研究分科会マシン理論ワーキング・グループ

	氏　名	所　属
委　員	横井俊夫	電子技術総合研究所パターン情報部推論機構研究室主任研究官
〃	佐藤泰介	電子技術総合研究所パターン情報部推論機構研究室技官
〃	元吉文男	電子技術総合研究所パターン情報部推論機構研究室研究員
〃	松本裕治	電子技術総合研究所パターン情報部推論機構研究室技官
〃	三国一郎	電子技術総合研究所パターン情報部音声認識研究室技官
〃	内田俊一	電子技術総合研究所ソフトウエア部情報システム研究室主任研究官
〃	長谷川洋	電子技術総合研究所ソフトウエア部情報システム研究室研究員
〃	二木厚吉	電子技術総合研究所ソフトウエア部言語処理研究室研究員
〃	塚本享治	電子技術総合研究所制御部情報制御研究室研究員
〃	大谷木重夫	電子技術総合研究所制御部論理システム研究室研究員
〃	島田俊夫	電子技術総合研究所電子計算機部計算機方式研究室主任研究官
〃	久野巧	電子技術総合研究所人間機械研究室研究室
〃	樋口哲野	慶応義塾大学工学部電気工学科
〃	松田利夫	東京理科大学理工学部情報科学科助手

基礎理論分科会知識ベースワーキング・グループ

	氏　名	所　属
委　員	古川康一	電子技術総合研究所ソフトウエア部情報システム研究室主任研究官
〃	長谷川　洋	電子技術総合研究所ソフトウエア部情報システム研究室研究員
〃	真野芳久	電子技術総合研究所ソフトウエア部言語処理研究室研究員
〃	白井良明	電子技術総合研究所パターン情報部視覚情報研究室主任研究官
〃	諏訪　基	電子技術総合研究所パターン情報部視覚情報研究室主任研究官
〃	富田文明	電子技術総合研究所パターン情報部視覚情報研究室研究員
〃	山崎正人	電子技術総合研究所制御部情報制御研究室主任研究官

基礎理論研究分科会情報基礎論ワーキング・グループ

	氏　名	所　属
委　員	田　村　浩一郎	電子技術総合研究所制御部論理システム研究室長
〃	梅　山　伸　二	電子技術総合研究所制御部論理システム研究室研究員
〃	大谷木　重　夫	電子技術総合研究所制御部論理システム研究室技官
〃	実　近　憲　昭	電子技術総合研究所制御部論理システム研究室主任
〃	大　津　展　之	電子技術総合研究所パターン情報部数理基礎研究室主任
〃	宮　川　正　弘	電子技術総合研究所パターン情報部数理基礎研究室主任研究官
〃	中　安　富士夫	早稲田大学理工学部数学科

目　　次

1. 基本的な考え方

1. 基本的な考え方

1.1 問題意識

基礎理論研究分科会の役割は，第５世代コンピュータ・システムおよびそれをベースとする新世代の情報処理技術体系のイメージづくりについて，基礎理論的な研究のサーベイを通して寄与することであろう。システム化技術研究分科会（第１年度社会環境条件研究分科会），アーキクチャ研究分科会との相対関係からいって，ここで「基礎理論」というのは，数学的な，といった狭い意味でなく，もっと広く，ソフトウェア基礎論の研究，人工知能の研究，パターン情報処理の研究を包含したものを考えるべきであろう。

それらの分野の研究のサーベイから，第５世代のコンピュータ・システムについて，確定した像を導き出せるわけではないであろうが，それに寄与できる段階には来ているように思われる。第５世代コンピュータ・システムについて前もっての定義はない。ただ，現行のシステムの漸進的改良という線上でなく，その先に飛躍した段階を想定したいという問題意識がある。その第一義的な意識の出てくる由縁は，現行のシステムに対する不満であり，それは漠然とした表現では「使いにくい」ということであろう。

そのことは，最外縁のユーザ大衆にとって現状の技術についてあてはまる。その不満は現在の技術の未熟さから来ているわけであるが，それは現行の技術体系をだんだんと改良していけば解消するものであろうか。あるいは現行のコンピュータ・システムの基本的な造りの悪さに起因するものであろうか。

同様の不満は，コンピュータ技術の内部からも出てきている。それは大規模なソフトウェア作成の困難さということに集約されよう。それは，ソフトウェア工学の未熟さによるのだろうか。それもある。一方，コンピュータの造り自身への反省も出てきている。

そこで第１の問題意識は，

◎ 使いやすいコンピュータ・システム

ということになろう。使いやすさにはいろんな側面がある。機能の面からもいくつかの把え方があるだろう。ユーザ側からのニーズとしては，社会環境条件研究分科会から抽出・整理されているが，その中に「日本語マシン」の問題意識がある。現状は，日本語の自由な使用という理想にほど遠い。最近ようやく，漢字かな入出力やそれに伴う処理技術が進歩しはじめたが，それはまだ「字」のレベルにすぎない。それが第一歩であるのは確かであるが，それを日本語として，データベースの問合せやプログラミング，質問応答や助言として用いるのは，研究面で試みが始まった段階にすぎない。自由な使用には「意味」の取扱いが可能になってこなくてはならないが，そのためにはまだ多くの研究が必要である。これは，日本語処理ということにも，もっと高機能化が必要であるということである。

日本語は，コンピュータとの自然な会話形態であるが，自然な会話（使いやすさ）ということからは，図形的会話ということも挙げておかなければならない。またコトバの入出力機能としては音声入出力も望まれる。これらの実現には高い機能を必要とする。そこで，

◎ 高機能化・多機能化・高性能化

が使いやすさの一つの条件であり，また問題意識として考えられるべきであろう。

コンピュータの高性能化ということで一番分かりやすい例は，超高速数値計算であろう。これはいまや特殊応用ということかもしれないが強いニーズがある。その高性能化からは現行のコンピュータの造りから離れた「非ノイマン型」アーキテクチャへの試みが始まっている。

一方，高機能化や多機能化という観点からは，数値計算だけでなく，もっと広く「記号処理」の高度化と高能率化が必要になってくる。それはコトバの処理，ソフトウェア作製などにとって必要なことである。

高機能化にとって今後重要になる観点は，

◎ 知識情報処理

である。現在のコンピュータ技術では，問題解決のほとんどの所を「プログラム」にしてやらなければならない。これがコンピュータの使い難さにも通じている。プログラムの自動作成は研究テーマの一つであるが，視野を広くとらなければ進歩は望めまい。一つには，ソフトウェア基礎論的検討がもっと必要である。第二には「知識」の組み込みが必要になってくる。問題の対象領域についての情報，さらにはそこでの法則性の情報を組み込む方向で，コンピュータの問題解決能力を一段向上させる必要がある。このような情報は「知識」と呼ばれるものに相当する。これはいいかえれば，コンピュータ・システムの中に世界のモデルを持たせることである。

これは，人間とコンピュータの相互了解の範囲を拡大することに相当し，コンピュータの問題解決能力を向上させるだけでなく，自然な会話にも必須の能力であり，「使いやすさ」を実現させる基礎である。

言語や知識を介して高機能を実現するということは，７０年代において「人工知能」研究の主テーマであった。これにはまだまだ数多くの研究テーマが残されているが，第５世代コンピュータ・システムを考えるときの素材がここに存在しているといえる。

この方向の高機能化の観点からも，新しいコンピュータのつくり（アーキテクチャ）への要望やそれへのヒントが出てきている。

そこで，

◎ 新しいコンピュータ・アーキテクチャ

の問題意識が生じる。これは，

◎ 伝統的な体系からの脱却

という問題意識につながるといえよう。そこで，

◎　新しい情報処理技術体系

ということが，使いやすさから出発した，総合的なゴールといえよう。

こういった問題意識について，

◎　基礎理論による裏づけ

が可能かということになる。

コンピュータの誕生には，チューリング機械の理論という（いかにも基礎理論らしい）数理論理学の成果が巧みに生かされた。また生理学でのマカロック・ピッツのモデルは論理素子の概念に結びついている。これらがノイマンらによって巧妙に統合化され，現在のコンピュータの構成原理（ノイマン型アーキテクチャ）が立てられたのである。

しかし，その後のコンピュータは，むしろ工学的に自律的に発展してきた。オートマトン理論などがあったが，それらがコンピュータの進化を直接導いたとはいいがたい。逆に，コンピュータ工学自体が，内容的には，既存の論理学より豊富なものをつくり出したといえる。

その内容の形式化（理論化）が本格的になったのは70年代に入ってからである。それは「ソフトウェア基礎論」の分野である。これは現在発展中で，まだ多くの研究テーマを今後に残している。しかし，一方では，新しいプログラミングのスタイルの提案や新しいアーキテクチャへの意識がそれに関連して生じている。

前述した「人工知能」の分野では，人工知能用プログラミング言語という問題意識から新言語の提案があったが，それもまた，新しいマシンへの意識をはらむものである。

注目すべきは，両者の動きが一応独立した展開にありながら，最近ではそれらのベクトルの向きが合ってきつつあるということである。

そういう現象からすると，広い意味での基礎理論が，新しい情報処理技術体系，ひいては新しいアーキテクチャ（第5世代コンピュータ）へ寄与するということは，80年代において大きな可能性があると思われる。

第5世代コンピュータ・システムへ向けての問題意識を整理すれば前述したようなレベル設定になるであろう。そういった観点で行なわれた検討結果からやや大胆にまとめれば，第5世代コンピュータ・システムに関して，

◎　知識情報処理システム

というイメージが浮かび上がってくる。

さらに，こういったイメージを実現するために，その過程において，

◎　システムを作りやすい高度の環境

が不可欠であるという問題意識が生じる。

これはゴール実現のための手段であると同時に，単なる支援システムでなく，進化するシス

テムとしての知識情報処理システムの要素として重要視されなくてはならない。それ自身，知識情報処理のプロトタイプの役割も果たすのである。これはまた，人材育成（我国では，先進的な研究者，技術者の数が米国等に比してあまりにも少ない）の環境として，層の厚い技術として知識情報処理を育てる土壌となろう。

以上のような設定は先進的にすぎるように見えるかもしれない。しかし，従来技術の漸進的改良の線上に難問の解決方策が見えず，それが新世代への悲観論を生んでもいるが，それは現行技術のある種の成熟現象と見ることができ，むしろ，新世代への機が潜在的に熟していると読むこともできよう。一方これは，これまでの先進的研究の延長上にあり情報処理技術の必然的な方向と見ることもできる。タイミングの問題は別にして，要は立ち止まるか，前進するかにあると思われる。

1.2　知識情報処理システムの構図

問題意識の帰結として，

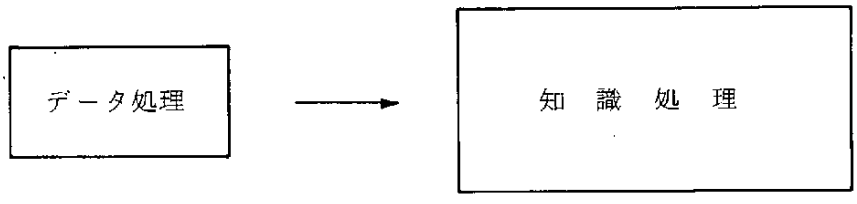
第５世代コンピュータ・システム

‖

知識情報処理システム

という設定を行なった。もう少し，その輪郭を明らかにするために，現在，最も広く使われているコンピュータ・システムと比較し，その特徴を浮彫にしてみる。

情報処理の形態は大きく進化する。単なるデータ処理から“知識”の処理へと移行する（図１－１）。知識は“意味”と言いかえてもよい。比較は，この進化の軸に沿ってなされる。



現世代　　第５世代

図１－１　処理形態の進化

まず，システムを処理の基本機能から比較する。情報処理の基本機能として３つを上げることができる（図１－２）。情報の管理を行なう機能，情報の加工を行なう機能（処理と呼ぶ），情報の伝達・交換を行なう機能（対話と呼ぶ）である。伝達・交換には，人間とシステムとの間の他に，ネットワークを介してのシステム同志のものもある。ここでは，ネットワークに関

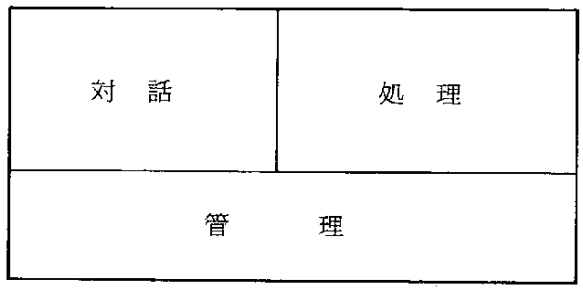


図１－２　処理の基本機能

する説明は省略する。管理機能に関しては，単純なデータの蓄積と検索から，知識の獲得・利用へと進化する（図１－３）。知識とデータの差は，簡単に述べると，"これこれのデータが

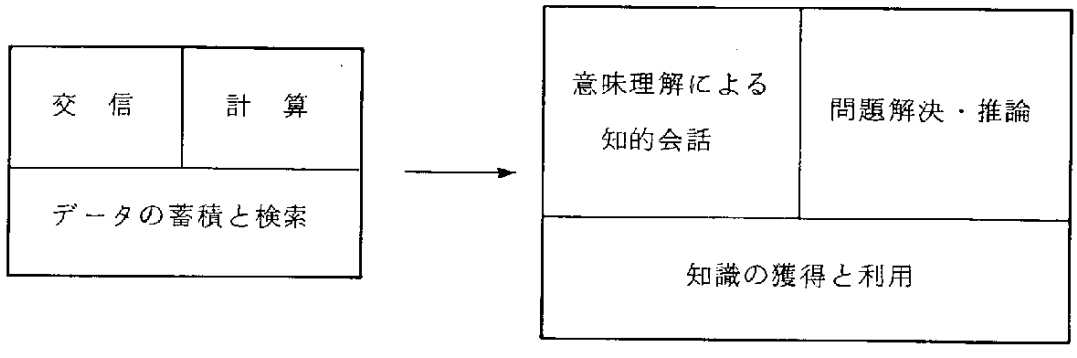
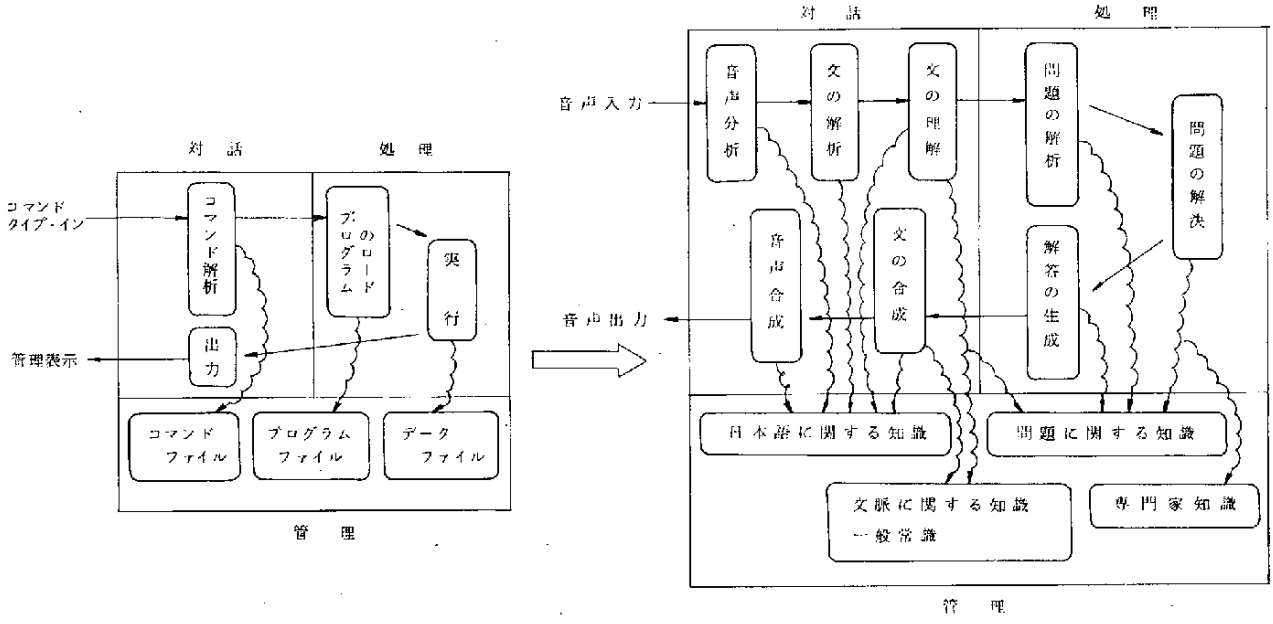


図１－３　基本機能の進化

存在すれば，こういうデータも存在するものと思ってよい"という規則や，さらに，それらの規則をどう利用するかといったメタ規則等々のことである。処理機能に関しては，数値計算や事務計算のように指示された手順を唯実行するというものから，高度な問題解決へと進化する。問題解決システムには，"こうしてほしい"という要求が，簡潔な記述で与えられる。その記述は，問題領域の知識や一般常識をシステムが持っていることを前提としたものである。そこで，問題解決システムは，それらの知識を用い，不足した情報を補い，どうすれば要求に答えられるかを探索・推論する。そして，求められている解答を生成する。最後に対話機能に関しては，データのやり取りという交信から，メッセージの意味を理解し合う会話となる。そのメッセージの媒体も，音声，自然言語，図形，画像と多岐にわたる。この３機能を関連づける機能イメージの進化の一例を図１－４に示す。



現世代

第5世代
（知識情報処理）

図1－4　機能イメージの進化

次に，情報処理システムを基本的な構成要素に分解して比較をより精密にする。システムは，機能の高さによる包含関係でその構成要素を表わすと，一番機能の低いものとして機械（マシン）系があり，順に記述系，応用系，そして，そのシステムを利用する人間をも含めると最外側に人間系という4層の構造を持っている（図1－5）。もちろん，この包含関係は，3つの

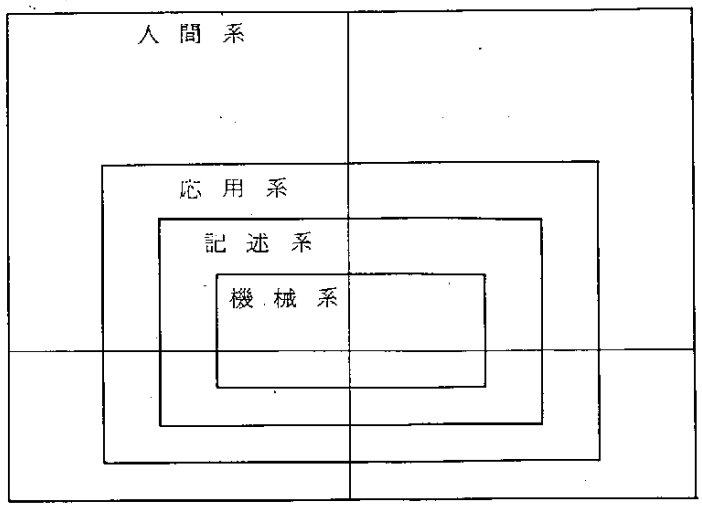


図1－5　系全体の構成要素

基本機能に対し，均等ではないし，単純処理に関しては，人間より機械の方が能力的に勝るが，ここでは，処理の質的な面から単純化して図化してある。機械系というのは，計算機ハードウェアを表わす。記述系は，その計算機の上で使用される最もポピュラーなプログラム言語を表わす。記述系から見ると，そのプログラム言語を解釈できる情報処理システムというイメージを与える。応用系は，その言語で書かれたプログラムやデータ群によって作り出される様々の機能である。

各構成要素に対して，基本機能の進化の有様を示したのが図１－５である。現世代は，最も一般的なものを挙げてある。少々公平さを欠く，きらいがある。細かな説明は省略するが，い

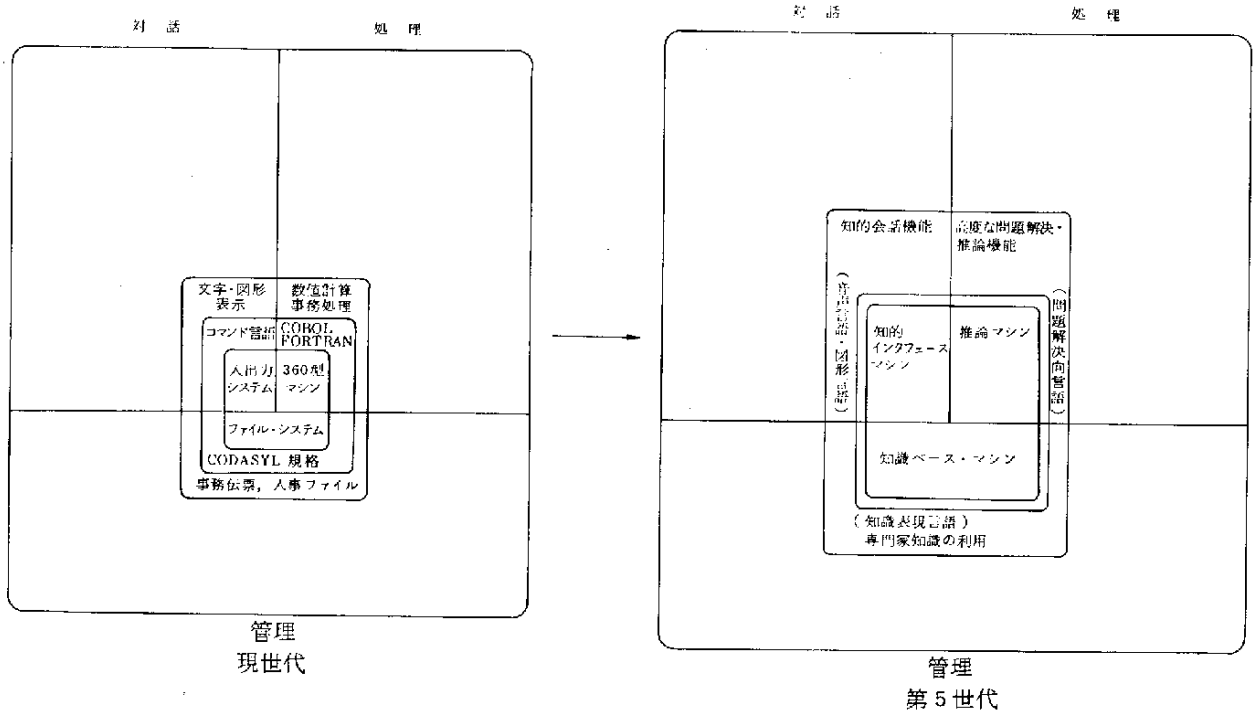


図１－６　構成要素から見た進化

くつかの留意点について述べておく。応用系の部分は，本来は，様々な応用システムを重ね合わせて表わさねばならない。しかも，各応用システムの基本機能への重点の置き方も様々である。ここでは，単純化し，代表的な機能によって表わすことにする。図に見られるように，構成要素間の相対的な差に変化がある。記述系と機械系の差が大きく縮まる。第５世代では，アーキテクチャが高レベル化され，セマンティック・ギャップの少ないマシンが開発され，そのマシン言語は，記述系に非常に近いものとなる。応用系と記述系の開きが大きくなる。第５世代では，システム化を支援する高度なシステム，例えば，知的なプログラミング・システムや高機能なモジュール群が開発され，プログラム化の対象が，大巾に高度化，広範囲化するから

である。図ではあまり判然としないが，人間系も第５世代では，少々成長する。これは，突如，利口になるという意味ではない。高機能なシステムを手にし，それを駆使することによって，新たな知的活動分野を開拓していくという意味である。

1.3　報告書の構成

２章に，知識情報処理システムとしての第５世代コンピュータ・システムの目標像を説明する。３章には，その達成を目差す研究・開発課題の内容を，各課題ごとに次の形式に従い説明する。

〔１〕　概　要

〔２〕　必要性

〔３〕　内外技術の現状

(1)　国内技術

(2)　海外技術

〔４〕　研究開発内容

(1)　研究開発項目

(2)　研究開発スケジュール

(3)　研究開発の目標・仕様

４章では，すべての課題を統合し，相互の関連付けを行ない，全体としての研究・開発のストーリを述べる。研究・開発の手順，研究・開発支援ネットワーク・システム，研究・開発スケジュールを説明する。５章，６章は，それぞれ，システム化技術研究分科会，アーキテクチャ研究分科会に対するインタフェースの役割を持つ。

なお，本分科会の報告書の付録に，各論エッセイとして関連研究の解説と，ＷＧ報告書として分科会を補完するため電総研，その他で組織されたワーキング・グループの報告を添付する。

2. 研究開発目標

2. 研究開発目標

2.1 設定の方針

第５世代コンピュータ・システムは，基礎から積み上げねばならない非常に多くの研究・開発要素からなる。したがって，達成すべきものを一つの明解なシステムとして描くことは困難である。しかし，調査結果からも明らかなように，一つのシステムにまとめ上げうるという高い確度を持って，研究・開発の方向付けを行なってもよいというのも大きな事実である。すなわち，一つの融合化されたシステムを想定できる。第５世代コンピュータ・システムを設定できるという技術的な条件が整った，あるいは整えうるという判断が，本調査の結論でもある。必ずそう達成されるというものではなく，そのようなものらしくなるというイメージ，目標“像”としてなら，現在でも設定は可能である。具体的にどのようなシステムに落着くか，一つの融合化されたシステムになるか，その融合化の仕方，あるいはいくつかのすぐれた専用システムを含んだものとなるかは，研究・開発の成果に照し合せながら注意深く判断されねばならない。目標システムの正確な仕様の決定は，中間目標達成後にはじめて可能となる。次節で述べるものも，あくまでイメージ，目標像としてのものである。

2.2 目標像

知識情報処理システムは，図２－１に示すように，現世代のシステムにアナロジーをとり，

知識情報処理システム〔第５世代コンピュータ・システム〕
- 知識情報処理プロセッサ（ハードウェア・システム）
- 知識情報処理オペレーティング・システム（ソフトウェア・システム）
- 知識情報処理応用システム

図２－１　システム全体像

プロセッサとオペレーティング・システムからなるものとする。この上に様々な応用システムが作り上げられる。目標は９０年代の汎用計算機システムである。現在の商用汎用計算機システムの流れは，ＩＢＭ／３６０を実質的な源としているが，第５世代から始まると想定する新しい計算機システムの流れの源となるものが目標である。もちろん，ＩＢＭ／３６０と第５世代では，汎用機の持つ意味も役割もことなる。第４世代から本格化する多様化の傾向は，第５世代にも受け継がれ，さらに強調され続いていくであろう。したがって，この目標像も，出来上ったものが，ほとんどそのままの姿で商品化されるという可能性も考慮するがそれと同時に，90

年代の新しい多様化に対応できる新技術の総体系として，一つのシステムにまとめ上げるという意味合いも強く含むものである。さらに，その時点での最高の知識情報処理機能を持ち，あくまでも性能追求を中心課題とする。速度，容量，機能どの面から見ても"超"という文字を戴くにふさわしいものとする。システムは単一ユーザを対象とするパーソナル・マシンとして開発する。"超"とはいっても，それなりのバランスのとれたシステムとなる。ただし，あくまでも性能追求を中心とする。

以上は，システムとしての目標のスケッチである。目標のもう一つの柱は，新しい理論体系，新しい情報処理理論の確立にある。現在の計算機技術の流れが，急速な実用化，商品化の流れにおされ，理論的な整備が後を追う形となり，結果として得られたシステムが多くの欠陥を内に含むものになっている。もちろん，諸技術の制約上，そうせざるをえなかった面もあるし，そのような急速な商品化の熱意が，現在の情報処理技術を支えてきたことも事実である。現在の計算機技術を細部にわたって点検し，骨格を組み直し，新しい情報処理技術の体系を作り出す。その結晶，あるいは実証として第5世代コンピュータ・システムをとらえる。これこそ，本プロジェクトの性格を決める要点である。理論の中心となるのは，新しい論理学である。自然言語の意味，プログラム言語の意味，知識ベース＝意味ベース，問題解決過程の意味記述等々，広範囲な意味の取扱いを可能にするためには，従来の制約を大きくのりこえる新しい論理学の構築が必要である。

知識情報処理システムとしての目標像をもう少し具体化する。知識情報処理プロセッサに関しては，第6章マシンのイメージを参照されたい。知識情報処理オペレーティング・システムに関しては，そのイメージを与えるため，図2－2に全体の構造を示し，以下にその各構成要素の説明を述べる。

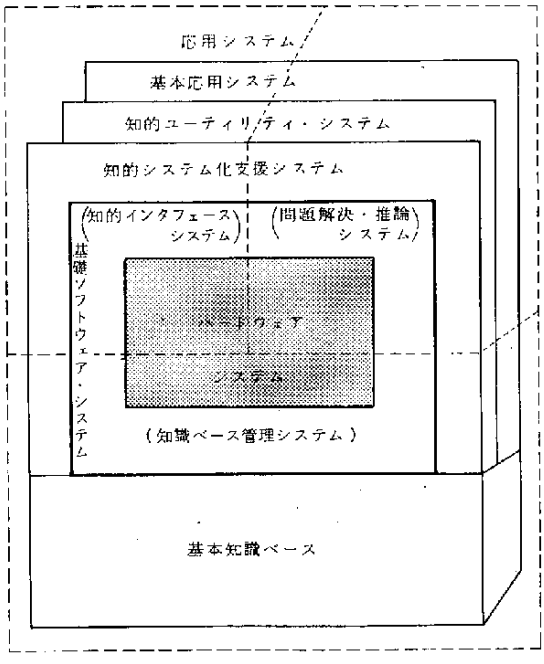


図2－2 知識情報処理オペレーティング・システムの構造

(1) 基礎ソフトウェア・システム

全システムの中核となる部分で，情報処理の基本機能（管理，処理，対話）に対応するモジュール群からなる。研究・開発課題（知識ベース管理システム，問題解決・推論システム，知的インタフェース・システム）の成果のうち，ハードウェア化されない部分と定義してもよい。

(2) 知的システム化支援システム

様々な応用に最適な情報処理システムを設計・作成（システム化）する際に，作成対象，作成過程等に関する知識を持ち，人間のシステム化作業を大巾に軽減するためのシステム群である。これは典型的な知識情報処理システムの例である。厳密な仕様記述言語と記述された仕様から，その対象の作成へと導びくサブ・システム，あるいは正しさを検証するサブシステム，動作をシミュレートするサブ・システム等からなる。対象別に見ると，さらに3つの支援システムにて構成される。すなわち，プログラムを対象とする知的プログラミング・システム，知識ベースを対象とする知識ベース設計システム，ＶＬＳＩチップや計算機アーキテクチャを対象とする知的ＶＬＳＩ設計システムである。

(3) 知的ユテイリティ・システム

システム自体の利用を容易にするための高機能な便宜を提供するシステム群である。これは，既存の商用機上に蓄積されたプログラムやデータ・ベースを目標機上に移植するための互換性維持システム，システム全体や各サブ・システムの機能や使用法等を解説したり，利用者からの相談に応じるシステム解説・教育システム，システムの簡単な故障に対しては，自動検査や自動修復を行なったり，複雑な故障に対しては，検査や修理の手導きや相談に応ずる知的故障診断・保守システム等々からなる。これらは，独自の研究開発課題としては設定されていないが，他の課題の研究開発段階に含まれて開発が進められる。

(4) 基本知識ベース

システム自身も利用するし，利用者，誰もが利用する普遍的な知識を知識ベースの形にまとめ上げたものである。前述の各システムの構成要素ともなっているし，利用者の作る様々な応用システムにも大いに利用される。大きく3種類の知識ベースにわかれる。すなわち，常識に類する一般知識ベース，システムに関する知識を集めたシステム知識ベース，ある応用分野の知識を集めた応用知識ベースにわかれる。一般知識ベースには，日常基本語・基本文型・基本スクリプトベースや各国語辞書・構文規則ベース等，自然言語関連のものが多くなる。システム知識ベースには，プロセッサ仕様記述ベースやオペレーティング・システム仕様記述ベースのようにシステム自身の仕様を集めたものや言語マニュアル・ベースや利用度の高いプログラムを集めたプログラム・モジュール・ベース等がある。応用知識ベースには，ＶＬＳＩ設計技術ベースや計算機アーキテクチャ・ベースや基本プログラム型ベース等がある。

(5) 基本応用システム

後に説明する基本応用システムグループに類別される研究・開発課題それぞれの最終目標性能値を持つシステム群である。表２－１に，これら各々のシステムの目標性能値の概略を示す。

表２－１　基本応用システムの目標像

基本応用システム名	目　標　性　能　値
機械翻訳システム	○多国語間翻訳 ○使用辞書項目数　１０万語 ○９０％の精度で翻訳し，人間の介在によって残り１０％を処理する。 ○テキストの編集，翻訳結果の印刷までの各工程で計算機が関与する総合システムとする。 ○全コストは，人間が翻訳した場合の３０％以下とする。
質問応答システム	○各種専門分野に対する質問応答システム・プロトタイプ ○使用語彙数　５０００語以上 ○推論規則数　１００００以上
音声応用システム	○音声タイプライタ：１万単語を対象とし，意味解析機能を持ち，音声認識上の誤りを自ら修正し，わかり易い文章を出力する。 ○音声応答システム：１万単語を対象にし，応答内容の意味を把握し，自然な会話となる。 ○話者認識システム：約100名以上を対象とし，実用的な時間内に認識する。
図形・画像応用システム	○１０万枚位の図形・画像を対象に，抽象レベルを含めた格納所要時間数sec以内，検索所要時間は平均100 msec以下とする。
応用問題解決システム	○数式理解システム：法則，公式をルール・ベース化した高機能数式処理システム ○碁プレーイング・システム：アマチュア初段程度の実力を持つシステム

(6) 応用システム

具体的な応用システムは，図２－３に類別されるように無数に設定できるが，目標機上には数例を実証例として実現する。

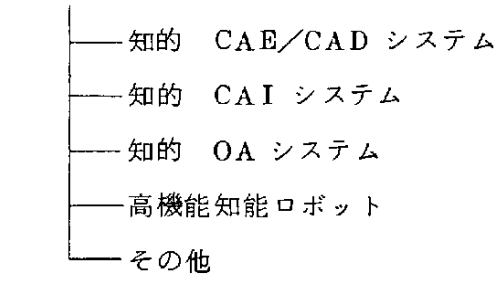


図２－３　応用システム

3. 研究開発課題

3. 研究開発課題

3.1 概　要

前章の目標像を目指し，３グループの研究・開発課題を設定する。各グループの概要は以下のとうりである。

(1) 基礎ソフトウェア・システム

基本機能（管理，処理，対話）のそれぞれに対応する，システムの中核となる記述系，アルゴリズム，基本ルール等を研究・開発する。これらは，知識情報処理システム目標機のアーキテクチャを規定すると同時に，オペレーティング・システムの核となるシステム群の仕様となる。

(2) 知的システム化支援システム

目標機の開発およびその上に様々な応用システムを構築していくための知的なツールを研究・開発する。厳密な仕様記述と検証システムが大きな役割をはたし，新論理系と推論機構，仕様記述方式の研究・開発が主要なテーマである。

(3) 基本応用システム

それぞれの機能（聞く，話す，見る，描く，考える，解く）を代表する基本的な応用システムを研究・開発する。本システム自身が非常に広い適用範囲を持つものであると同時に，各基本応用システムを構成するプログラム・モジュール群や知識ベース群は，それぞれが高い利用価値を持つ部分品となる。さらに，基礎ソフトウェア・システムで提案される記述系，アルゴリズム等や知的システム化支援システムを具体的に実証するという役割も持つ。

3.2 基礎ソフトウェア・システム

3.2.1 知識ベース管理システム

〔1〕 概 要

人間の持つ知識を一定の形式で表現して計算機内に蓄積し，利用することによって，問題解決のプロセスにおいて人を援助する知的計算機の技術を確立する。開発の方針としては，特定の問題分野に依存しない汎用のシステムを目指し，かつ既存の計算機技術および蓄積された情報の利用を保証しつつ，新計算機への円滑な移行が可能なものとする。本システムの主たる目的は，その高度な知的機能により，従来の計算機技術のもとでは困難とされた，設計・診断・意思決定などのプロセスやプログラム開発などにおいて人を援助すること，自然言語・音声・図形・画像等の理解を含む強力なマン・マシン・コミュニケーションの能力によって非専門家による計算機の利用を容易にすること，機械翻訳・知能ロボットなど高度の応用技術を可能にすることなどである。これを汎用性のある計算機の上で実現することにより，従来の計算機技術のもとで期待できなかった広範囲な分野に計算機利用を拡大することができる。このような新らしい計算機システムを知識情報処理システムと呼ぶ。

従来の計算機とこのような新計算機の基本的な相違は，与えられた問題にたいし，前者では計算機入力以前に人がその問題の結果を得るための手順を求め，その手続きを一定の形式で表現して計算機入力とするのにたいし，後者では問題の意味そのものを一定の形式で直接に計算機に入力し，その結果を得るプロセス自身を計算機自身が構成し，結果を得る点であり，いわば前者におけるプログラム作成に相当する部分の一部を計算機が受け持つ。

結果を生成するための手順を人間が生成し，プログラム化するには問題に関する多くの知識を利用する。人間が現在行なっているこの仕事を情報処理過程と見做せば，入力である問題表現から出力であるプログラムを生成するためにこれらの知識は不可欠である。いわばプログラムのもつ情報量は問題表現に含まれる情報量にこの知識に含まれる情報量（の一部）を加えたものに等しい。この状況は知識情報処理システムでも同様であり，前述の機能を発揮するためには計算機自身が問題に関する多くの知識を保有していることが基本条件となる。

このために必要となる基本技術は，(1)知識の表現とその蓄積の技術，(2)問題解決のためにその中から必要を知識を検索し，利用する技術，(3)人間の持つ知識を移植し，また人間の知識の増進に応じて，蓄積された知識情報を更新・挿入・削除する技術であり，この上にさらにマン・マシン・システムとしての各種技術，人間の知識獲得のために計算機が援助し得るための技術など多くのものが必要となる。

従来の計算機では問題を手続きにより表現し，この手続き内の命令を形式的に実行する機械として汎用性が実現された。個々の問題ごとの固有性は基本命令の列として手続きを構成する順序に含まれ，これは手続き内の個々の命令を前もって定められた順序で逐次実行するように

構成された計算機の機構に影響しない。一方，知識情報処理システムでは問題解決に際して必要とされる知識の内容は問題ごとに異なる。従って，これを汎用計算機として実現するためには知識の記述形式を定め，それを用いて記述される知識の内容と，それを問題解決に役立てるための利用手段とは完全に独立なものとする。すなわち処理系は記述の形式に基づいて動作するようにする必要がある。この結果，問題の固有性はこの形式を用いて表現された知識の内容にすべて含まれ，これは計算機の機構に影響しない。

このように構成された知識情報処理システムにおいては，分野ごとに必要な知識の総量は異なるため必要な計算機の規模はそれに応じて異なることはあっても，機構は原則的には問題によらない。しかし，実現をめざすこのような計算機開発の第一目標として，2,000語程度の用語（共通語1,000語，分野別専門用語1,000語）を用いて表わされる。約1,000の事実，法則，諸定義を表わす知識の集まりである知識ベース，関連データの集積であるデータ・ベースおよび知識利用の際に必要な基本アルゴリズムの集まりであるアルゴリズム・ベースを備え，それらを利用して問題を解決するための推論機構を備えた知識ベース・システムを想定し，これらを統括的に管理する知識ベース管理システムの実現をめざす。

〔2〕 必要性

前節で触れた設計・研究開発のような創造的な作業，医療診断や意思決定などの作業，機械翻訳や自然言語・音声・図形・画像理解などのような情報の意味に基づく処理，知能ロボットなどの高度な応用システムなどの分野への計算機の利用はこれまで各種の試みはなされてはいたものの，実質的にはほとんど効果を挙げ得ないままに残されていた分野である。しかも社会全体の情報化，知識集約化の傾向を反映して計算機化の要求が急速に増大している分野である。すでにこのような知的な作業分野に従事する人口が急増しているが，近い将来，要求に応ずる人的資源の供給が追いつかなくなり，知的作業の効率化のために計算機による援助システムを開発することは生産性の向上のみならず，それに従事する人々を煩瑣な作業から解放し，より人間的な作業環境を整える上でも重要になる。さらに計算機産業にたいしては，この分野は非常に大きな市場を形成することが指摘されねばならない。それは従来の計算機が社会活動の特殊な部分にのみ用いられてきたのにたいし，今後は利用分野が拡大し，利用者層が急増すると予想されるからである。

従来の計算機技術が何故このような要求に応え得なかったかについては少くとも2つの理由がある。第1はハードウェアが高価であったことであり，第2はさらに本質的な情報処理機械の原理と構成が欠如していたことに関するものである。

前述のような要求を実現するには大量の記憶容量と高速の処理装置が必要である。また計算機にたいする負荷が大きく，計算機の占有時間も大になる。したがってハードウェアが高価であった時代には実現は困難であった。近年，ハードウェア価格は急速に低下し，性能も向上し

ている。同一性能の計算機のハードウェア価格と，人件費との比率は1960年代と比較して現在はすでに3桁の程度で変化している。このことは貴重な人間の能力を発揮するために計算機を投入することが多くの分野で経済的にも可能となってきていることを示している。

第2の理由は，計算機技術の基本に関わるものである。従来の方式のもとでは人が問題に対する解の手順を書き表わした。この方法は，記述形式が計算機の構造に基づいて定められているため，ユーザは少なくとも計算機に関して準専門家としての知識を（計算機を使うために）要求されること，問題を解くプログラムを作るために多大の時間を要すること，が利用者層の拡大の障害になってきた。そしてさらに重要なことは，プログラムを書くためにはすべての可能な処理の道筋が前もって見通されていることが前提になっていることである。すなわち，利用分野はこのような見通しが可能なものに限定される。しかるに前述の諸要求は状況に応じて動的に処理手順を変更しつつ，発見的に解を求めることを基本とするものであり，処理方式が基本的に異なる。これが従来の計算機技術の枠内でこれらの要求を満たし得なかった最大の理由であり，またこのために新らしい情報処理機構を必要とする理由である。

このような問題の基本的な解決策は処理手続きを実行する前に人が行なうという方法によらず，これを問題に応じて動的に生成することである。さらに，必要なのはプログラムという中間生成物を生成することでなく，問題に関する知識を有効に利用して結果を得ることである。

知識は非常に多様性に富んだものであり，かつ一般には不完全なものである。これを計算機内に蓄積し，有効に利用するために，そしてさらに新らしい知識を獲得してゆくためには，知識を収容する知識ベースを管理する知識ベース管理システムの技術が重要である。この機能はユーザが与えた問題にたいし，解を得るために利用できる知識を探索し，これを用いて原問題を変換して解に近づけるというプロセスを繰返して最終的に解を見出すこと，新らしい知識の挿入，削除，更新を行うこと，またそのために知識ベースの無矛盾性チェック，冗長性チェックを行なうこと，必要に応じて既有の，もしくは新らしいデータベース管理システムに管理されているデータを要求し解のプロセスに用いることなどである。問題解決に際して知識の効率的な探索がなされるように知識の組織化が行なわれ，また有用な知識が複数存在する時には最適な実行順序をきめることが要求される。なお無矛盾性チェックに際しては，いくつかの相互に矛盾する仮説を挿入する場合もあるので知識の中にフレームを設定し，各フレームの中でのみ無矛盾性をチェックする方式をすることも重要である。さらに知識が不完全である場合には得られる結果も不完全となるが，知識の信頼度に基づく結果の信頼度評価や，データを通して知識の信頼度を変更してゆく学習的機能が必要となる。

このように知識情報処理システム内の知識はいわば従来の計算機の応用プログラムに相当する。したがってこれを操作する処理系，すなわち推論機構はこれより1レベル上のメタ・レベル処理系である。これは知識表現の形式情報に基づいて処理を行なうから，動作は規則的であ

り，近い将来ハードウェアによって実現することも可能である。これにより処理速度も著るしく向上することが期待される。

〔３〕 内外技術の現状

(1) 国内技術

国内では東京大学で汎用システム化のための知識ベース管理システムの研究や医療用コンサルテーションシステムの研究が，京都大学で知識表現のためのセマンティック・ネットの研究および日本語理解のための知識利用方式の研究，大阪大学でアクタを基礎にした知識利用システム，九州大学で知識利用の情報検索方式の研究，電総研で推論方式やデータベースへの動的アクセスの研究，豊橋技術科学大学で知識を用いた画像理解の研究，電電公社武蔵野通研で知識ベースに基づく日本語による図形の世界の処理の研究，東京理科大において医療診断用システムの研究などがそれぞれ進められている。最近では知識表現や推論方式の研究にたいする関心が増大しており，研究者も増えつつある。しかし現状では多くの研究は未だ開始されたばかりであり，知識ベース管理システムの必要性，その満たすべき条件等が明確にされているものは少ない。今後はこれらの条件と，そのうち満たすためのシステムの基本アーキテクチャを明確にし，またそれを満たす知識表現，推論方式データベースの利用等に関する研究を促進することが必要である。

(2) 海外技術

知識ベース・システムの研究は米国において活発であり，医療分野のコンサルテーション・システムとしてMYCINやEXPERTが有名であるが，この他にもこの分野では十指を越えるシステムが開発され，あるいは開発中である。最近ではさらに医療以外の分野にもこのような知識ベースを利用するシステム開発が盛りであり，これらは知識工学という新らしい技術分野を形成している。

米国における知識ベース・システムの基礎的研究はこれらの具体的な分野ごとの利用システムの開発以前に知識表現，推論方式，知識表現用言語などの基礎分野の研究，自動プログラム合成の実験的試みなどという形で実績をもっているが，汎用の知識ベース管理システムについてはEMYCINのような概念が出されているものの，必ずしもその要求仕様が明確化されていない。今後，この方向の研究も増えるものと予想される。

〔４〕 開発内容

(1) 研究開発項目

(a) 知識表現・利用技術の研究

一般的な知識の表現法に関する研究を行う。第１に，知識そのものの研究が必要である。知識についての多くの様相を明らかにし，それを利用の観点から分類し，それら各種の知識について適当な表現法を研究開発すべきである。知識の様相として考えられるのは，一般・専

門，大局・局所，完全・不完全，仮説・恒真などである。

知識を利用の側面からみると，その利用のされ方は，多様である。そのため，各目的に合った形に容易に変換されることが必要である。たとえば，人間に対する応答では，言語や，画像の形で表現してやることが必要である。また，データベースに対する手続きなどのプログラム化も必要である。

このような知識表現の問題は，知識ベース管理システムの中枢部分であり，可能な表現方式を研究開発し，その評価を行う。そして，知識表現言語を設計・開発する。

また，知識の利用技術として，知識ベースからの知識のとり出しのための検索言語の研究開発を行う。さらに，知識利用に関する知識についての基礎理論とそれを表現・利用するための基本的枠組の研究を行う。

(b) 知識獲得・学習の研究

知識を獲得して，システムに付け加える場合，すでに存在する他の知識と矛盾してはならない。そのためには，第1に無矛盾性のチェック機構が必要である。これは論理学で一般には限られた時間内に決定できないとされている難問である。ある条件の下で有効に働くアルゴリズムの研究開発が必要である。第2に矛盾が生じた時に，どのようにしてその矛盾を取り除くかが問題である。この2つの場合がある。それは，新たに入力された知識が誤っている場合と，元々の知識ベース中に誤りが含まれている場合である。これは，とくにDefault値との間で起こり易い。一種のTruth Maintenance Systemの開発が必要である。

学習については，データの集合から，規則を発見することが必要となる。この場合には，どのような規則を考えるかが最も大変な問題である。そのためには，類推や補間などの帰納的推論が重要となる。そのようにして作られた規則は，実際に正しいかどうかを確かめなくてはならない。統計的仮説検定論は，ある種のデータに対して，その検証のための有力な方法論を与える。

(c) 大規模知識ベース・システムの研究

知識ベース・システムでは，変数を含まない個別データも他の手続きを表わすデータと同じ形式で扱われる。そのため，記憶効率や処理効率の面から大規模化が困難である。大規模化を達成するには，2つの方法が考えられる。第1は，上に述べた高い機能をもった知識ベースの枠組の中で，ソフトウェアおよびハードウェアの改良によって，容量・速度の両面での改善を図る方法である。たとえば，ハッシングを用いたり，あるいは連想メモリを用いたりすることが考えられる。第2は，個別データの蓄積・管理に従来のデータベースの手法が援用できることから，その部分を別に管理し，上位の知識ベースとの結合を図る方法である。この方式を確立することは，知識ベース・システムを通して既存のデータベースを利用する上で重要であり，一般には分散データベースとして知識ベースからこれらにアクセスする方

式を研究する。具体的には，項目(a)で開発される知識ベース検索言語を拡張し，既存のデータベース・アクセスをも含むようにする。

(d) 知識ベース管理システムの開発

項目(a)の知識表現・利用技術の研究で開発される知識表現言語並びに検索言語をサポートする知識ベース管理システムの開発を行う。知識ベース自身は，多くの種類のものが，各所に分散されて蓄えられる分散型となることを想定し，その管理もディレクトリ管理によって行う。また，知識ベース全体が矛盾を含まない一つの世界を表わしているのでなく，仮説的な世界をも含めた，互いに相矛盾するようないくつかの世界を同時に表現できることも必要である。これらの多世界の管理も同時に行う。

さらに，項目(c)の研究成果を実現するために，知識ベースと一般の分散データベースを結合した大規模知識ベース管理システムのプロト・タイプを開発する。

(e) 知識ベース・マシンの開発

項目(a)および(c)の研究によって得られる知識表現および検索言語のための高級言語マシンを作る。これは，データベース・マシンを高度化したものと考えることができる。その成分として，意味ネットワーク・プロセッサ，あるいは，よりデータベース寄りのマシンである関係代数マシンなどが考えられる。

本研究は，項目(d)の知識ベース管理システムの開発および使用，評価の後に，ハードウェア化によるメリットを吟味した上で行う。知識ベース・マシンの開発は，本プロジェクトの中でも，最も重要な課題の1つである。さらに，最終的には本マシンと推論マシンの融合化，統合化が図られねばならない。

(2) 研究開発スケジュール

本研究課題では，前期においては，開発支援システムの一部として，研究開発パーソナル・コンピュータ上に，小規模知識ベース用の知識ベース管理システムおよび大規模データ用のデータベース管理システムを作る。それは，プロダクション・システム，ＫＲＬ，セマンティック・ネットワークなどの知識表現言語をサポートするシステムである。

さらに，新しい知識表現言語の開発を目差して，知識の表現方式，利用手法，獲得・学習手法の研究を行う。また，分散した知識の管理を含む大規模知識ベースの構築技術についての研究を行う。

研究の重点は，基礎的な理論についての体系化，中期に開発すべき知識ベース管理システムの仕様決定のための基礎データの収集におく。また知識表現言語および知識ベース検索言語を中心とする知識ベース管理システムの第一次仕様を決定する。

中期では，前期収集したデータと大規模知識ベースの構築技術に基づき，関係型等のデータベース・マシンを土台にして，知識ベース・マシンのプロトタイプを試作する。さらに，前期

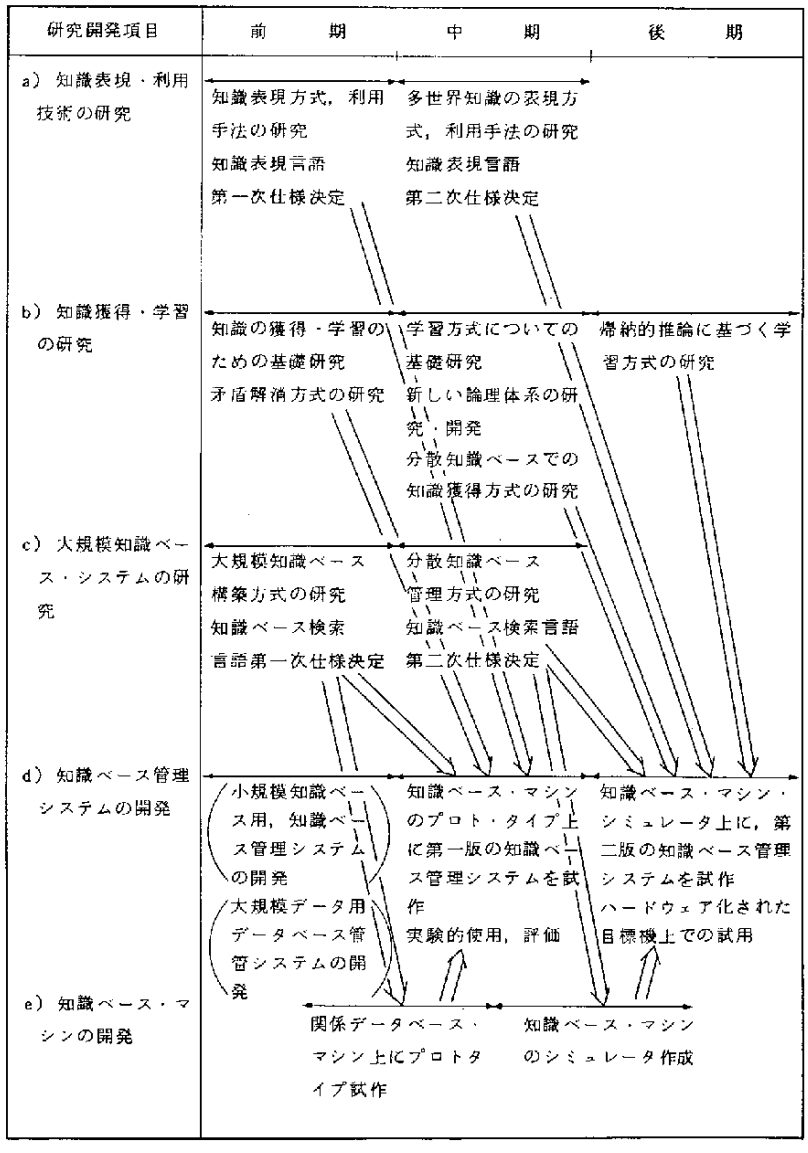
で決定した知識表現言語および知識ベース検索言語をサポートする知識ベース管理システムをこのプロトタイプ上で試作し，実験的に使用し，システムの評価を行う。

基礎研究としては，分散知識ベース系の管理方式の研究と，そこでの知識獲得方式の研究を行う。さらに学習方式についての基礎研究を前期に引き続いて行い，新しい論理体系の可能性と，その体系に基づく知識表現方式の研究開発を行う。

上に述べたシステムの評価と，基礎研究の成果をもとに，後期で開発すべき知識ベース・マシンおよび知識ベース管理システム（知識表現言語および知識ベース検索言語を含む）の第２次仕様を決定する。

後期では，中期で決められた仕様に基づき目標とする知識ベース・マシンのシミュレータを作成し，その上で知識ベース管理システムを開発する。この開発の過程においても，適宜，仕様の修正，精密化をすすめる。このシミュレータは，ハードウェアの開発に利用する。

表３－１　知識ベース管理システムの研究開発スケジュール



最後に，ハードウェア化された目標機と結合し，応用システムの試用に供する。また，応用システムの試用を通して得たデータに基づき，より高度な学習機能などの付加を目差した拡充のための研究開発を行う。

(3) 研究開発の目標・仕様

前期の研究開発用知識ベース管理システムの研究開発の目標・仕様は，S分科会報告書システム化技術の部分を参照されたい。

以下に，前期研究開発項目(d)および(e)について，その目標・仕様を中期と後期に分けてまとめる。なお，(e)の知識ベース・マシンの研究は，他の研究項目との関係で，中期・後期共に1～2年先行して開始される。

研究開発項目	中期目標・仕様	後期目標・仕様
知識ベース管理システムの開発	ルールとデータの同時管理。 データベース・アクセスの最適化機構。 矛盾解消機構。 推論マシンとのインタフェース。	多世界知識ベース。 分散知識ベース。 帰納的推論に基づく学習。 推論マシンとの融合化。
知識ベース・マシンの開発	2000個のルールと，100万項目のデータを収容検索可能。 （1項目 10^3 B）	20000個のルールと1億項目のデータを収容検索可能。 （100 GB）。

3.2.2 問題解決・推論システム

〔1〕 概　要

第5世代コンピュータ・システムの機能は現世代コンピュータのもつ機能とは質的に異なったものとなる。指示された手順のみに従った融通性のない処理から，与えられた問題を理解し，自らその解決へ向っての推論を進めながら処理を行なう。いわゆる知能的処理機能を有するようになる。

このように知能化された高度な機能を現世代コンピュータ上に実現することには，ソフトウェア，ハードウェアの両面からみて無理があり，問題解決・推論システムを核に据えたコンピュータ・システムを新たな視点に立って考え，その制御機構，処理形態，マシンの構造等を明確にしていかなければならない。

問題解決・推論システムは次の2点で重要な意味をもっている。

(1) 第5世代コンピュータ・システムに於ける基本応用システム，各種応用システムの構築

者，或いは一般利用者（エンドユーザ）のためのプログラミング環境を提供する。

(2) 知識情報処理に於ける基本的処理構造を明確化し，その仕様を第5世代コンピュータ・ハードウェアへの要求条件として提供する。このように，問題解決・推論システムは第5世代コンピュータの処理系の基礎を与えるものであるが，さらに，知識ベース管理システムでの管理系の記述，知識表現法の研究，或いは知的インタフェース・システムでの自然言語の処理系記述等にも密接に関連するものである。したがって問題解決・推論システムのもつ機能が第5世代コンピュータ・システムの提供する知識情報処理の質を規定すると言っても過言ではない。

問題解決・推論システムの研究開発項目は次の様に設定される。

(a) 問題解決・推論アルゴリズムの研究

(b) 問題解決向記述言語の研究

(c) 推論マシンの開発

推論アルゴリズムの研究は，演繹的推論の高速化，帰納的推論の確立等問題解決・推論システムが基礎におくアルゴリズムを体系化するものである。問題解決向記述言語の開発は，第5世代コンピュータ・システム上でのソフトウェア開発のための記述手段を開発するものである。問題解決向記述言語は推論アルゴリズムをベースに，より直観的なわかり易い問題記述機能を埋め込んだ言語・記述系として体系化される。推論マシンの開発では，問題解決向記述言語で書かれた問題解決・推論プログラムを効率的に実行するマシンのハードウェアベースを確立し，第5世代コンピュータのアーキテクチャを体系化する。

上記3つの研究開発項目の研究開発は理論的モデルの設定と現実システムの構築，実証の2面から進めていかなければならない。

(a) 推論アルゴリズムの研究

推論はシステムが自ら与えられた問題の解（ゴール）を求めて処理を進める過程である。推論では述語論理等で表わされた命題の真偽を判定する証明過程が基本である。システムは知識として蓄えている公理・定理を用いて演繹的推論，或いは帰納的推論を進め，与えられた命題の真偽を判定したり与えられた命題を真ならしめるように動作する。このような推論動作過程及びそれによりもたらされる効果が問題解決・推論システムでの処理である。

推論に基づく処理に於いては，従来のデータ処理とは異なり求めるべき解へ至る手順とは陽に示されない。システムは発見的方法或いは試行錯誤的方法により自ら解へ至る道を求めて処理を進める。

推論アルゴリズムの研究では，的確な発見的手法を求めること，出来るだけ無駄な試行錯誤をなくす推論法則を求めること，試行錯誤の制御を効率化する方策を求めること，推論過程での必要な公理，定理を早く見つけ出す探索手法を求めること等，とくに演繹的推論を高速化するアルゴリズムの研究が重要である。

人間の特徴である不完全な知識に基づく推論は，帰納的推論の一種であり，今後中心的な研究を進めて行く必要がある。これはデータベース或いは知識ベースの無矛盾性チェックの問題とも関連する重要なテーマである。

(b) 問題解決向記述言語の開発

問題解決向記述言語の開発は第５世代コンピュータ上で基本応用システム，並びに各種の応用システムを構築していく際のソフトウェア言語インタフェースとして重要な意味をもっている。

プログラミング言語の研究・開発はこれまで勢力的に展開されＰＬ／Ｉ，ＡＬＧＯＬ６８を頂点として現世代コンピュータ上でのプログラミングの概念，言語体系が一応確立したように思える。しかし，現在の言語体系はあくまで，"どのように"処理するかを記述する手続型記述言語であり，しかもその処理記述は記憶概念に基礎を置いたものである。

知識情報処理を指向する第５世代コンピュータ上でのプログラミングは推論に基づく問題解決法を記述するものであるから，そこでは"なにを求める"かを記述する非手続的記述を主体とする言語が必要となる。したがって第５世代のプログラム言語には大きな質的変革が望まれることになるが，この変革に向けての芽は，ここ１０年間の人工知能研究やプログラム基礎論での研究成果の蓄積，関数型言語および論理型言語，人工知能向言語，抽象データ型言語，等での具体的な提案や使用経験の蓄積により次第に育ってきている。

問題解決向記述言語の開発というテーマのもとで研究開発される言語には次の様なものがある。

① 核言語
② 意味記述言語
③ 仕様記述言語
④ システム記述言語
⑤ エンドユーザ向言語

このうち意味記述言語，仕様記述言語は知識ベース管理システム，知的インタフェース・システム，知的プログラミング・システムと密接に関連するものであり，それぞれのシステムのもとで詳しく説明されるのでここでは簡単にふれる。ここで特に注意すべきことは上記の各言語が個々ばらばらに設計されるのではなく，全てに共通の基本的な機構があり，その上にそれぞれの目的に合った言語が設計されるということである。

これらの記述言語に共通する言語構造は，関数概念に基づくプログラミングを指向した関数型言語の構造と，手続的記述を排除して述語論理に基づくプログラミングを指向した論理型言語の構造とが融合されたものである。さらにそこには，(工芸的でなく)工学的手法に基づく合理的なソフトウェア生産をサポートするための基本概念であるモジュラー化プログラ

ミングを実現させる種々の機能が盛り込まれることになる。

(c) 推論マシンの開発

知識情報処理向マシンは，問題解決・推論システムを現実化するためのハードウェア機構である。知識情報処理では推論が基本動作であるゆえ，現世代コンピュータに比してそのハードウェア機構は相当高度なものが要求される。更に試行錯誤的，非決定的実行が必然であると考えられるため，並列処理機構の導入等による高速化が必要とされる。

推論マシンは問題解決向記述言語，特に核言語に適合した高級言語マシンとして設計し，言語のもつ基本構造（セマンティクス）とマシンが備える制御機構との間に大きな差異（セマンティック・ギャップ）が生じない様にしなければならない。この様に設定された推論指向の基本機構の上に，各種の応用に沿ってそれぞれの専用マシンが，ソフトウェアによる応用システムの構築と融合しながら開発されることなる。推論マシンの"エンジン"部とも言うべきものは，推論実行制御機構である。これを推論エンジンと呼ぶ。推論エンジンは更にその基本的構成要素として推論過程での定理の探索に欠かせないパターン照合機構，試行錯誤的実行の制御に欠かせないバック・トラック機構，非決定性制御機構等があり，それぞれ重要研究テーマである。

一方マシンを実現するための要素技術としては，VLSI 技術の進展，データフロー・モデルやリダクション・モデルに基づく並列処理，分散処理概念の明確化が期待される。この様な部品，材料技術の進展とマシン・モデルの理論的進展とが融合されて，推論マシンを核に据えた第5世代コンピュータ・アーキテクチャが確立されることになる。

〔2〕 必要性

概要の項で第5世代コンピュータ・システムに於ける問題解決・推論システムの位置づけ及びその重要性についてふれたので，ここではその研究・開発の必要性を整理して述べることにする。

問題解決・推論システムの研究は第5世代コンピュータに於いて処理機能の明確化という面で中核をなす。即ち，その処理モデルの設定により理論的に処理能力を明らかにすると同時に実際にシステムの実現に向けての基本技術確立の基盤を与えるという極めて重要な位置をもつ。

第5世代コンピュータがどの程度の知能的処理能力をもち得るか，そしてこの第5世代コンピュータの上に築かれた知識情報処理システムがどの程度の"やわらかさ"を発揮して使い易いシステムとなるかは，その核となる部分のもつ処理能力に依存する。したがってその核部分のもつ能力は出来るだけ理論的モデルに基づいて明確化されなければならず，そのための理論的基盤を確立しておくことが必要である。

一方システム実現の見地からは，効率的な推論アルゴリズムの開発は不可欠なテーマである。またそのアルゴリズムをベースとした処理をわかり易く表現する記述法，言語システムの開発

は，各種の応用システムを開発する際のソフトウェア生産性を高める上で不可欠なテーマである。更に問題解決向マシンの開発は推論アルゴリズムの効率的実現のためのハードウェア基盤を確立し，問題解決向記述言語の処理系としてのハードウェア仕様を明確化して第５世代コンピュータ・アーキテクチャの要求条件を提示するという重要な位置をもつものである。

第５世代コンピュータは問題解決向記述言語および知識ベース管理システムが提供する仕様を対象とする高級言語マシンとしてそのアーキテクチャが設定される。また基本応用システム及び各種応用システムは問題解決向記述言語および知識ベース管理システムのもとで開発される知識表現言語により記述されることになる。したがって第５世代コンピュータ・システムに於いて中核となる記述系が問題解決向記述言語と知識表現言語ということになる。

情報処理システムを総合化された体系として機能させるためには，システムの構造とその制御・処理の記述系が一体化されなければならない。即ちマシンの構造とその上での記述系がある基本的機構のもとで統合化され一体となったものでなければならない。そしてこの統合・一体化のためにハードウェアとソフトウェア間の橋渡しの役目を果たすのが，問題解決・推論システムである。

〔３〕　内外技術の現状

(1)　国内技術

(a)　推論アルゴリズムの研究

人工知能研究の一環として，定理証明プログラム，知識ベース上での推論アルゴリズム，ゲーム・プログラム等の研究が電総研，武蔵野通研等の国公立研究機関，および東大，京大，阪大等の大学で行なわれているが，研究対象自体の規模は余り大きなものでなく，創造的研究成果はまだ目立ったものがない。しかし今後の研究発展のための基盤は蓄積されつつあると思われる。

(b)　問題解決向記述言語の開発

電総研，通研，東大，京大をはじめとする多くの研究所・大学で新汎用言語の研究・開発の努力が続けられいくつかの成果が得られている。しかし新しいプログラム言語（記述系）を考案し，世の中に広めていくという組織立った言語開発の試みはＡＬＧＯＬ　Ｎの研究開発があるという程度で欧米に比べて大きく遅れている。今までは極少数の例外を除いて，ほとんどが輸入によって賄われてきたといってもよく，日本人による言語開発を悲観視する風潮すらある。

しかし，最近の新言語研究の国内状況を見ると，電総研，通研をはじめとする研究機関，東大，京大，等の大学での人工知能研究グループのＬｉｓｐ 処理系の開発およびその使用経験の蓄積がなされてきており，関数型プログラミングの技法確立のための基盤が固められつつある。特にデータフロー処理概念との融合により新たに関数型プログラミング技法，および

その記述言語設計に対する関心が，電総研，武蔵野通研，東大等を中心に高まりつつある。

また論理型プログラミングの研究はPROLOG 処理系の移植・実現により，その使用経験を踏むという形でその第一歩を踏み出した状況にあると言える。

(c) 問題解決向マシンの開発

問題解決向マシンの開発そのものは今後の研究テーマであり，国内外共に未だ目立った研究の動きはない。しかし問題解決マシンへ繋がる研究としてはLisp マシンの開発，データフローマシンの開発を上げることが出来る。Lisp マシンは電総研，武蔵野通研，京大，神戸大，慶大で研究が進められている。

データフローマシンの研究は武蔵野通研，電総研，東大等で研究が進められており，欧米で提示された処理概念の後追い的状態から抜け出して，新たな処理領域の拡大とそれに伴った処理概念の創出等，独自の研究ステップへ踏み出す素地が出来つつあると思われる。特にデータフロー処理概念を推論機構と融合させて知識情報処理向マシンのアーキテクチャ基盤として捉えている点は日本独自の新規な発想であると評価される。

(2) 海外技術

(a) 問題解決・推論アルゴリズムの研究

問題解決・推論アルゴリズムの研究はCMU，Stanford 大学等においてProduction System で，或いはRule-based System 研究の一環で新ルール適用の方策を最適化する研究として進められている。推論アルゴリズムに数学的に明確な基礎が与えられたのはJ．A. Robinson による導出原理の発見（1965）があってからである。以後推論アルゴリズムを実現する単純簡潔なプログラムが計算機上に作成されるようになり，導出原理による演繹手順の効率化をめざした各種の導出手法が考案されている。導出原理による推論アルゴリズムが言語の概念にまで昇華したのはLondon 大学のR. Kowalski によるPROLOGの提案があってからである。

他に自然言語の意味論の研究分野ではKRL上での推論，Semantic Network 上での推論のアルゴリズム研究も進められているがまだ目立った成果はない。

最近，より柔軟な推論システムを指向するアルゴリズム研究としてNon Monotonic Logic の研究が注目されてきている。

(b) 問題解決向記述言語の開発

言語の開発は欧米で極めて活発に行なわれている。

まず関数型言語についてはLisp が代表的であり，欧米の多くの大学，研究所の人工知能研究者はLisp を常用語としている。Lisp にはいくつかの方言があるが，MITのMaclisp，BBN，Xerox PARCで開発されたInter Lisp，California大学Irvine校のUCI Lispが公用語的存在である。IBMで開発されたAPL言語も関数型言語の特性をもつ言語である。

関数型言語の開発は，最近別の観点から注目されている。ＩＢＭのＪ．Backus はソフトウェア工学の見地からＦＰ（Functional Program）とそのコンピュータ・アーキテクチャの必要性を説いた。以来非ノイマン型マシンと関数型言語の融合性が強く意識されるようになりリダクション・マシン，データフローマシンを意識した関数型言語の研究開発が進められている。後者の例としてＭＩＴのＶＡＬ，California大学Irvine校のＩｄ等がある。

論理型言語についてはＰＲＯＬＯＧがあり，仏のMarseille大学，英のEdinburgh大学，カナダのWaterloo大学等で処理系が開発されている。

抽象データ型の対象指向言語はSimula 67 に端を発す。現在では，モジュラー化プログラミングの概念を実現する言語として各所で言語および処理系の開発が進められている。Xerox PARC の Mesa，Smalltalk，ＭＩＴのＣＬＵ，CMU の Alphard，それに米国防省の標準言語Ada等がある。

(c) 問題解決向マシンの開発

問題解決向マシンの開発として直接の動きはまだないが，最近のマシン・アーキテクチャの技術として注目されるのはＶＬＳＩ 技術の進展に支えられて，パーソナル・コンピュータ開発の動きが活発なことである。Xerox ＰＡＲＣのＡＬＴＯ，ＤＯＲＡＤＯ，Three Rivers社のＰＥＲＱ，ＣＭＵのＳＰＩＣＥ,Intel の ｉＡＰＸ４３２などがある。ＡＬＴＯ，ＤＯＲＡＤＯはMesa，Lisp，Smalltalkを，またＳＰＩＣＥ,InteliAPX432はAdaを搭載言語としている。

記号処理マシンの開発としてはＭＩＴやＣＭＵでのLispマシンの開発がある。ＭＩＴのLispマシンは何回かのVersionの進歩があり，パーソナルLisp マシンとして商品化されるまでに至っている。

非ノイマン型マシンの研究はリダクション・マシンとデータフローマシンの研究が活発である。リダクション・マシンとしては独GMDのリダクション・マシン実験機，米North Carolina 大学の分散型構造リダクション・マシンの構想等がある。またデータフローマシンの研究はＭＩＴ，Utah大学，California大学Irvine校，英のManchester大学，仏のToulouse大学等で進められている。

〔４〕 研究開発内容

(1) 研究開発項目

(a) 問題解決・推論アルゴリズムの研究

（ａ．１） 理論モデルの研究

推論機能を基礎とする処理方式の諸特性を明らかにし，その性能評価を客観化することを目的とする。本研究は第５世代コンピュータ・システムの理論的ベースを与えるものであり，したがって本研究はプロジェクト進行中全期間にわたって続けなければならない。

本研究は，演繹論理，帰納論理，様相論理等の論理学や計算意味論，計算の理論，言語理

論，オートマトン理論等の観点から進めることになる。

（a．2） 高速化推論アルゴリズムの研究

問題解決・推論をアルゴリズム面で高速化することはシステム実現の観点から特に重要である。演繹的推論の機構は基本的であると考えられ，演繹に基づく問題解決過程を高速化するアルゴリズムの研究が必要である。そのためには問題解決の並列実行，無駄のないバック・トラック法，大規模データの高速検索などの研究を進めていく必要がある。

不完全な知識に基づく推論は帰納的推論の一種であり，システムにより高度な処理機能をもたらすためには，今後積極的に研究を進めていかなければならない。この方向の研究の進め方のひとつとしてTruth Maintenanceの概念に基づく推論，Default Reasoningによる意味の推察とそれに基づく仮説的推論などNon Monotonic Logicによる推論アルゴリズムの研究をあげることが出来る。

(b) 問題解決向記述言語の開発

（b．1） 核言語の開発

核言語はその名の通り，第5世代コンピュータがもつ基本的な言語インタフェースである。核言語が基礎となってさらに上位の言語仕様が定められる。また各種システム構築のための処理プログラムは核言語によって記述される。核言語は低位の言語インタフェースではあるが，現世代の高級言語に比べてはるかに高い記述機能をもつものである。

核言語の基本的構造は論理型と関数型とが融合したものとして設定される。さらに，対象指向の要素を強く反映し，モジュラー化プログラミングによるソフトウェアの高生産性を配慮した言語仕様となる。

核言語の仕様設定に当っては次の各項を設計要因として考慮しなければならない。

① 知識ベースとの融合

知識ベースへのアクセスと，それから得たデータの処理とは一体化されなければならない。したがって核言語仕様設定に際しては知識表現系の構造，知識ベースへのアクセス記述等を考慮しなければならない。

② 記号処理（高度なデータ構造と演算）

核言語は従来の数値計算指向の言語系とは異なり記号処理（非数値データ処理）指向の言語系として設定される。推論を基礎として知識情報を処理するためには高度のデータ構造を扱かえることが必要である。基本データ型として，リスト，対，集合等が含まれる。さらにデータ抽象化機能により任意の構造のデータ型を定義することが出来なければならない。

③ 並列処理

核言語は関数型と論理型の融合した構造である。言語のもつ基本的意味構造としては従来の逐次処理概念に代わって，並列処理概念に立脚したものでなければならない。さらに加え

て陽に並列処理，直列処理を表現するための高機能な表現要素を持つことが要求される。

④　非決定性処理

核言語では知識ベースへのアクセス記述や推論動作の記述に於て非手続的表現が出来なければならない。非手続的処理ではパターン照合により，複数個のゴールが非決定的に選択される。したがって非手続的記述ではパターン照合による非決定性処理の記述が言語仕様化されなければならない。

非決定性処理の記述は並列処理概念を基礎とし，これにバック・トラック制御の記述を融合させた形で言語仕様化されることになる。

⑤　対象指向（抽象化技法）

大規模ソフトウェアを構築していくには，単に論理型と関数型との融合だけでは不十分である。最近のソフトウェア工学に於いて対象指向言語によるモジュラー化プログラミングの手法が提案され，その有効性が検証されつつある。

核言語設計に於いても，このモジュラー化プログラミングが可能な仕様を設定することが必要である。

抽象化技法によりプログラムをモジュール化し，これを知識ベース化することが出来れば半自動的なプログラム生成が可能になり，ＣＡＤによるソフトウェア生産への繋がりが出来ることになる。（知的プログラミング・システムの項を参照）

⑥　プログラミング・システム化

当然の事ではあるか核言語は言語として単独に存在するだけでは用をなさない。コンパイラ，インタプリタ，デバッグ支援システム，検証システム，エディタが一体化したシステムとして知識ベース化され，知識ベース管理システムの下で機能しなければならない。

（ｂ．２）意味記述言語

意味記述言語は核言語をはじめ，種々の言語の意味を形式的に記述する言語である。また知識ベースにおける知識表現のための記述言語，自然言語処理における意味の形式的表現，意味辞書記述等に使用される。人工言語での意味記述と自然言語，知識表現での意味記述とでは多少異なる面もあるか，機械的処理の観点からは，両者を統一した形式的記述手段が要求される。

意味記述の方法としては操作的，指示的，公理論的等種々あるか，各手法を比較検討し，より広い適用性をもつ記述法を確立していかなければならない。

知識ベース，自然言語処理での意味記述言語仕様については知識ベース管理システム，知的インタフェース・システム，知的プログラミング・システム，質問応答システムの各研究との連絡を密にしながら設計を進めていく必要がある。

（b．3） 仕様記述言語

各種システム（ハードウェアも含めて）の仕様を厳密に記述するための言語である。仕様記述には機能仕様記述と構成仕様記述がある。機能仕様には高位の抽象度が高いものから下位の詳細厳密化された仕様までいくつかのレベルが考えられる。構成仕様は対象がソフトウェアかハードウェアかにより若干異なると思われるが核言語に多少の変更を加えることにより流用できると考えられる。

データフローの概念を基礎にして仕様記述を考えると，原理的にハードウェア，ソフトウェアの境界がなくなり，さらに抽象化技法を持ち込むことによって機能仕様記述と構成仕様記述との間に連続性が出てくると期待される。

知的CADによってシステムを設計する場合，知識ベースに蓄えられたシステム構成部品群の中から要求された仕様を満足する部品が検索される。したがって仕様記述法の確立及びそのための記述言語の問題は知識ベースにおける知識表現法とも関連する。またシステムの検証は仕様記述に基づいて自動的に行なわれることになる。このように仕様記述言語は知識ベース管理システム，知的プログラミング・システム，知的CAD／CAMシステムと密接に関連する。

（b．4） システム記述言語

システム記述言語は基本応用システム，各種応用システムを構築する際のシステム設計用記述言語である。

システム記述言語には核言語がほとんどそのまま当てられるが，特に，より効率的なシステムの設計を望む設計者のためにハードウェアに近い記述要素を取り入れた言語仕様が設定される。

（b．5） エンドユーザ言語

一般の利用者にはその用途，適用分野によって種々の言語が提供されることになる。第5世代コンピュータ上での自由闊達なプログラミングのためには核言語が使用される。知識ベースへのアクセス，質問応答システムでは図形言語，表言語，自然言語等が用いられる。これらの各言語の仕様は各種の応用システムのもとで定められるべきものである。

(c) 推論マシンの開発

この研究開発項目は，第5世代コンピュータ・アーキテクチャ開発のための理論的基盤を固めると同時に，問題解決・推論システムの持つべき機能を整理し，ハードウェアへの要求条件を明確化するものである。本研究で示された要求条件に沿ってのアーキテクチャの具体化が，アーキテクチャ関連課題の研究で詳細に展開されるのでここでは基礎理論研究とアーキテクチャ研究とのインタフェースという意味で簡単にふれることにする。

（c．1） 計算機構理論

推論処理を高速化するためには，並列処理，分散処理，連想処理，非決定性処理を盛り込んだ処理概念とその計算機構を明らかにしなければならない。

計算機構をモデル化して捉える場合，そのモデルが現実のアーキテクチャと遊離したものであってはならない。VLSI による高機能化素子等いくつかのアーキテクチャ構成要因を考慮した上で計算機構のモデルが立てられる。

計算機構としては関数型，論理型のものがモデル化され，これらを融合する計算機構が設定されなければならない。さらに，抽象データ型を処理する計算機構へと拡張されなければならない。現在リダクション・マシン・モデル，データフローマシン・モデルが VLSI 技術を前提として高度の並列処理，分散処理を実現する計算機構，モデルとして有望であると考えられている。今後は，論理型計算機構並びに高度のデータ構造処理機構がこのモデル上でうまく機能するかという問題を解決していかなければならない。

（c．2） 推論エンジン

問題解決・推論システムの"エンジン"部である推論エンジンのハードウェア実現は第5世代コンピュータ・アーキテクチャにとって中核的問題と言える。推論エンジンの実現に当っては次の構成要素の開発を進めなければならない。

① 高速パターン照合機構：単なる文字列照合では不十分である。変数を含むパターンへの照合に際しては値の結合を行なう，またパターン中に関数が現われたときはその関数の評価を行なう，などの Unification機能をもたなければならない。Unification機能をどのレベルまでをハードウェア実現とし，どのレベルからソフトウェア実現とするかは設計上重要な問題である。

② 非決定性制御機構／バック・トラック機構：高度の並列処理機構をベースとするか，逐次処理をベースとするかで非決定性制御機構の実現方式は大きく異なる。逐次処理ベースではスタックを用いたバック･トラック制御が大きな比重を占める。一方並列処理ベースでは複数の選択枝が同時に実行され，有効な結果だけが残されるという実行制御がとられるからバック・トラックの比重は小さくなる。（すなわち，バック・トラック制御は本質的には不要となるが，現実には選択枝の広がりを制限することが必要であり，バック・トラック機構を排除するわけにはいかないと思われる。）

③ メモリ機構：推論システムに与えられた公理，定理等の推論規則或いは PROLOGのような論理型プログラム，推論実行中のデータを格納するメモリ機構とその装置構成も重要な問題である。推論規則や論理プログラムはパターン照合の対象となるから，これを格納するメモリ装置とパターン照合装置とは一体化させて設計を進める必要がある。

推論マシンはこれらの各構成要素から組み立てられる。マシン構成に際しては， VLSI

素子を駆使して推論マシン各部品の機能を分散化し，システムの制御はデータフロー概念に基づいて実現されるということになると考えられる。

(2) 研究開発スケジュール

推論マシンの開発は次の2面から進めなければならない。

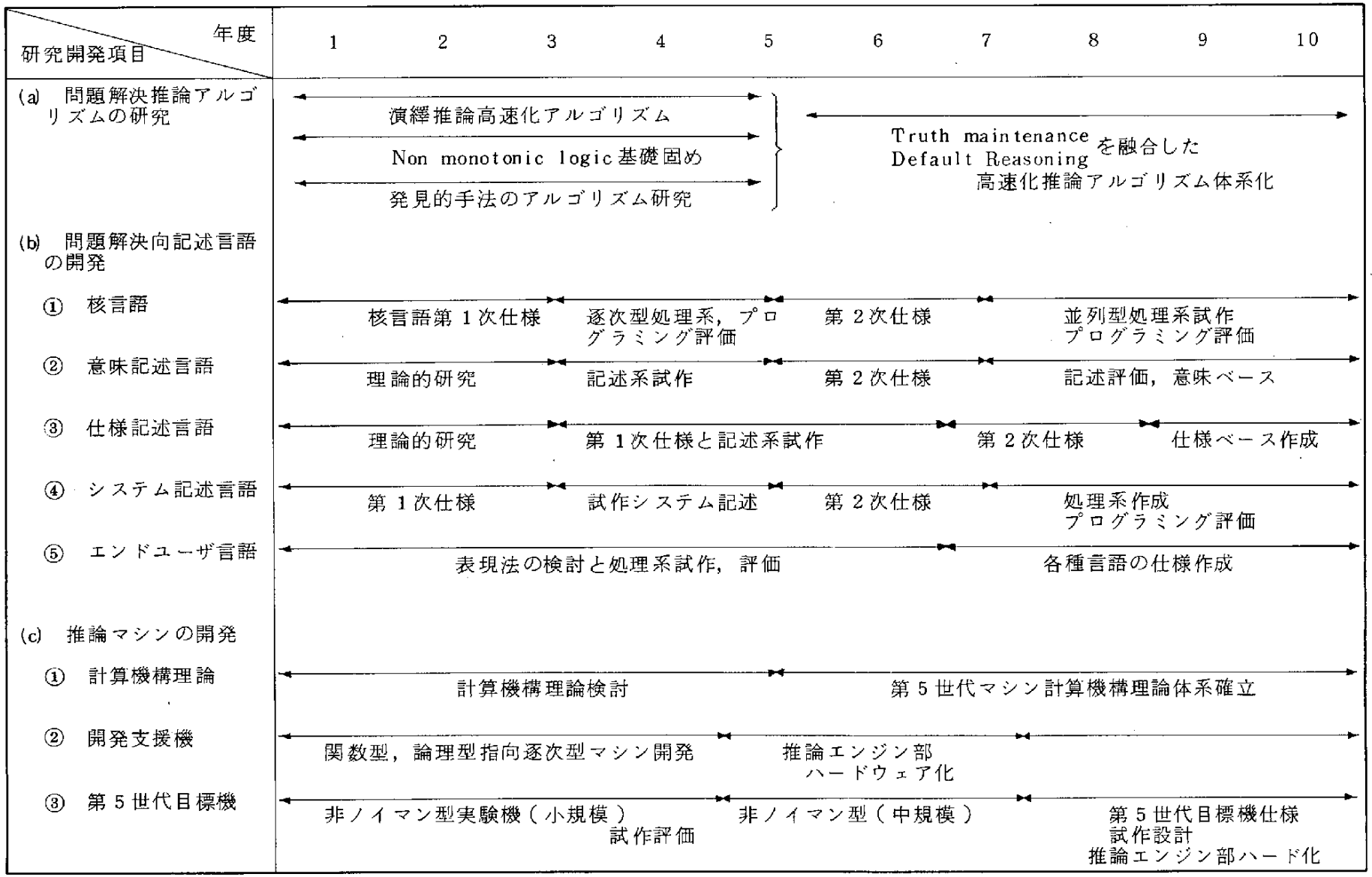
① 開発支援機の開発：現在のコンピュータ・アーキテクチャとそれ程の開きがない実現可能なアーキテクチャで記号処理専用マシンを開発し，その上でソフトウェアにより知識情報処理システムの縮小モデルを構築して各種評価を行なう。このマシンは現在の Lisp マシンを発展させた記号処理向逐次型マシンである。このマシンは第5世代コンピュータへ至る途中の踏台的意味をもつ。ソフトウェア・言語，各種応用システムの評価実験はこのマシンを道具としてくりかえされる。

また推論マシン各部のハードウェア化とその評価実験もこのマシンをベースとして行なわれる。

② 第5世代目標機の開発：第5世代のコンピュータ・アーキテクチャの開発を革新的概念に沿って進める。当初から並列処理，連想処理を指向して，ノイマン原理に基づかないアーキテクチャの確立をボトムアップ的に進める。まずは小規模な実験システムによる原理的な動作検証と性能評価実験を進め，次第に処理概念を拡張しつつ実験機の規模を拡大していく。規模を拡大していく段階で①の逐次型マシン上で開発されたソフトウェアを実験機上に移植して行き評価実験の規模を拡大すると同時にソフトウェア開発を逐次型マシンから並列処理マシンと変えて行く。

具体的な研究開発スケジュールとしては例えば，次の様な設定が考えられる。

まずデータフローマシンの実験機を試作する。この実験機は小規模なものであり，関数型言語によるプログラミングがサポートされる。この段階では論理型プログラムに対するサポートはデータフローマシ上でのソフトウェア・インタプリタによる。（これは逐次型マシン上での論理型プログラムのソフトウェア・サポートに対応する。）後期スケジュールでは実験機の規模を拡大すると同時に，ソフトウェアでサポートされていた推論マシンの各部を徐々にハードウェア化してゆく。ハードウェア化に当っては逐次型マシン上で開発された部品を構成要素として用いることも考慮しつつ，データフローマシン上での実験評価をくりかえしてゆく。そして最後にデータフローマシンを基礎とする問題解決向マシン・アーキテクチャのプロトタイプを確立する。当然この研究開発スケジュールの中には，抽象データ型をサポートするための基盤となるメモリ機構の開発も含まれる。


年度
研究開発項目
1
2
3
4
5
6
7
8
9
10
(a) 問題解決推論アルゴリズムの研究
演繹推論高速化アルゴリズム
Non monotonic logic基礎固め
発見的手法のアルゴリズム研究
Truth maintenance を融合した
Default Reasoning
高速化推論アルゴリズム体系化
(b) 問題解決向記述言語の開発
① 核言語
核言語第1次仕様
逐次型処理系，プログラミング評価
第2次仕様
並列型処理系試作
プログラミング評価
② 意味記述言語
理論的研究
記述系試作
第2次仕様
記述評価，意味ベース
③ 仕様記述言語
理論的研究
第1次仕様と記述系試作
第2次仕様
仕様ベース作成
④ システム記述言語
第1次仕様
試作システム記述
第2次仕様
処理系作成
プログラミング評価
⑤ エンドユーザ言語
表現法の検討と処理系試作，評価
各種言語の仕様作成
(c) 推論マシンの開発
① 計算機構理論
計算機構理論検討
第5世代マシン計算機構理論体系確立
② 開発支援機
関数型，論理型指向逐次型マシン開発
推論エンジン部
ハードウェア化
③ 第5世代目標機
非ノイマン型実験機（小規模）
試作評価
非ノイマン型（中規模）
第5世代目標機仕様
試作設計
推論エンジン部ハード化

(3) 研究開発の目標・仕様

第５世代目標機の性能は推論処理を高速に実行できること。高度の並列処理，連想処理を駆使したマシン構成により推論実行速度にして現世代マシンの $10^5 \sim 10^6$ 倍程度の性能を発揮すること。

推論マシンの性能指標として，ＬＩＰＳ（Logical Inference Per Second：推論実行速度）を単位にとると，現世代マシン $10^4 \sim 10^5$ ＬＩＰＳ 程度である。第５世代コンピュータでは $10^2 \sim 10^3$ Mega ＬＩＰＳ程度の性能を目標とする。

問題解決向記述言語系は関数型，並びに論理型プログラミングをサポートし，かつ対象指向に基づくモジュラー化プログラミングを可能とすること。生産されたプログラムモジュールは，ソフトウェア部品として知識ベース化され，知的プログラミングに於いて有効に利用されること。

3.2.3 知的インタフェース・システム

〔１〕 概　要

第５世代コンピュータの目標は，「使い易いコンピュータを実現する」ということである。ここでいう「使い易さ」の意味は，特定の（コンピュータの）専門家は言うにおよばず，一般の（コンピュータの素人である）使用者にとっても使い易いコンピュータを実現することでなければならない。

このような観点から現在のコンピュータを眺めてみると，「使い易さ」に対する機能が非常に不十分であると言わざるを得ない。その最大の要因は，（使用者である）人間とコンピュータとの使用言語の相違に基づくギャップにある。そこで，この様なギャップを解消するためには，コンピュータ・システムが柔軟な会話機能を持つこと，すなわち自然言語や音声，図形，画像による会話機能を持つ必要がある。第５世代コンピュータのなかの知的インタフェース・システムの果す第１の役割は，このような柔軟な会話機能を実現し，人間とコンピュータとの間の使用言語（これには自然言語や音声，図形，画像も含まれる）の相違に基づくギャップを解消することにある。

第２の役割は，知的インタフェースを実現することにより，我々の思考を支援し推進することが可能になる，という点にある。自動車のエンジンが，我々の歩行能力を推進拡大したのと同様に，第５世代コンピュータを，あたかも我々の思考能力を推進拡大するための“思考のエンジン”として機能させる必要がある。知的インタフェース・システムは，それを実現するための中核技術の一つである。第５世代コンピュータでは自然言語，音声，図形や画像は，単にコンピュータへのデータの入出力の手段と考えるのに留まらず，人間の高度な推論や判断をコンピュータが支援する際に，それを効果的に行うための手段として捉え，自然言語，音声，図

形，画像の表わす内容について理解し得るようなシステムを作り上げる技術の開発を狙っているのである。

知的インタフェース・システムは，自然言語・音声系と図形・画像系とに分かれる。自然言語・音声系は，基本応用システム中の音声応用システム，質問応答システム，そして機械翻訳システムとも密接に関連する。基礎ソフトウェア・システムとしての自然言語・音声系は，これら三者（音声応用システム，質問応答システム，機械翻訳システム）に共通する研究課題と，三者個別に，各々のシステムで本質的で中核となる研究課題となるものを取り上げて研究開発するものである。一方，図形・画像系は，基本応用システム中の質問応答システムおよび図形・画像応用システムのベースとなる諸技術を研究開発するものである。

自然言語関連では，我々人間が，日常の会話で用いている自然言語を用いてコンピュータと知的な対話を行うための諸技術の確立をする。この技術には，構文解析技術，意味解析技術，談話解析技術，文章合成技術，言語データ・ベースの構築が含まれる。以上の諸技術で開発した洗練された解析アルゴリズムを統合しハードウェア化した自然言語処理マシンの開発を行う。

音声応用システムでは，人間とコンピュータの間で"知識"が取り交わされるとき，キイボードからの入力やディスプレイの表示を読みとる作業よりも，自然音声による入出力の方が，はるかに能率が良く，しかも一般非専門家にとって使い易いものとなる。コンピュータに推論能力や問題解決能力が備わった時，いわゆる"常識"が組み込まれると，人間の間で交わされる不完全な文章も内容を補って理解できるようになる。このような状況では人間がコンピュータに，キイボードから"完全"な文章を入力する必要はなく，人間に対して話しかけるように，音声により適度に省略された自然な会話ができることが望ましい。

さて音声は，我々の通信手段の中で最も容易かつ効率良く情報を伝達できる方法である。言語情報は音声生成器官によって音声波に変換され，聴覚機構によって受容される。このシステムは，我々が幼時から年月をかけて修得あるもので，それら複雑な仕組みを解明して機械的に実現することは容易ではない。

1970 年代に入って音声認識の実用化の研究が開始され，社会的ニーズと相まって国家的プロジェクトのテーマとして取り挙げられるようになった。我が国では 1971 年からスタートした大型プロジェクト「パターン情報処理システム」の 1 テーマとして音声認識の研究が始められ，米国では時を同じくして ARPA による「音声理解システム」の研究が行なわれた。我が国の大型プロジェクトで培った音声認識技術では，実時間で単語を識別する音声認識システムが実用化し，基礎研究では，新しい方法による音声生成過程の推定や，音声分析レベルでの種々の研究成果があげられており，今後の音声認識研究の土台となるものである。

本節で述べる音声応用・知的インターフェースの中心テーマは，従来の音声認識研究を発展させて，音声理解システムを研究開発することである。すなわち，音声分析で得た音素記号列

（ある程度の誤りを含む）から，構文・意味・語用論的情報を用いてその意味内容を理解して文章として内容を把握することを目標とする。米国における「音声理解システム」開発プロジェクトの評価論文で指摘されているように，十分な音声分析技術をベースにしなければ，その上にいくら理解技術を上乗せしてもシステムとしての性能には限界がある。従って本研究開発の前期には，音素識別方式の研究が重要であって，併せて音声合成技術の基礎研究や，音声研究用共通データベースの作成等を行う。そして後期には知識情報処理技術を用いて言語情報をとり入れて音声理解研究を中心に行う。

図形・画像系では図形および画像という視覚情報を媒体として，人間がコンピュータと知的に対話するための技術を確立し，知的インタフェース・システムを提供する。このシステムは視覚情報に基づく推論機構，高精度高機能のカラー図形・画像入出力装置，図形・画像処理マシンおよび図形・画像ファイルより構成される。その機能は，人間が見ることによって情報を得たり他人とのコミュニケーションに使うような図形あるいは画像が表わす内容を理解してコンピュータの中の記述に変える機能と，コンピュータ内の記述を，人間が理解しやすい図形や画像で表示する機能の２つがその柱となる。

開発の方針は，図形・画像を理解するシステムを構築するための共通技術を研究開発する事に主力を置くとともに，知的インタフェース・システムを作るために図形・画像生成アルゴリズムの研究とシステム化技術の開発も併せて行う。また知的インタフェースを自然言語理解システムと組合せて構成するための技術開発も行う。

マンマシン・インターフェースを考える際，第一に考えなければならないのは，使い方の自然さと円滑さである。その実現をサポートするためのハードウェアの開発も重要な課題である。

本節で研究開発される自然言語や音声，図形・画像による知的インタフェース構成のための要素技術をもとに，使い易いコンピュータ実現のためのスマート端末を開発する。開発の方針は，研究開発期間の前半の内に，それまでに利用可能な要素技術を総合して，パイロット・モデル（前期）を作成し，試用評価を終て，後期のモデルを設計・製作することにする。パイロット・モデル開発の途中で各要素技術へのフィードバックを密に行い，本節における研究開発を有機的かつ効果的に推進する。

〔２〕　必要性

概要のところに，必要性が比較的詳しく述べられているので，ここでは，簡単に要約しておく。第一に，人間とコンピュータ間の使用言語の相違に基づくギャップを解消するためには，第５世代コンピュータに自然言語や音声，図形や画像を用いた柔軟な会話機能を付与する必要がある。

第二に，自動車のエンジンが我々の歩行能力を推進拡大した様に，第５世代コンピュータを，あたかも我々人間の思考能力を推進拡大させるための“思考のエンジン”として機能させる必

要がある。

知的インタフェース・システムは，以上を実現するための中核的基礎技術を確立するものであり，それにより，第５世代コンピュータに対する社会の広汎な分野の需要が一層喚起されることが期待できる。

自然言語関連による会話は，柔軟な会話機能と密接に関連することは明らかだろう。音声・図形・画像についてはどうだろうか。

第５世代コンピュータは専門家ばかりでなく，一般家庭も含めた非専門家ユーザが幅広く使用すると考えられる。その時，現在のようにプログラムをキーボードから入力する方式では，手続きの煩雑さがネックとなることが考えられるため，自然な会話形式でコンピュータへ音声入出力ができることが望ましい。また，第５世代コンピュータを開発する場においても，膨大なソフトウェアを生産性良く作成する必要があるため，知的プログラミングによるプログラムの自動作成及び自動検証と併せて，音声による入出力が必要とされる。

現在実用化されている音声認識システムは，単語単位の認識方式で，話者毎に認識対象単語を登録してパターン・マッチングによって識別を行う。数百程度の単語については実時間処理が行われているが，単語数を数千から数万に拡大すると，登録に要する手間や認識時間が実用的でなくなる。そこで認識単位を小さくして音素程度にすれば，登録すべき音素数は限られており，単語数に関する制約はほとんどない。認識単位を小さくすることに従って調音結合の処理，セグメンテーション等の研究が重要になってくる。

音素単位で認識した記号系列から単語や文章へと組み上げていくには，構文で意味等のいわゆる言語情報を利用する必要がある。しかし現在の数値計算向きコンピュータでは能率が悪く，言語情報の十分な利用が達成されていない。

また，人間とコンピュータの会話で重要な音声の合成では，現在は文節や単語単位で録音しておいた音声を編集して出力する方式が音質の最も良い合成音を生成する。しかし語彙数や記憶容量の限界から，合成のための単位を音節や音素にして，合成できる語彙数を大幅に拡大する必要がある。分析合成方式や規則による合成方式等の研究をさらに進める必要がある。

その他，個人差の学習方式や韻律情報の抽出方法を発展させて実用化する必要がある。

社会生活における情報処理活動の主体は，言うまでもなく人間である。コンピュータによる情報処理システムを設計（製作する場合に大きな役割を果すのは，従って人間の知的活動である。

新しい情報処理系を作り上げようとする場合に，この人間の知的活動を効果的に引き出すシステムが必要であることが，最近広く言われている。また，その可能性を示す実例も生み出されている。このようなコンピュータによる支援のプロセスの過程にあって，推論あるいは思考の手掛りを，"視覚化"する事が有効であることは容易に想像できる。

新しいシステムを作るような場合でなくとも，何かデータを分析しつつ判断を進めるような作業をコンピュータがサポートするような場合，同様のことが言える。

このような局面は従来CADの分野でしばしば取上げられて来ている。そこでの問題点は，使い易いシステムを決定し，設計製作することが非常に困難であり，時間と人手が必要であるという点であった。

一方，図形や画像を入力する局面について考えるならば，人間が理解できる図形や画像から，それに含まれる意味・内容をコンピュータが理解し，コンピュータの中でそれを後から使える形で記号化し記述する機能が必要である。これと，推論・問題解決のメカニズムとによって，図形や画像を介して，コンピュータが人間とコミュニケーションを行うことが可能となる。

このようなコミュニケーションがコンピュータと人間との間でできるようになると，人間の知的活動を効果的に引出す真の知識情報処理システムの実現が可能になる。従って，図形・画像による知的インタフェースを開発する必要性は非常に大である。

〔3〕 内外技術の現状

(1) 国内技術

(a) 自然言語・音声系

国内では，京都大学で機械翻訳システムの研究が，九州大学では，語彙の意味分析の延長上に，機械翻訳システムの実現が模索されている。最近では，東大，東工大，名大，北大等の大学にも研究グループが存在し，この分野の研究が一層活発化している。電総研では，自然言語理解のための諸技術の開発，また電々公社武蔵野通研では，質問応答の研究が行われている。国立国語研究所では，日本語の分析が進められている。日本IBMでは，質問応答システム"やちまた"を作動させている。東芝，富士通，日立，日電等にも研究グループが存在している。

音声認識の研究については，我が国において多くの秀れた成果があげられているが，ここでは音声理解システムについて主として述べる。

国内における音声理解の研究開発は，京都大学を始めとするいくつかの大学と，電電公社武蔵野通研を中心として行なわれている。

京都大学で研究開発されたLITHANシステムでは，101単語の語彙からなる「計算機網への状態問合せと指令」をタスクにしており，男性話者10名の発声した200文章に対して，文章認識率64%，単語認識率93%を得ている。今後は，個人差学習機能の強化と韻律情報の利用等による能力向上を目差している。

電電公社武蔵野通研で開発された，列車の座席予約をタスクとしたシステムでは語彙数112で文節認識率86%を得ている。

京都工繊大，山梨大でもBASICやFORTRANプログラムを音声入力するシステムを開

発し，メーカでは日本電気をはじめとして，1979年には富士通，東芝，日立，三洋電機が相次いで音声認識装置を発表している。しかし，音声理解システムとしての本格的な検討は未だ行なわれていない。

また，電総研では，音素識別方式の基礎として，調音結合の処理方式や子音の識別方式，セグメンテーション等について研究し，成果をあげている。

（b）図形・画像系

我国の図形・画像情報処理技術の水準は，世界的に見てもかなり高い。基礎から応用まで，多くの研究者，研究グループが研究を手掛けている。ハードウェアの低廉化がもたらす効果も大きい。すなわち，この分野の研究に着手する際，ハードウェアへの初期投資が以前に比べて低減し，大学の研究室レベルでの研究がやり易くなって来た事によって，研究者層の増大が促進されている。

技術水準について言えば，応用範囲が限られているとはいえ，各種実用システムが開発されて来ている。たとえば，郵便番号自動読取り装置の実用化は，歴史的にマイルストーンと言える。工場の生産ラインにおける画像処理システムの応用（LSI ボンディングの自動化等），医用画像への応用（白血球解析装置等），リモートセンシング画像処理など多数の応用システムが生み出されている。

このように，国内におけるこの分野の研究は非常に多い。

画像理解の研究が屋外風景や航空写真，医用画像などを対象に，京都大学，東京大学，山梨大学，電総研，東芝などで行われている。工業用ロボットの目としての画像情報処理の研究が大阪大学，電総研，日立などで行われている。

(2) 海外技術

（a）自然言語・音声系

海外では，1970年代初頭のMITの研究が著名であるが，その後，スタンフォード大で，自然言語理解システムの開発が行われ，その技術の延長がエール大で行われている。以上の大学は実用というより，自然言語理解のための基本技術の開発に重点を置いている。同様な立場から，BBNでは，構文解析のための武器としてATN方式を開発し，自然言語理解（音声理解も含む）システム実現に向けて種々の実験が進められている。

Xeroxでは，柔軟なマンマシン・インタフェースをもったパーソナル・コンピュータが，自然言語理解を含む新世代コンピュータを意識する立場から進められている。

自然言語による質問応答システムの実用化を目指す立場からは，ミシガン大，ダートマス大，カーネギーメロン大，SRI等で進められ一部実用に供されているシステムも出現している。

機械翻訳は，カナダ（モントリオール大），ヨーロッパ（グルノーブル大等）で活発化し

ている。パリ大では，談話理解に重点をおいている。このようなヨーロッパでの動きは，米国にも波及し，最近では，テキサス大に，機械翻訳のグループが再編されている。

ＩＢＭでは，サンノゼ研究所でデータベースに対する質問応答システムの研究開発が進められていた。最近になって，ワトソン研究所でも，従来のグループの他に自然言語理解のための新しい研究計画のあることが明らかにされた。

西独ではジーメンス社の自然言語理解システムが，政府の援助の下に進められている。

英国のエジンバラ大でもＰＲＯＬＯＧをベースにした自然言語理解システムの研究が行われている。

一方，音声研究は米国を中心に，スウェーデン，フランス，イギリス等で盛んに行われている。

米国では，国防省のＡＲＰＡ のサポートによって，1971 年から５か年計画で音声理解システム研究開発が行なわれた。当初８つの研究機関が参加したが，1973 年に３グループに統合され，1976 年８月に終了した。そのうち，カーネギーメロン大学が開発したＨＡＲＰＹシステムのみが当初の目標（適応化した多数話者の発声による，1000 語程度の語彙の連続音声を，意味的に１０％以下の誤り率で認識すること）を達成した。同大学では，ＨＡＲＰＹの前にＨＥＡＲ　ＳＡＹ－Ｉ，ＤＲＡＧＯＮ，ＨＥＡＳＡＹ－Ⅱ を開発している。最も結果の良かったＨＡＲＰＹシステムでは，５名の話者の発声による 184 文章に対し，文章理解率９５％が報告されている。タスクはニュース検索であった。

その他に，ＢＢＮ社のＳＰＥＥＣＨＬＩＳシステムとＨＷＩＭシステム，ＳＲＩとＳＤＣ グループによるシステムがＡＲＰＡ計画により作られた。いずれも文理解率は５０％以下であり，成功したとは言えないが，言語処理法や音声分析法等に成果があったと言われている。

ＡＲＰＡ計画以外では，ＩＢＭ，ＭＩＴ，ＵＮＩＶＡＣ，ＢＥＬＬ研究所等で音声研究が盛んである。ＩＢＭのシステムでは，各話者に対して１時間以上の音声を用いて話者適応後，250 語からなるタスクに対して文理解率９５％が得られている。

製品化された音声認識システムでは，いずれも単語認識方式を用いて Threshold Technology 社は最大２５０単語を９９％以上の認識率を得ており，Interstate Electronics 社はやはり２５０語のシステムを開発している。

（b）図形・画像系

米国の国防省がサポートする「Image Understanding」のプロジェクトが 1975 年から発足し，ＭＩＴ，ＣＭＵ，Stanford 大学等，米国の情報処理研究の主要な大学が研究を推進している。

Stanford 大学ではＡＣＲＯＮＹＭ という，三次元モデルを用いて物体を認識するシステムの開発を行っている。ＭＩＴ では，明るさの情報から形へ直接変換する研究を行っている。

CMU ではロボットの眼の開発を開始している。

〔4〕 研究開発内容

(1) 研究開発項目

(a) 自然言語・音声系

① 構文解析技術

入力された自然言語の文を構成する単語の認定（活用語尾処理を含む）を行ない（これは形態素解析ともよばれる），単語の品詞の並びが妥当なものかどうかを文法規則を用いて検査し，単語と単語の結び付きを示す構文構造を作り出す。この構文構造は，意味解釈をするための中間結果として次段階の意味解析で利用する。

以上の様な構文解析を行なう諸技術，特に高速アルゴリズムを開発しシステム化する。意味解析技術と構文解析技術を融合させる技術を開発する。

② 意味解析技術

構文解析により，単語の品詞の並びの妥当性が検査されたら，それが意味をなす文かどうかを調べ，言語の種類に比較的依存しない意味構造を抽出する技術が意味解析技術である。したがって，意味構造をどのような形式にするか，また語用論的な知識を意味解析でどのように利用すべきかをも考慮しなければならない。また構文解析技術と談話解析技術とを利用して意味解析を行う技術を開発する。

③ 談話解析技術

複数個の文の系列からなる談話（discourse）を解析する技術である。これには，省略語の補強，代名詞の指示するものの決定，主題の把握等の文脈解析技術が中心になるので，語用論的解析（pragmatic analysis），意味解析が関連する。以上の諸技術を研究開発する。

④ 文章合成技術

抽出された意味構造，もしくは推論・問題解決結果を自然言語に翻訳するための技術である。ここには，文法情報が主として用いられるが，自然言語として"自然"な文を出力するためには，代名詞化，適度の省略等が必要になる。そこで談話解析技術をも応用しなければならない。ここで，構文解析で用いる文法と別の文法を利用して文章を合成すれば，機械翻訳システムが実現される。以上の諸技術を研究開発する。

⑤ 言語データ・ベースの構築

自然言語理解に必要な日常言語に対する辞書データ・ベース（数万語）と科学技術用語データバンク（50万語以上）を作成する。シソーラスの研究も行う。

⑥ 自然言語処理装置

以上の形態素解析や構文解析，意味解析等に用いられる基本技術は，非数値計等を主体

とする記号処理技術である。記号処理技術に適し，しかも効率的な構文解析アルゴリズムと，辞書や文法規則等を含む知識ベースを効率的に検索するための連想アルゴリズムをハードウェア化した自然言語処理装置を研究開発する。

⑦　多言語基本文法の作成

日本語，英語を含む数ケ国語の基本文法（重文，複文を含む）を作成する。

⑧　音素識別方式の研究

音素は音声の中で最も小さい単位で，母音や半母音，｜p｜，｜r｜，｜m｜等の子音からなる。連続音声中では，前後の音韻環境によって音素の特徴が様々に変化するため，このような調音結合の処理法の確立が必要である。また，連続音声から音素記号列を抽出するためのセグメンテーションも重要な課題であり，言語情報の利用が考えられる。子音の中でも鼻音や流音，半母音等は識別が難しく今後の研究が期待されている。

子音と母音を組み合せた音節単位の識別方式は，その組み合せの数だけ識別数が増加するが，単語単位の認識よりも格段に語彙の拡張性が良く，音素識別方式と関連させて研究する必要がある。

各方面での音声研究を効率化するため，標準音声データベースを作成する。

⑨　文章理解方式の研究

音声分析レベルで抽出した音素記別列から音節，単語，文節，文章へ認識対象を拡げていくときに，構文規則，意味情報，語用論等を用いる必要がある。このような言語情報を効率良く利用するには，数値計算よりも記号処理に適したコンピュータが必要である。また，言語情報を音声理解用に，音形規則，単語辞書，構文規則等の形式で格納した知識ベースが必要であり，このようなデータを辞書形式で作成することも重要な研究課題である。

イントネーション等の韻律情報は，疑問調等の文章の意味を把えるために重要であるばかりでなく，単語や文節のセグメンテーションにも大きな役割を果たす。

また，音声波形から抽出した音素記号列はある程度の誤りを含むため，その修正には，大局的な言語情報を利用する方式を開発する必要がある。

⑩　音声合成方式の研究

音声合成には，原音声をそのまま用いる録音編集方式，1ピッチ波形や子音素片の形式で記憶しておいて目的に応じて合成する素片編集方式，音声分析で得られた特徴パラメータを記憶しておいて合成する分析合成方式，与えられた文字記号列から言語的・音響的規則を問いて合成する規則による合成方式がある。分析合成方式は，最近ＴＩが“Speak & Spell”で実用化し注目されている。音質は録音編集方式に比べて遜色なく，語彙数も1万以上のものでできている。しかし規則による合成方式は音節や音素等の小さい単位を用いるので合成対象の文章や語彙数の柔軟性が格段に良い。アクセントやイントネーションを

始め多くの研究課題が残っているが，人間の音声生成過程に最も近いという意味で，研究開発の意義が大きい。規則のアルゴリズムやソフトウェア開発が重要である。

⑪　話者個人差の学習方式等

音声生成器官の形や大きさ，発声のしかたは話者によって異なり，音声認識上で障害要因となっている。個人差の的確な抽出法，その正規化法，音声を認識していく過程で個人差を学習していく方式等を研究開発する必要がある。個人差の抽出は一方では，話者認識に密接に結びついており，電話等の手段によって遠隔地からでも話者の識別が可能になることが考えられ，応用分野は広い。

⑫　音声理解システムの開発

音素識別方式，文章理解方式，音声合成方式，個人差の正規化等を総合して，音声専用ハードウェアや適切な規模のコンピュータを用いて，音声理解システムを開発する。

前期（第Ⅰ期）には，音素識別方式を中心に，他の諸方式の基礎研究を進める。後期（第Ⅱ期）には，音声理解方式を中心に，言語情報の利用を図り，最後に音声理解システムにまとめあげる。

（b）図形・画像系

図形・画像を人間とコンピュータとの知的コミュニケーションの媒体として利用するために解決しなければならない技術的隘路は，図形・画像をコンピュータ内で抽象レベルで取扱うための技術にある。従って研究開発の方針としては，この隘路を解消する技術を最重点にし，合わせて，知的インタフェース・システムを構成するために不可欠ないくつかのシステム要素，ならびにシステム化技術の研究を行い，目標達成を図る。

①　図形・画像の構造化技術の研究

入力した図形・画像から，構造化された抽象レベルの記述を作るために必要な基礎技術を確立する。本研究開発項目は，ⓐ特徴抽出の研究，ⓑ記述法の研究，およびⓒ知識利用方式の研究より成る。

ⓐ　特徴抽出の研究では，入力された図形・画像を表現し，記述するための特徴パラメータの特徴記述因子（領域，線，孔，距離，その他）を高度化することを目標に，新しい特徴記述因子の開発をしつつ，特徴記述因子と図形・画像が本来持つ情報との関係を明確にすることによって，特徴記述因子間相互の構造性を究明する。

ⓑ　記述法の研究では，図形・画像に含まれる対象を，コンピュータによる推論や問題解決のプロセスの中で扱い得るようにするための手法の研究を行う。基本となるのは，図形・画像が表わすものの構造を明らかにする研究である。これにはいくつかのレベルが存在する。最も入力データに近いレベルでは，図形・画像入力から直接得られた特徴記述因子を構造化して，あるまとまった図形あるいは画像の部分要素を組み立てること

であり，順次高いレベル構造が抽象化されていく。

ⓒ　知識利用方式の研究では，処理の対象ならびに処理方式の両方に関して，処理を進める上で不可欠な知識が駆使されるが，その知識を扱う手法の研究を行ない，知的図形・画像理解の基礎を築く。１つには，図形・画像を処理する際に，そこに表わされている対象に関する知識が不可欠である。たとえば，医用画像などでは，医学的知識がそれに当たる。他方，図形・画像インタフェースを構成する場合には，図形・画像の処理技術（第ⓐ項で開発される）に関する知識が必要である。たとえば，線画が表わし得る矛盾性に関する知識などである。このような知識は，高度なシステムを作ろうとすれば，たちまち大型になってしまうようなものであるから，特に，画像ファイルとの関係においても，十分な技術開発を行わなければならない。

②　図形・画像生成アルゴリズムの研究

コンピュータの中で処理される各種データを人間が目で見て理解しやすいように，図形・画像で表示する必要がある。現在すでに，オフィス・オートメーション用の端末機器等には，数表を判り易いグラフ表示に容易に変換できるシステムが組み込まれているものも出回っている。

ここでは，単に数表のグラフ表示に留まらず，図形・画像として，コンピュータの推論・問題解決の過程に現われる情報をディスプレイ等を介して表示するための，図形・画像生成アルゴリズムを研究開発する。

③　システム化技術の開発

知的インタフェース・システムを構成するための技術を開発する。開発の核は，前項での研究成果を取入れて作るⓐアルゴリズム・ベースの開発と，そのアルゴリズムを高度にプログラムするためのⓑ図形・画像処理用プログラム言語の開発である。

ⓐ　アルゴリズム・ベースの開発

図形・画像を媒体とした知的インタフェースを構成する場合には，それまで開発した種々のアルゴリズムを問題対象に適した形に変形して使う形式を取ることになる。そのために，そのベースとなる種々のアルゴリズムを集積したアルゴリズム・ベースを開発しなければならない。このアルゴリズム・ベースには，各アルゴリズムの特徴および使い方が，知識ベースとして推論レベルで使える形で含まれているようなものとなる。

ⓑ　図形・画像処理システム構築用プログラム言語の開発

図形・画像の生成も含めた処理のアルゴリズムは，アルゴリズム・ベースとして，ファイルへ蓄積されるので，本プログラム言語は，それらを用いて，システムを記述するための言語として機能することを目標とする。それには，アルゴリズム・レベルでの推論機能を，言語自身が持っていなければならないし，また，それに付随する知識を利用

する機構も有する。

④　図形・画像処理マシンの開発

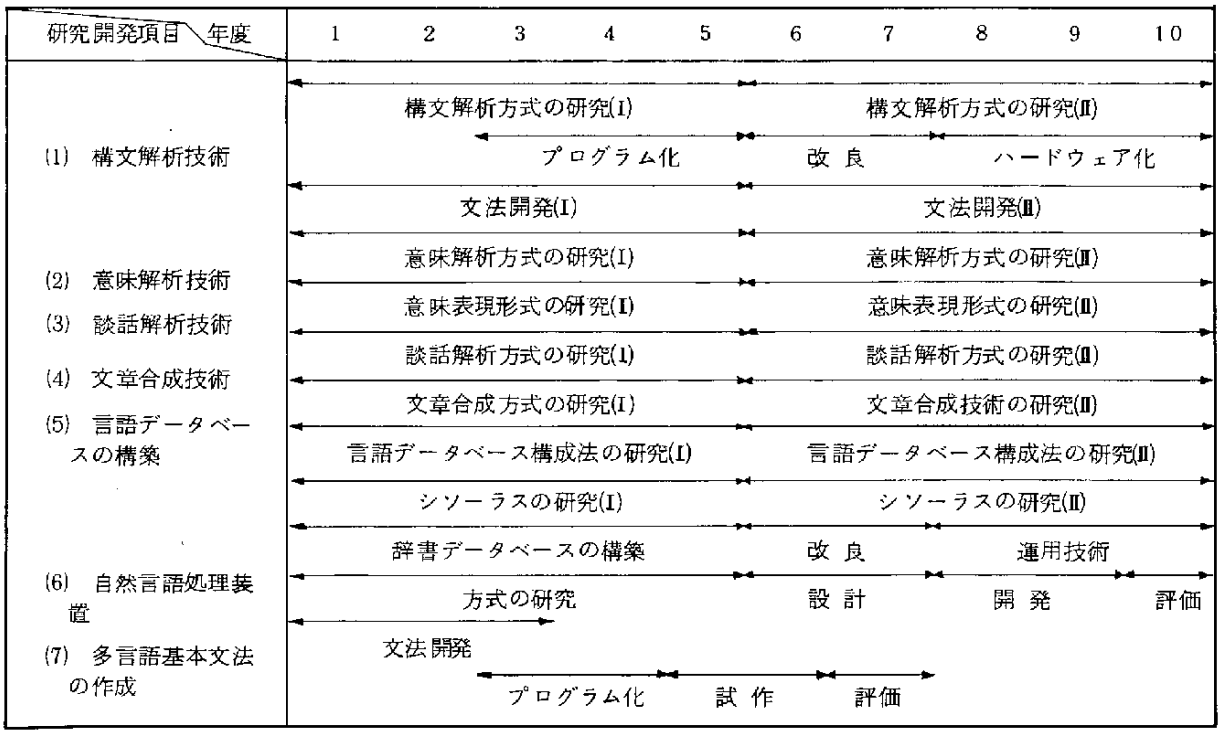
“原理的に実現可能”であることと，“実際に実現した”こととでは，全くその価値が違うことがある。特に，情報処理技術は，言わば全て“原理的には実現可能”な技術を扱っているのであるから，知的インタフェース・システムでは，人々がその真価を認識するような“実際”のシステムのデモンストレーションが不可欠である。知的インタフェースの処理速度と質を保障するために，ⓐ図形・画像処理用高速演算装置，ⓑ高機能高精度図形・画像入出力装置及び，ⓒ大容量図形・画像ファイル・マシンの開発を行う。ただし，本ハードウェアは，第５世代コンピュータの開発プロジェクトの外部に於いて，適当なハードウェアの開発がなされ，本プロジェクトへ時期的，技術的に，その機能を提供しうるのであれば，それを採用する方が望ましい。

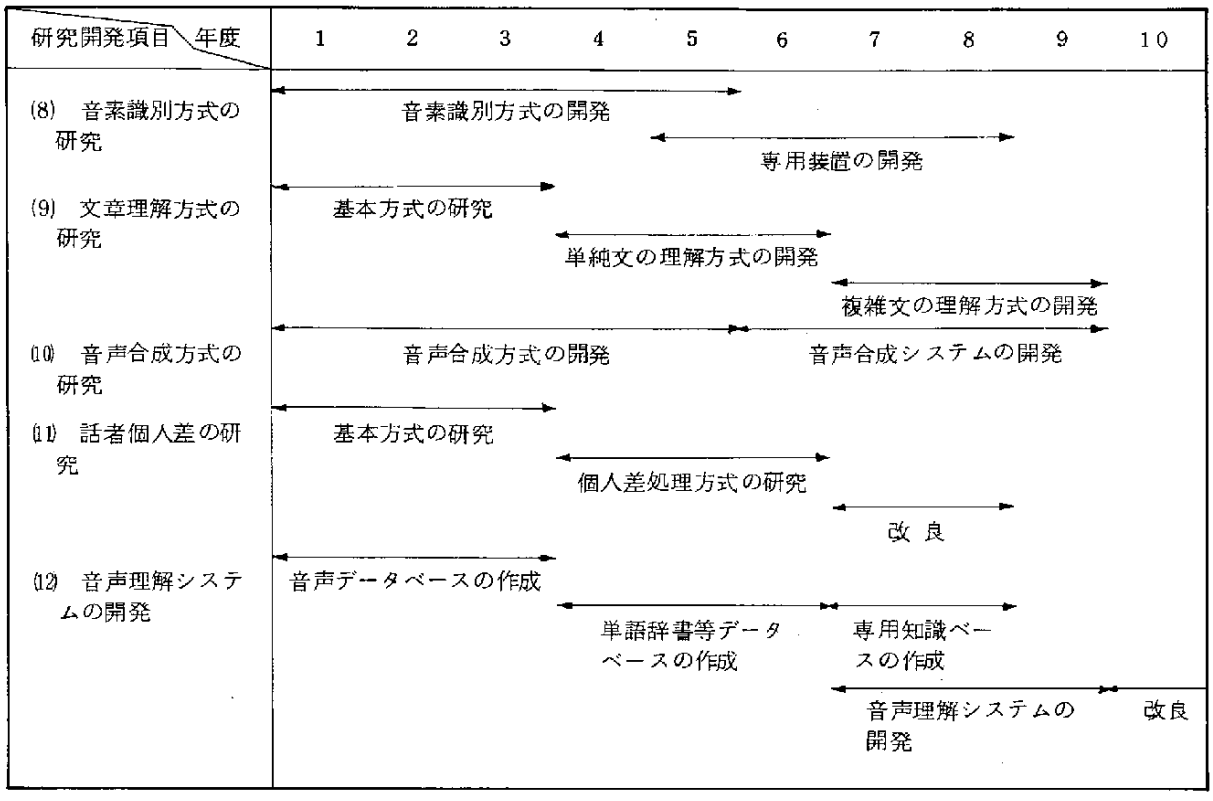
⑤　自然言語と図形・画像との関連づけの研究

知的インタフェースは，自然言語，音声等の技術と，図形・画像等の技術とを融合することにより完成する。本項目では，その融合技術を研究開発することを目的とし，図形・画像の内容を，言語で記述したり，あるいは言語で記述された内容を，人間の理解しやすい図形・画像の形態に変換して表示するための語彙の開発，語彙と図形・画像との変換ルールの開発等を行う。

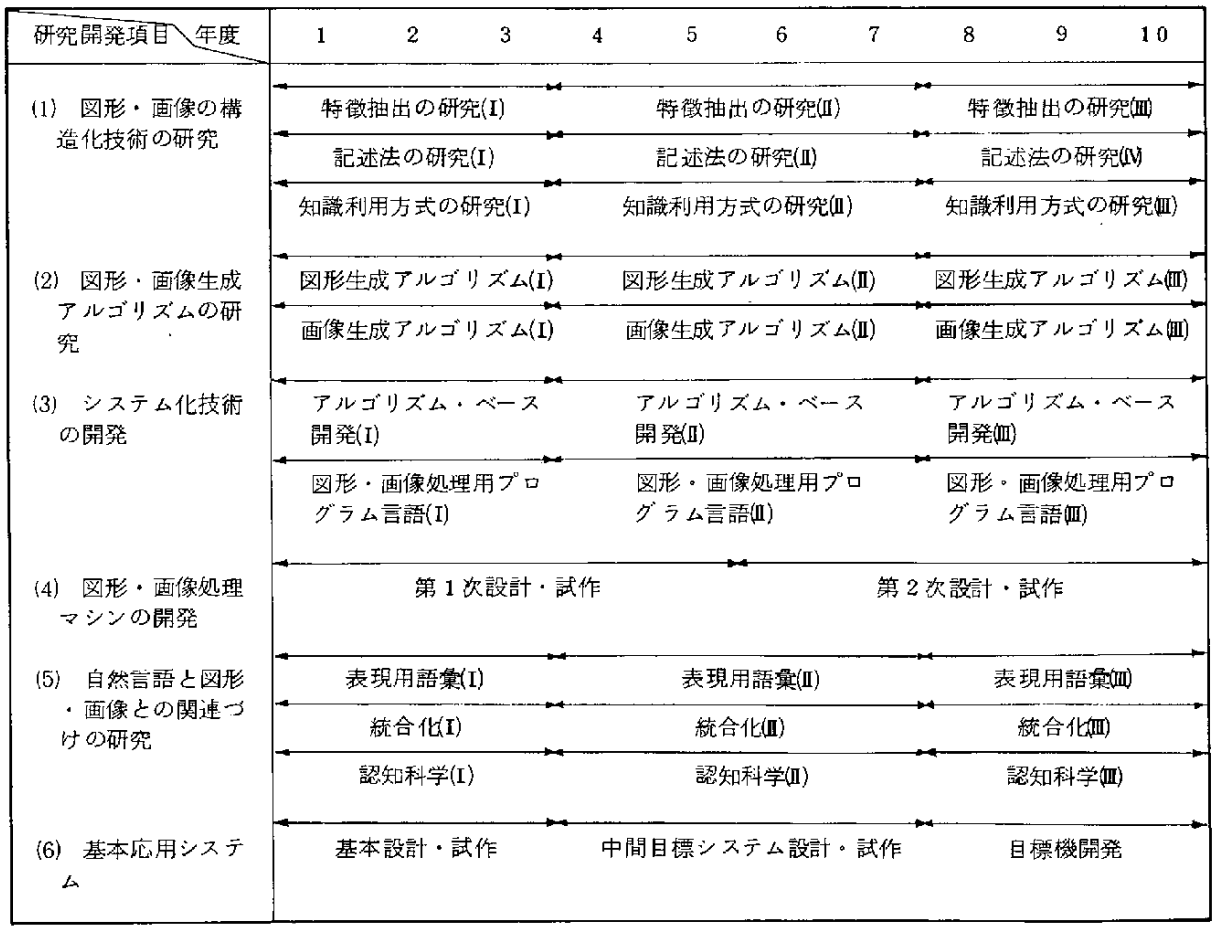
(2)　研究開発スケジュール

（a）自然言語・音声系




研究開発項目＼年度
1
2
3
4
5
6
7
8
9
10
(8) 音素識別方式の研究
音素識別方式の開発
専用装置の開発
(9) 文章理解方式の研究
基本方式の研究
単純文の理解方式の開発
複雑文の理解方式の開発
(10) 音声合成方式の研究
音声合成方式の開発
音声合成システムの開発
(11) 話者個人差の研究
基本方式の研究
個人差処理方式の研究
改良
(12) 音声理解システムの開発
音声データベースの作成
単語辞書等データベースの作成
専用知識ベースの作成
音声理解システムの開発
改良

（b）図形・画像系


研究開発項目＼年度
1
2
3
4
5
6
7
8
9
10
(1) 図形・画像の構造化技術の研究
特徴抽出の研究(I)
特徴抽出の研究(II)
特徴抽出の研究(III)
記述法の研究(I)
記述法の研究(II)
記述法の研究(IV)
知識利用方式の研究(I)
知識利用方式の研究(II)
知識利用方式の研究(III)
(2) 図形・画像生成アルゴリズムの研究
図形生成アルゴリズム(I)
図形生成アルゴリズム(II)
図形生成アルゴリズム(III)
画像生成アルゴリズム(I)
画像生成アルゴリズム(II)
画像生成アルゴリズム(III)
(3) システム化技術の開発
アルゴリズム・ベース開発(I)
アルゴリズム・ベース開発(II)
アルゴリズム・ベース開発(III)
図形・画像処理用プログラム言語(I)
図形・画像処理用プログラム言語(II)
図形・画像処理用プログラム言語(III)
(4) 図形・画像処理マシンの開発
第1次設計・試作
第2次設計・試作
(5) 自然言語と図形・画像との関連づけの研究
表現用語彙(I)
表現用語彙(II)
表現用語彙(III)
統合化(I)
統合化(II)
統合化(III)
認知科学(I)
認知科学(II)
認知科学(III)
(6) 基本応用システム
基本設計・試作
中間目標システム設計・試作
目標機開発

(3) 研究開発の目標・仕様

自然言語・音声という言語情報，もしくは図形および画像という視覚情報を媒体として，人間がコンピュータと知的に対話するための技術を確立し，知的インタフェース・システムを提供することを目標とする。

音声の研究開発の目標・仕様は次のとおり。

① 対象語彙は，コンピュータ関連用語や科学技術文献等の中の1分野につき，専門用語および常用単語を含める。

② 話者適応機能を持ち，不特定話者を対象とする。

③ 日本語および英語の音声出力を可能とする。

④ 識別処理は実時間に近い能力を目標とする。

図形・画像系の研究開発の目標・仕様は次のとうりとする。

① 図形・画像を介して，人間がコンピュータと円滑に対話するシステムの構築に必要なソフトウェアならびにハードウェアの開発を行う。

② 対象とする図形は，中規模以下の機械の設計図面程度の複雑さ，画像は医用写真程度の複雑が取扱えるものとする。

③ 処理速度は，人間との対話手段として円滑さを損なわない程度に十分速いものとする。

④ 中間目標として，扱う図形・画像の複雑さを最終目標の約7割を狙う。処理速度については，中間目標は特に定めずに，処理方式の確立に主力を注ぐこととする。

3.3 知的システム化支援システム

本節では，基礎ソフトウェア・システムの上で働く様々な知識情報システムを実現するために欠くことのできない支援システムについて述べる。それらの支援システムは，ソフトウェアの開発のために用いられ，一般に，プログラミング・システムとなる。そして，これらの支援システム自身が，知識利用システムとして実現されると考えられる。

このようなプログラミング・システムとして，知的プログラミング・システムと知識ベース設計システムの２つを研究開発課題として取り上げ，以下にその詳細を述べる。

3.3.1 知的プログラミング・システム

〔１〕 概　要

プログラミング技術の過去における発展を見ると，機械語から始まって，アセンブラ言語，コンパイラ言語が発明され，プログラミングはその度に飛躍的に技術の向上を見た。

しかしながら，ソフトウェアはハードウェアに比べて技術の進展が遅いことも事実であり，ソフトウェア危機は潜在的に広まっているという見方が強い。その一番の原因は，プログラミングが非常に高度な作業であり，しかもはっきりとした方法論がないことである。すなわち，プログラミングは，現在は未だ工芸的（ art 的 ）段階に止まっているわけである。今後ハードウェアがますます進歩し，それをソフトウェアで十分に利用し切れなくなるという事態を回避するためには，プログラミングを工学的段階にまで高める必要がある。

工芸的な段階から工学的な段階に進むためには，プログラムの機能についての考察が必要となる。とくに，プログラムの自動作製システムを実現するためには，プログラムを構成する部品すなわちプログラミング言語の各言葉が表わす機能をコンピュータで処理できなければならない。これは，従来研究されてきたプログラミング言語のシンタックス的，あるいはプログマティックス的側面の問題とは異なるセマンティックス的側面の問題である。

工学的手法の基礎は，モジュラー・プログラミング，すなわちモジュールの組合せによるソフトウェアの製造技術である。モジュラー・プログラミングを支える技術は，２つある。第１は機能モジュールとそれに基づく組合せ技術で，第２はデータ抽象化によるモジュール化と，それに基づく組合せ技術である。前者の組合せ方は，部品を横方向に結合して新しいソフトウェアを組立てていくので，水平型組合せ技術と呼ぶことにする。後者の組合せ方は，抽象データ型を操作する抽象機械の上で動くプログラムと，抽象機械を実現するプログラム群とを垂直方向に組合せてソフトウェアを組立てて行くので，垂直型組合せ技術と呼ぶことにする。

この直交する２つの組合せ技術は，プログラムの正当性の証明およびプログラムの自動合成のための基礎技術となっている。水平型組合せ技術によって，合成されたモジュールがどのような機能となるかを容易に知ることができる。また垂直型の組合せ技術は，モジュール分割技

術とも考えられ，プログラムの正当性の証明は，抽象機械上のプログラムと抽象機械を実現するプログラム群とに分けてできるので，その規模が小さくなり，証明は著しく容易になる。

プログラムの自動合成は証明の逆過程とも考えられ，証明が容易になるにつれ合成も容易になる。

水平型組合せ技術は，組合せて得られたモジュールの機能が構成部品の機能と接続方法によってのみ定まることを基礎にしている。このような性質は，functionalityの原理と呼ぶ。大域変数を介した状態の変更という副作用を利用しない言語は，この性質を満足している。それはLISP，PROLOG などである。関数型言語でのモジュール化は，PROLOGに代表される述語論理型言語では，より一層進む。LISP での条件文は，PROLOGでは，各条件毎の文の集まりとして表される。さらに，そのような文の集まりは，データの形でシステムに蓄えられる。文同士の組合せは不要となる。それは，パターン照合に基づく手続き呼出し機構によって実行時に行われる。

これらの言語は，Algol，FORTRAN，PL／I などのメモリの更新を直接扱う言語に比べて，経過依存的な計算を扱えない点で言語としての能力は劣るが，数学的な扱いが容易であり，上に述べたような良い性質を備えている。

垂直型組合せ技術は，仕様と言語の深いギャップを埋め合わせる大切な技術である。その間に何層かの適当な中間のインタフェースを設けて，仕様から基本言語への橋渡しを行う。この技術によって，要求分析ー仕様記述ープログラミングに渡るプログラミングの活動を，統一した考え方，記述体系によってサポートすることが可能となる。

現在のところ，水平型および垂直型の組合せ方式を兼備したプログラミング言語は未だ出現していないが，それは，これからの理論的研究のメイン・テーマの１つである。

プログラムの自動合成を実現するためには，人工知能研究の成果を採り入れて，プログラミングの知識を利用した推論システムによって実現するのが妥当である。言語あるいはモジュールの機能の数学的記述は，推論システムが利用できる知識として，知識ベースに蓄えられる。そして，目的とするプログラムの持つべき機能を手掛りに知識ベースを探索し，推論を積み重ねることによって，プログラムの自動合成が可能となる。

知的プログラミング・システムは，知識情報システム開発のための高度な支援システムであり，またそれ自身が，知識情報システムとして実現されるものである。この意味では，KIPSの応用例でもあり，プロトタイプとして考えることもできる。

〔２〕　必要性

知的プログラミング・システムの必要性の第１は，ソフトウェア危機の解消であることは言うまでもない。ハードウェアのコストの減少に比べ，ソフトウェアの生産性の向上は遅々として進んでいない。その原因は，ソフトウェアの作成が工学的な技術として正しい発展形態が見

られなかったからである。工学的技術の原点は，機能のモジュール化，部品化であり，その組合せによる製品化技術である。そのような意味で，知的プログラミング・システムは，本来の技術的発展過程の上にソフトウェア技術を作り直す道具立てであると言える。このことによって，ソフトウェアのかかえている問題は，その多くが解決されると期待される。

このような適正技術の進歩は，実は処理速度や記憶容量などのハードウェア的要件を考慮せずに，議論することはできない。モジュール化技術は，当面は現在の計算機の使い方に比べて十分効率的とは言えない面がある。しかしながら，ハードウェアの進歩，あるいは，ソフトウェア指向のハードウェアの開発によって，この問題は解決されるであろう。さらに，ソフトウェアの開発費までを含めたTotal Costを考えると，新しい方法の技術的妥当性がより明らかとなるであろう。

知的プログラミング・システムは，その技術の延長上に自動プログラミング・システムの実現を見込んでいるが，これは，次世代の計算機の利用が，職業的プログラマから，一般のユーザへと拡大されることを考えると，その技術の重要性を認識することができるであろう。

知的プログラミング・システムにおける検証技術は，今後VLSIの出現によってソフトウェアのハードウェア化が進んで行く状況を考えると，非常に大切になってくる。プログラムの単純な誤りは，ハードウェア化の前に完全に取り除いておくことが必要だからである。

〔3〕　内外技術の現状

(1)　国内技術

データ抽象化をサポートするプログラミング言語には，京都大学のイオタ言語，阪大，電総研での代数的アプローチなどがある。これらは，すべて未だ試作実験段階であり，システムの実用性についての検証はこれからの段階である。

PROLOGは，電総研，武蔵野通研，東京大学などで，実験的使用，改良などがなされている。並列PROLOGや図形言語の提案もなされている。

プログラム・ベースの研究は，始まったばかりである。電総研で，データ抽象化技法とからめた研究がなされている。東北大で，プログラムの自動合成のためではないが，各種のプログラムの相互利用を目指したプログラム・ベースの研究がなされている。

関数型言語の研究としては，抽象データ型をもった関数型言語の研究が東工大でされている。さらに，属性文法による仕様記述から関数型プログラムを導き出す研究が同じく東工大でされている。

データベースのための関数型言語である関係代数を基にした知的データベースが電総研で研究開発されている。そのシステムは，関数型言語上でのプログラムの合成，改良，およびデータ抽象化技術によるシステムの実働化を行っている。

(2)　海外技術

データ抽象化をサポートする言語は，Algol系の言語であるSIMULA 67が最初のものである。SIMULA 67は，並列に動作する対象群を記述し，シミュレートさせるために開発された言語である。そのとき，個々の対象（Object）を中心とした記述を行うために，クラスの概念が発明され，データ抽象化技術に育っていった。また，Object間の情報のやりとりに着目した計算モデルとしてACTORがMITで開発された。ACTORでは，データ，手続きなどをすべてメッセージとしてObject間でやりとりし，計算を進めていく。その機構はデータフロー機構を包含するもので，理論的枠組としての重要性が認識されている。その後，対話型の言語としてSMALLTALKが作られた。SMALLTALKは，Objectの考えでコンピュータ・グラフィックスの機能を整理し，ユーザにとって使い易い言語となっている。抽象データ型の形式的定義と，それに基づく証明系を持った言語としては，Alphard（CMU）とAFFIRM（USC－ISI）がある。代数的アプローチに基づく言語としてOBJ（UCLA）データ抽象化，手続き抽象化，制御抽象化などを取り込んだ実用的言語を目指したものとしてCLU（MIT）の研究開発がある。

論理型プログラミング言語PROLOGは，主としてヨーロッパを中心に研究開発が進められている。多くの応用プログラムの記述がなされると共に，コルーチンなどの新しい制御機構，あるいは抽象データ型などの概念をPROLOGの中に取り込む研究が続けられている。他方，PROLOGの並列実行方式とマシン化の研究も始められている（米国カリフォルニア大学）。

〔4〕　研究開発内容

(1)　研究開発項目

(a)　モジュラー・プログラミングと検証理論の研究

関数型言語に基づく水平組合せ技術と，データ抽象化に基づく垂直組合せ技術によるモジュラー・プログラミングの手法を確立し，その手法で作られたプログラムの検証理論を研究開発する。

検証を行うためには，プログラミング言語を構成する各命令の機能の正確な記述と，作るべきモジュールの仕様の正確な記述が必要となる。垂直組合せ技術では，作られたモジュールは，より上位のモジュールを作る際の部品あるいは基本命令となるので，これら2つの言語は，一つの枠組の中で統一的に使用できるものでなくてはならない。

(b)　仕様記述とプログラムの合成理論に関する研究

モジュラー・プログラミングを基礎として，与えられた仕様を満足するプログラムを合成する方式を研究開発する。

作るべきプログラムの仕様は，単純な入力・出力の関係による機能記述にとどまるものであってはならない。すなわち，計算速度・使用記憶量の両面からの性能記述などが含まれていなくてはならない。

プログラムの合成を行うには，多くの部品（モジュール）の中からそのプログラムを組立てるのに役立つ部品を選び出し，それらを調整して組合せることが必要となる。とくに，各部品はデータ表現の差異を吸収できるようになっていなければならない。この調整技術は，用意すべき部品の数を適当な規模に抑えるために不可欠な技術である。

プログラムの自動合成にとって，もう一つの重要な技術は，どのようにして部品を選び出して，どのようにして組立てて行くかである。これは，人間がプログラムを作るときと同様の知的な能力を必要とする。本研究では，プログラマの専門的知識を知識ベース上に再現して，組立て作業の自動化を図ることを目指す。そのために，アルゴリズムのパターンによる分類等の研究を行う。

(c) プログラムの検証・合成システムおよびプログラム・ベースの開発

プログラムの検証・合成は，その挙動を記述された多数のモジュールが蓄積されてはじめて有効なものとなり，実用となる。この蓄積が，プログラム・ベースとなる。その上に，(a)(b)で開発された方式に基づくプログラムの検証および合成を行うシステムを開発する。

モジュールの蓄積は，各種応用分野毎に広範囲にわたって推進することが必要である。この作業は，モジュールの標準化の作業とも考えられ，困難ではあるが大切な作業である。また，標準化を行うためには，概念の整理をつきつめて行うことが要求される。いわば，プログラミング技術の集大成となるべきものである。また将来のＶＬＳＩ化の種を提供するものでもある。

(d) プログラムの保守・改良・管理システムの開発

プログラム開発の大きな分野に，プログラムの保守・管理あるいは改良の問題がある。プログラムが大きくなるにしたがって，このような活動は困難性を増してくる。知的プログラミング・システムの目的の１つは，このような作業の機械化である。とくに，仕様の変更や，パフォーマンスの改善などのために，プログラムを容易に修正できる機能は必須である。そのために，以下の点について，今後の研究が必要である。

- 柔軟なプログラムの追求
- 等価変換技術
- プログラムのドキュメント管理，セマンティクスとプログラムの中身との対応づけの技術

(e) プログラム設計コンサルタント・システムの開発

プログラムの作成にとって，問題の仕様をきちんと与えることは，大変に困難な問題である。プログラミング言語が高度になり，思考に適合した言語になるに従って，その作業は除々に楽になってきてはいるものの，とくにプログラムの非専門家にとっては，最も困難な仕事として残されている。本研究では，プログラム設計コンサルタント・システムの開発を目

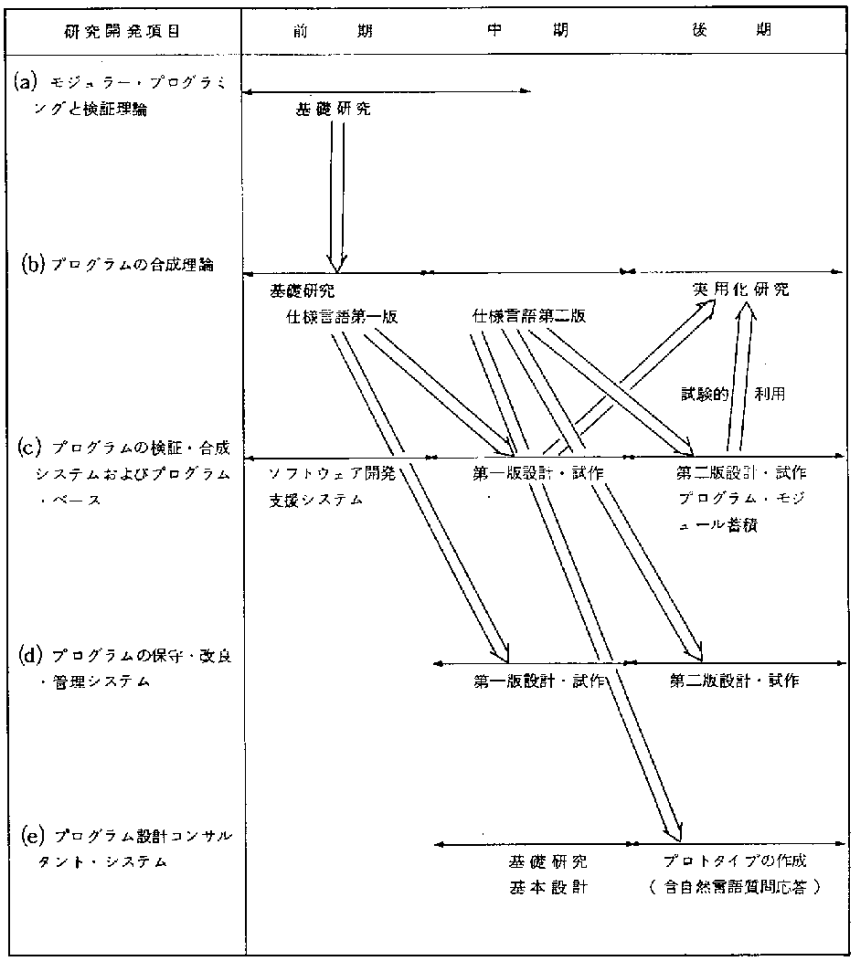
指す。具体的には，自然言語による質問応答システムとするため，プログラムの設計，仕様記述に必要な自然言語のサブセットを理解する言語理解システム，設計を支援するための know how を蓄積したコンサルタント・システムの開発を行う。その結果得られた仕様記述は，プログラム自動合成システムの入力となるものとする。

(2) 研究開発スケジュール

本研究課題では，前期においては，本プロジェクトの研究開発にとって必要なソフトウェア開発支援システムの開発を行う。その詳細については，システム化技術研究分科会報告書の第Ⅱ部 3.1 システム化技術を参照されたい。また，ここであげた 4 つの研究テーマについても，基礎研究を中心に同時に着手する。

中期においては，基礎研究の成果をもとに，プログラムの検証・合成システムのプロトタイプを試作する。プログラムの保守・改良・管理システムについても同様である。プログラム・ベースの基本設計，並びにプロトタイプの作成を行う。プログラム設計コンサルタント・システムについては，そのための基本設計を行う。

後期においては，上記システムの最終版を設計・開発する。さらに，プログラム・ベースにプログラム・モジュールの蓄積を行う。プログラム設計コンサルタント・システムは，自然言語による質問応答を含めたシステムのプロトタイプを作成する。



(3) 研究開発の目標・仕様

以下に，前記研究開発項目(c)～(e)について，その目標・仕様を，中期と後期に分けてまとめる。

研究開発項目	中期目標・仕様	後期目標・仕様
(c) プログラムの検証・合成システムおよびプログラム・ベース	データベースへの検索手続きなど比較的小さく，一定分野のプログラムの合成，変換による改良。 小規模なプログラム・ベースの開発。 関数型，論理型およびデータ抽象化プログラムの検証システムの作成。	データベース管理システム，言語プロセッサなどの大規模プログラムの合成。 大規模なプログラム・ベースの開発。
(d) プログラムの保守・改良・管理システム	関数型，論理型プログラムについて，プログラム理解システムを作成。 等価変換実験。	プログラムの性能評価システムと等価変換による改良システム。 プログラム修正システム。
(e) プログラム設計コンサルタント・システム	（基本設計）	自然言語による質問応答。 データベース管理システム，データベース応用システム，などの設計コンサルテーションが可能なシステム。

3.3.2 知識ベース設計システム

〔1〕 概 要

知識ベースシステムを設計し開発を行ない，運用段階へ至らせるまでの過程で，知識ベースシステムを作成する者を支援するための技術を確立する。

知識ベース設計システムは，知識情報処理の基本応用システムの一つと見做すことができる。その基本的な機能は，知識情報処理システムを設計・開発・運用するのに必要な種々の技術と知識を知識ベースの中に組織的に所有し，その知識ベースに基づいて，知識ベースシステムを構築しようとする者を支援することである。

従って，知識ベース設計システムの構成も，基本的には，一般の知識ベースシステムと同様

に，知識ベースと推論機構から成っている。

研究開発の項目としては，一般の知識ベースシステムに関連した部分は基礎ソフトウェアシステムの知識ベース管理システムの項で研究開発を行うので，本項では取り上げないで，知識ベースを設計・開発・運用するという立場から特に重要である課題に重点を置くことにする。

知識ベースの設計・開発・運用で最も基本的な共通技術は，知識に関する知識（これをメタ知識と呼ぶ）をコンピュータ内でどのように表現し，それに基いて抽象レベルの推論を行なうかという課題である。この課題を，知識ベース設計システムの基礎研究課題として位置づける。

次に知識ベース設計システムの核である知識ベースの開発という課題がある。この知識ベースには，知識ベース設計・開発・運用のための知識が収納される知識ベース，知識獲得のための方式や表現，あるいは人間やデータからどのようにすれば知識を抽出して得ることが出来るかというような知識が収納される知識ベースなどが主な開発対象である。

更に，知識ベースが広く一般のユーザに使われだすと，途端にシステムの規模が大型化するという傾向があるので，そのために大型の知識ベースを構築する技術の研究開発も，重要な課題である。この大型化の傾向は，研究室から現場へ出て，世界的に実用的レベルで使われているDENDRALという知識ベースシステムで発生しているもので，実用しているユーザは，コンピュータの制約条件などはとかく無視して，性能本位でシステムを使うというのが本来の姿であるということを反映するものである。ハードウェアの大型化と，アドレス空間の拡大がこの傾向を助長するので，大型の知識ベースの構築技術の開発が不可避である。その中心的技術は，知識ベースの共同開発と拡張のための技術である。

知識ベース設計システムの目標性能は，知識の量がルールに換算して20,000個程度で，知識の獲得を半自動的に円滑に行う機能と，推論の過程を日本語で説明する機能とを有する知識ベースシステムを，２人／月以下で構築可能とするものとする。

〔２〕　必要性

社会の広汎な分野で知識ベースシステムに対する期待感が高まりつつあるが，知識ベース構築の技術が確立すれば，現在は潜在している知識ベースシステムへの需要が，一気に表面化すると考えられる。それは，知識ベースシステムが，問題に適した柔軟な応用システムとして最適であるからである。

知識ベースシステムは，問題の記述がやり易く，拡張性や保守性に勝れているために，大型のシステムや，カスタムメイドの木目の細かいシステムを，比較的安価に作ることを可能にする。またその推論メカニズムは，非決定論的な推論を必要とするような問題の解決システムを可能にするので，従来の手続き型プログラム技術で実現困難であった種類の問題についても，応用システムとしてまとめる手法を提供する。更にこの知識ベースシステムの基本的構造が問題解決の手順の部分と知識の部分とに分離してできていることから，知識ベースシステムの核

となる知識ベースそのものが，一度作られると一連の作業において共通に使うことができる。たとえば，応用システムとして，何か高度な装置の故障診断のための知識の他に，その装置の構造や機能に関する知識が必要であるが，これはこの装置を設計し製作する際に開発される知識ベースを再び使える。すなわち，知識ベースシステムは，それを構築する際のオーバーヘッドは，システムがいろいろ作られて拡張されるに従って，知識ベースを共通に使うことにより，急速に軽減するという利点がある。これは情報の蓄積を有効に活用できることを意味する。

このように知識ベースシステムには多くの利点があるが，これを利用するために要求されるのが，そのような知識ベースシステムを効果的に効率よく構築する技術である。従来のプログラミングシステムとは異なり，非論機構と知識ベースとが明確に分離した構造を持ち，問題に応じて最適な推論メカニズムを構成し，その問題の関連知識を知識ベースとして組織する機能を持つ知識ベース設計システムは，それ自身，知識ベースシステムの応用システムである。

従って，たとえ高級プログラミング言語があっても，知識ベースシステムの枠組である推論機構から知識ベースまでをその言語レベルから作製すると，それ程大規模ではないシステムでも少なく見積っても３～４人／年のマンパワーが必要となる。

本来知識ベースシステムは，応用分野の個性や利用者の好みに合わせて，いわば＂カスタムメード的＂に使われるという魅力がキャッチフレーズでもある。この特色を経済性を保ちながら実現する技術の研究開発の必要性は明らかである。

〔３〕　内外技術の現状

(1)　国内技術

国内においても，知識ベース設計システムに対する関心が高まりつつあり，いくつかの提案がなされ始めた段階である。たとえば，東京大学では，医用コンサルテーションシステムの構築の要件の整理作業が進んでいる。

知識ベース設計のための技術の基礎は，知識ベース管理システムのための技術である。それに関しては，東京大学，京都大学，大阪大学，九州大学，豊橋技術科学大学，東京理科大学，電総研，武蔵野通研等で行なわれている。（知識ベース管理システム参照）。

(2)　海外技術

米国では，知識ベースシステムのツール（道具）の研究として，すでに７年以上の歴史がある。

代表的な例としては，Stanford 大学で開発が行なわれているEMYCIN，AGE，Rutgers 大学で開発されているEXPERTなどのシステムがある。

EMYCINはプロダクションルールに基いた知識ベースシステムを構築する場合に，ルールのインプリメントを支援するシステムである。文法上の誤りやミススペルのチェックも行う機能を持っている。EMYCINを用いて，MYCIN，PUFF，SACON 等のシステムが非常に

短期間に開発することができた。

AGE は，知識ベースシステムを作ろうとしている人を支援するシステムで，黒板モデルに基いたシステムである。知識工学上の種々の技術をパッケージ化してあり，ユーザに対するアドバイスの機能も強力である。

EXPERTはやはり知識ベースシステム構築の支援システムであるが，FORTRAN で作られており，移植性が高いという特徴がある。

知識ベースの誤りを見出し，修正するシステムとしては，Stanford大学のTEIRESIAS がある。

知識表現に関するシステムとしては，同じくStanford 大学でUNIT，RLL などというシステムの開発が行われている。

〔4〕　研究開発内容

(1)　研究開発項目

(a)　メタ知識の表現方式とその利用技術の研究

知識ベースシステムは，知識ベースと推論エンジンから構成される。従って設計技術は，この 2つの要素の設計技術と，全体のシステム化を行うための設計技術が中心となる。

知識ベースシステムは，応用分野の問題の物質や，そのシステムが対象とする利用者の好み等によって，見掛けはかなり違う形を取ることになるであろう。従ってここでは，共通性が強い基礎的な技術の研究開発を重点におくことにする。

この観点から，2つのサブ項目を設定する。その第 1は，知識に関する知識の表現方式の研究である。知識に関する知識を，メタ知識と呼ぶことにする。メタ知識には，知識の構造に関するメタ知識と，推論の制御手順に関するメタ知識の 2種類がある。

知識ベースシステムを設計する場合に，そこで使われる知識を構造化して，知識ベースの枠組を作っていかなければならない。現在のところ知識を階層的な構造に組立てるのが良いとされている。この場合，知識のどのような側面，あるいはレベルで階層化を図るかは，問題の記述と推論の効率に重大な影響を及ぼすことは明らかである。

そのための技術は，個々の問題の特質に依存することは避けられないが，知識を「抽象レベル」で分類し，そのレベルで構造化を図り，具体的な知識を構造化する際のガイドラインを生み出すという手順によれば，共通技術として多大な効果を設計の段階に及ぼすことになる。

第 2は，メタ知識を対象とした推論方式と検証理論の研究を挙げる。知識の構造と推論の制御手順についてのメタ知識を用いて，抽象レベルの推論を行なうことは，この知識ベース設計システムで設計した知識ベースシステムのシミュレーション機能を持つことになり，従って出来上った知識ベースシステムの間接的検証技術を提供するとともに，そのシステムを

修正，拡張する際に，有効な助言を与えることを可能にする。

またメタ知識自身が完全であることを検証するための理論研究及び手法の開発も行う。

(b) 知識ベース設計開発支援システム

知識ベースシステムを構成する知識ベースと推論エンジンの内，知識ベースの設計開発法の開発は特に重要である。その理由は，個別の知識ベースシステムでは，その推論エンジンはシステムによって，それ程のバリエーションが存在するとは考えられないので，基本的な推論エンジンが用意されていれば，それを実際の知識ベースシステムの推論エンジンとして使うことは，それ程手数は掛らないと予想される。ところが知識ベースは，出来上った知識ベースシステム毎に，かなり個性的なものが要求されていて，かなりのバリエーションが存在すると考えられるので，それを統一的に取扱える設計道具の開発は，効果が大きい。その機能を果たすのが知識ベース設計開発支援システムである。

これは，２つのサブ項目よりなる。第１は知識ベース設計法の開発である。知識ベースの枠組として，現在のところプロダクションシステム，フレーム理論，セマンティックネットあるいは黒板モデル等いくつか具体的な提案がなされている。また基礎ソフトウェアシステムの知識ベース管理システムというテーマの下に新しい枠組の開発が行なわれる。

このような知識ベースの枠組は，問題によって適不適があり，知識ベースの設計において，どの枠組をどのように使うとよいかも，設計者へ助言できるシステムを開発することを目標に，基礎研究を行う。

第２のサブ項目は，知識ベース設計開発技術の知識ベース化である。前記サブ項目の研究成果をもとに，知識ベース設計開発に関する知識，すなわち「知識ベース設計開発の専門家の知識」を知識ベースとしてコンピュータ・システムに固定する作業を中心に，知識ベース設計開発支援システムを開発する。

(c) 知識ベース増植支援システム

知識ベースシステムが広く一般に使われるようになると，システム自体が大型化する傾向にある。このような大型のシステムは，一度に完全な全システムを開発することが困難になり，基本部分から逐次大きく成長させていく手順を踏むのが普通である。そのために必要な技術が知識ベース増殖支援システムである。

特に知識を如何にシステムの中に取り込むかという技術の成否は，大型の知識ベースシステムを開発する場合のボトルネックとなる。また，知識ベースの保守技術，システムの効率を上げるための知識ベースのコンパイル技術なども重要となる。

ここでは，次の３つのサブ項目を特に取り上げ，知識ベース増殖支援システムの開発を目標に研究を推進する。

第１は知識獲得技術の知識ベース化である。知識獲得技術の基礎的な研究開発は，基礎ソ

フトウェアシステムの中の知識ベース管理システムで行なうが，ここでは，知識獲得のために必要な，いろいろなノウハウを中心とした知識ベースの開発を重点に行う。

この知識はいろいろな局面に関する知識より構成される。たとえば，知識提供者から如何にすれば，上手に知識ベースの枠組に合った形で知識を引出せるかという知識なども含まれる。

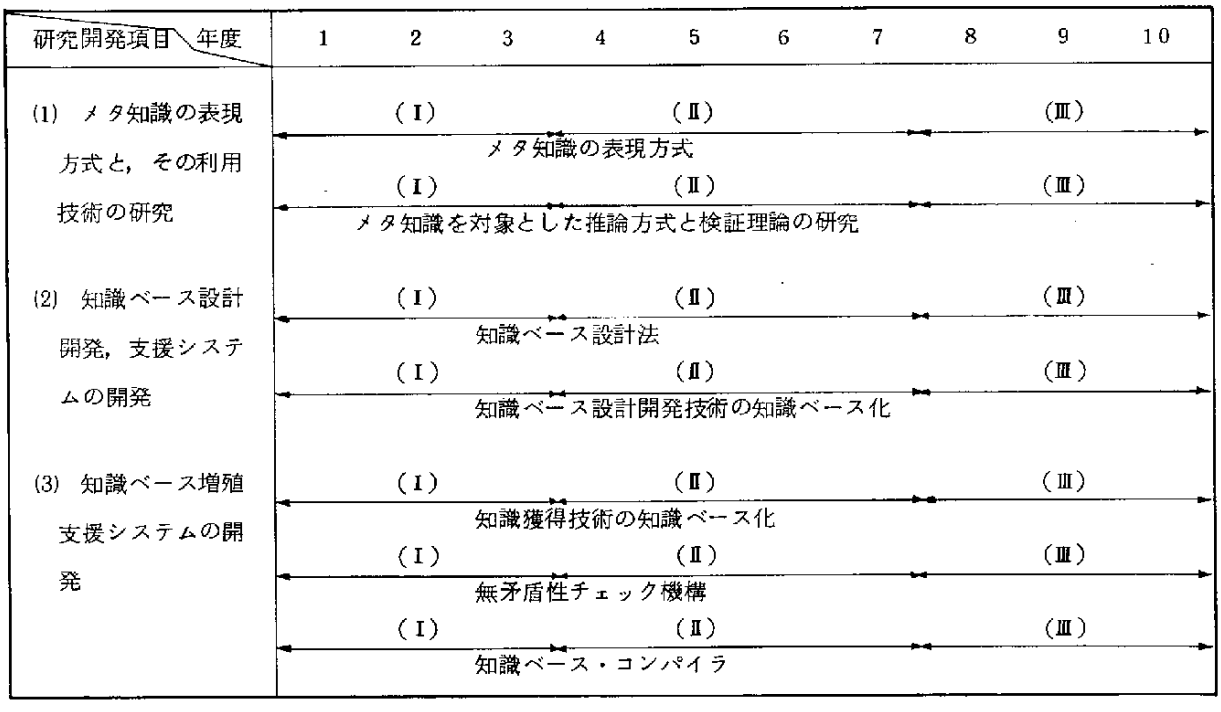
第２は無矛盾性チェック機構の研究である。ここでは，知識ベースの内容に，互に矛盾するものが含まれていないか，あるいは欠如しているものがないかを，自動的に検証するメカニズムの研究開発を行う。その結果は，知識の獲得の自動化へも利用できる。

この機構により，大型化する知識ベースの改良，保守が容易になる。

第３は知識ベース・コンパイラの開発である。知識ベースシステムとして具体的な応用システムを作ろうとする問題は，往々にして問題自身十分に解明されつくしていない場合がある。たとえば，医療診断のコンサルテーションを行なう知識ベースシステムを作る場合でも，知識ベースを作っている段階では必ずしも診断の理論が確立しているとは限らないのである。従って，作られる知識ベースの内容に矛盾がないとは言えない。これについては前述の無矛盾性チェック機構の研究により，調べることが出来る。

更に，その知識ベースの内容が，推論を実行するのに具合い良く構成できているかを自動的に調べ，できるだけ効率よく知識の起動がなされるような機能を持つ知識ベースコンパイラの研究開発を行う。

(2)　研究開発スケジュール



(3) 研究開発の目標・仕様

知識ベースシステムを設計し開発を行ない，運用段階へ至らせるまでの過程で，知識ベースシステムを作成する者を支援するための技術を確立することを目標にする。その開発目標・仕様は次のとおりとする。

① 相当な専門知識を必要とする高度な問題について，専門家にとっても有意義なコンサルテーションを提供する知識ベースシステムの構築を容易に行なえるものとする。

② 設計する知識ベースシステムの規模は，その知識を規則表現に換算して，約２万規則程度のものとする。

③ メタ知識レベルで，システムの検証がある程度行なえ，大型の知識ベースシステムのデバッグが容易に行えるものとする。

④ 中間目標として，設計すべき知識ベースシステムの規模を，目標の３割程度とする。

⑤ この知識ベース設計システムによって設計すべき応用システムの例として，化学プラントあるいは原子力プラントなどの運用のためのコンサルテーション・システム，ＶＬＳＩ設計のためのコンサルテーション・システム，新材料開発のためのコンサルテーション・システム，医療診断コンサルテーション・システム，知的教育システム，知的エイジェント・システム，あるいは法律に関するコンサルテーション・システム等，社会的要請が大きくかつ実現により大きな効果が期待されるものを想定する。

3.4　基本応用システム

3.4.1　機械翻訳システム

〔1〕　概　要

社会の国際化・情報化の急速な進展に伴い，国際間にわたる各種の情報を迅速に収集分析し，我国の立場を速やかに世界に周知させ，資源や貿易摩擦等にみられる国際的問題の解決を図ることは，貿易国日本にとって，今後の大きな課題になっている。そのためには文書翻訳システムの開発がまず第一に必要である。

本研究で開発するシステムは，これまでの機械翻訳システムの研究・用語集作成等にみられるドクメンテーション技術の研究・知識利用に関する人工知能研究，等の成果を総合し，実際のテキストを翻訳する本格的な機械翻訳システムを目指す。また，これに伴って，機械翻訳の結果を翻訳専門家及びその分野の専門家が，適宜修正し，完全な翻訳テキストを作るための Word Processing 技術の開発も同時に行なう。

以上のシステムを開発するために必要となる自然言語処理理論の開発，大きな用語集・シソーラスを管理するためのデータベース技術の開発，対象分野の知識を体系的に整理し，機械翻訳のために活用するための知識工学的手法の開発，テキスト用データベースの開発，各種の構成要素からなる機械翻訳システムを開発するためのソフトウェア・ツールの開発，高解像度・高速なグラフィック・ディスプレイを用いた本格的なテキスト・プロセシング技術の開発等を行う。同時に，上記の各開発プロジェクトに必要となる高機能ハードウェアの作成を進める。具体的には，自然言語処理を行なうためのマシン，用語集・シソーラス及びテキスト・データに向いたデータベースマシン，テキスト処理用の高解像度・高速なグラフィック・ディスプレイを開発する。このような計算機ソフトウェア・ハードウェアの研究開発の早い時期に，ある特定分野（複数）の用語集・シソーラス等の基礎的な言語データの収集とその計算機化を進め，本研究開発の円滑な推進を図る。

実動するシステムの目標としては，辞書項目数（複数単語からなる用語も含む）10万を持ち，90％以上の精度で人間の介在なしに翻訳し，対象分野の移行が計算機サポートによって半自動的に行えるものとする。可能な対象分野は，科学技術分野の技術報告書・マニュアル，経済市況等，一般のテキストに比べて，比較的専門用語の占める割合が大きく，用語の概念規定が明確な分野とする。

機械翻訳システムは，基礎ソフトウェアシステム（3.2章）中の知識ベース管理システム（3.2.1節）と知的インターフェイス・システム（3.2.3節）をベースにして構成される。これを図3－1に示す。

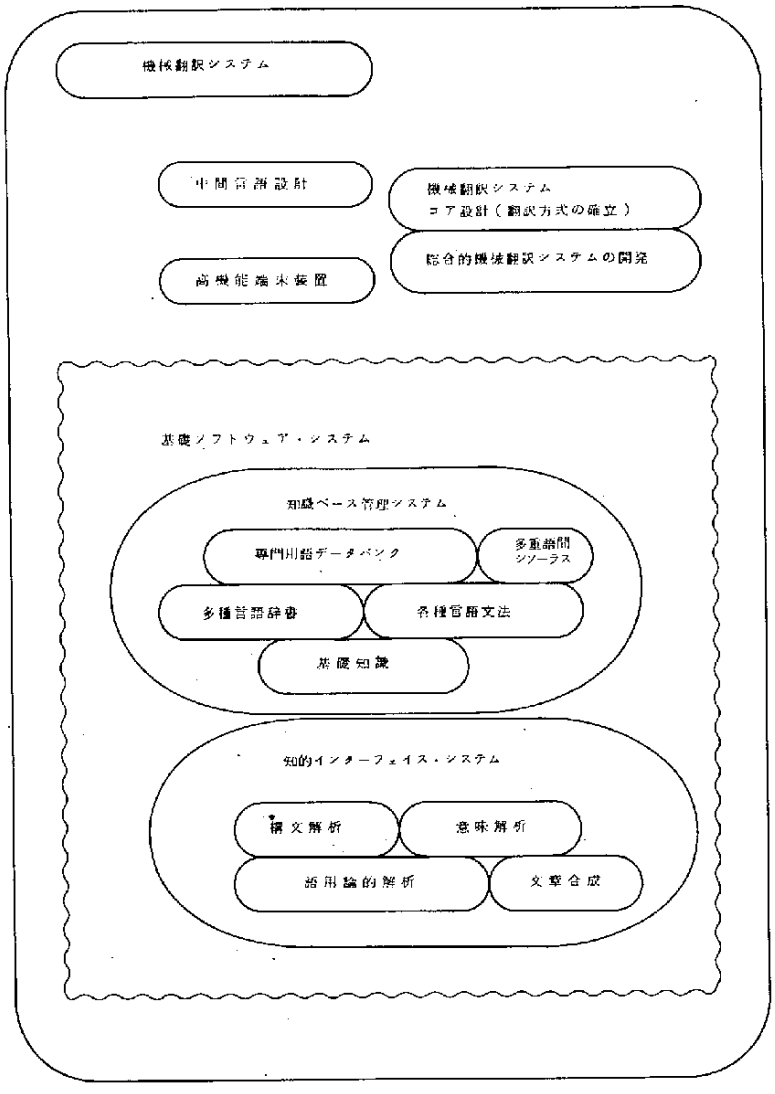


図３－１　機械翻訳システム

〔２〕　必要性

社会の国際化・情報化に伴って，政治・経済・科学技術など，すべての分野において海外からの大量の情報を速やかに処理し，また，海外に対して同様に大量の情報を速やかに拡散してゆく必要性が増大してきている。しかるに一方，日本語はいわゆる孤立言語であり，各種の外国語を自由に扱える日本人が非常に少なく，また，日本語ドクメントを直接読むことのできる外国人も非常に限られている。このことが，とくに日本において，言語障壁として，海外との情報交流を著しく阻害している。とくに，社会の国際化は，あらゆる分野で，正式ドクメント以外の各国語で書かれた資料を，できるだけ速く入手し，それに対応することを要求しているが，日本においては，かろうじて英語が処理できる以外には，例えば，アラビア語・中国語等，これからその重要度を増すであろう諸言語に対して，全くその対応手段を持っていない。逆に，日本語は諸外国において全く未知言語であり，日本国内での諸活動が，諸外国に正当に伝わらず，多くのトラブルのもとになっている。

このような状況において，文書翻訳，とくに，日本語から他言語への翻訳，あるいは，英語以外の諸言語からの日本語への翻訳に対する社会的要請は極めて大きい。

以上のような社会的要請と同時に，機械翻訳システムは，人間の情報活動の中心的存在である言語情報の計算機処理という，今後の情報化社会の基幹となるべき技術の発展に対しても大きな貢献となる。Word Processing，Text Processing の技術，及び，言語を媒介とした情報の集積とその処理に関する技術，知識情報処理技術等に，大きな波及効果を持っている。

〔３〕　内外技術の現状

(1)　国内技術

自然言語を計算機によって処理しようという研究は，日本語固有の漢字の処理という障害のために，諸外国，とくに米国に比べて遅れていたのが，極く最近までの状況であった。しかしながら，ここ２～３年の間に，漢字処理のためのソフトウェア・ハードウェアが急速に進歩し，実用化されるに伴い，より高度な自然言語処理技術の開発への意欲が，各大学・研究所，メーカの中で急速に高まってきている。

また，とくに機械翻訳についても，ＥＣ・カナダ等の多言語社会に比べて，その必要性に対する関心が薄かったが，情報化社会の進展に伴い，世界的規模で多言語社会へと移行しようとしている現在において，その関心は急速に高まってきている。しかしながら，研究それ自体は，現在プロトタイプ的なシステムが各所で開発されはじめた段階で，これを本格的な規模で実験し，実用化するまでには至っていない。

京都大学では，これまでの人工知能的手法を機械翻訳システムに適用し，対象分野を計算機マニュアルに限定した日英機械翻訳システムを開発，実験をすすめている。また，科学技術論文の英文表題を日本語へと翻訳するシステムを開発し，１０万文献の表題に適用して，その結果の評価を行っている。

九州大学では，知識・概念構造を中心にした機械翻訳システムの設計を行っていると同時に，各種日本語表現の収集と，その対応する英語の構造の把握を大規模に行っている。

電総研では，自然言語処理用のプログラミング言語 ＬＩＮＧＯＬ を開発，また，これを使って，自然言語の詳細な意味処理を行うシステムを開発しており，人間の知的情報処理と自然言語処理との相互関連について，すぐれた結果を得ている。

また，富士通研究所においても，機械翻訳システムのプロトタイプを完成している。

この他，機械翻訳のための基礎技術となる日本語情報処理については，東芝，富士通，日立をはじめとする各メーカにおいて，活発に研究されている。

(2)　海外技術

ヨーロッパＥＣでは，EUROTRA 計画が超国家的プロジェクトとして取り上げられ，本格的な実用化を目指して，大規模な研究が始められている。この計画が進み，ＥＣ内での多言語

間翻訳が実用化された場合，ＥＣ内の政治経済の発展や，科学技術の向上に与える利益は，はかり知れないものがある。

カナダでは，モントリオール大学を中心とするTAUM プロジェクトが，政府の援助をうけて，大規模な計画を実施しており，気象通報のための英仏翻訳システムTAUMMETEO が実動している。また，より本格的な航空機マニュアル文の翻訳システムTAUM−AUIATIONがほぼ完成している。

仏のグルノーブルでは，CETAプロジェクトの後をうけたGETA プロジェクトを進めており，この計画自体はEUROTAの中核的役割を果している。

また，米国CDCは，カナダのTAUMプロジェクトのソフトウェアを導入し，本格的な機械翻訳システム開発に取り組む計画をかためている。

〔４〕　研究開発内容

(1)　研究開発項目

①　機械翻訳システム・コア部分の設計

機械翻訳の中心となる入力文解析，出力文合成のための基本的アルゴリズムの設計，および両言語間の構造を結びつけるための中間言語の設計を行なう。この際，対象とする言語対が変化しても，システム概念および基本的アルゴリズムを変える必要のない多言語間翻訳のためのシステム概念を確立する。

②　各種言語の文法開発

①で設計されたシステムに従って，入力文解析の手法を確立すると同時に，日本語を含めた複数の言語に対して実際に大規模な文法を作成し，その妥当性を検証する。作成された文法は，機械翻訳のためだけでなく，一般のテキスト処理にも直接適用できる汎用的なものとする。

③　中間言語の設計

入力文の解析結果を表示するための中間言語を設計し，これと，言語にはよくない知識記述用言語との関連を明らかにし，推論・演繹等の人工知能的な高度機能を機械翻訳に適用する手法を開発する。

④　生成用文法の開発

単に原言語の意味を忠実に伝えるだけでなく，前後の文脈に最も適合し，かつ，人間にとって理解し易い文体で文を生成するための手法を開発し，実際にいくつかの言語に対して，その文法を作成する。ここでの文法も機械翻訳用に限定せず，データベースアクセスを含んだ自然言語による応答システム用の文生成システムとして使える汎用性のある文法を開発する。

⑤　人間の介在を含む総合的機械翻訳システムの開発

実際に翻訳を実行する機械翻訳コアシステムを中心部分に据え，これをとりまく形で人間との interactionを行なうための方式を設計する。このために必要な高機能Word Processing技術の開発，および辞書メンテナンス用ソフトウェアを含むサポート・ツールの開発を行ない，総合的な機械翻訳システムを開発する。

⑥　専門用語データベース（知識ベース）の開発

機械翻訳システムを現実のテキストに適用し，大規模な実用化を行なうために，テキストに含まれる用語についての網羅的な辞書が必要となる。ある特定の専門分野（複数）をとって，そこでの用語を収集し，大規模な専門用語データベースを開発する。このデータベースは，各用語の各言語での訳語を持つ多言語データベースであるとともに，各用語の用例，用語間の相互関連，意味を記述した知識ベースでもある。これら複合的で，かつ，複雑な構造を持ったデータを格納し，有機的に検索・利用するためのシステムも同時に開発する。ここで開発されたデータベースは，機械翻訳システムだけでなく，人間の翻訳作業にも使用され，翻訳の質を向上するのに使われる。また，一般の知識ベースとしても機能する。

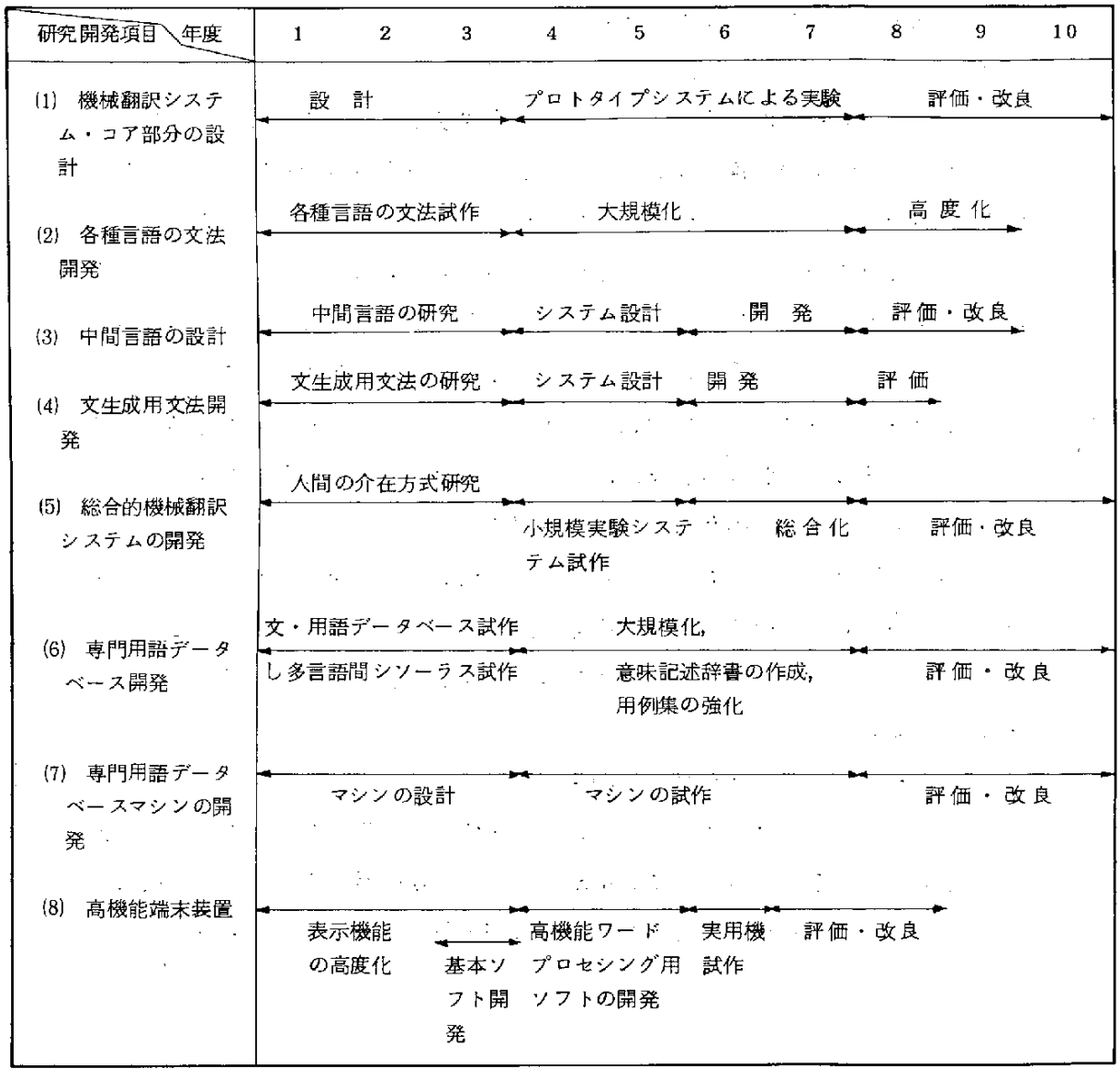
⑦　専門用語データベースのためのマシンの開発

上記⑥の専門用語データベースは，知識ベース的機能と，大規模なテキスト的データとを管理する機能とを，同時に持ち，かつ，高速に処理できなければならない。これを実現するために，従来のデータベースマシンの技術をさらに高度化した，専門用語・知識ベース用のマシンを設計，開発する。

⑧　高機能ワードプロセシング技術の開発

翻訳結果の人間によるエディティング作業を容易にし，図や絵も含めたテキスト全体の編集や印刷用原版の作成を行なうために日本語およびその他の言語テキストを自由に操作し，図・絵等のテキストに含まれる非言語的情報の管理を行なう高機能なワードプロセシング技術を開発する。

(2) 研究開発スケジュール



(3) 研究開発の目標・仕様

① 国際化・情報化社会が急速に進展する状況において，言語テキストの翻訳に対する需要は，今後急速に増大する。この需要の増大に応えるために，人間援助型の本格的機械翻訳システムを開発する。

② 開発されるシステムの仕様は以下の通り。

○中間目標としては，小規模実験システムとして

使用語彙5000語で人間が介在するシステムを開発する。

精度は80％とするが，実験のし易さを考えたシステムを設計する。

○最終目標は：

2－1. 使用語彙数10万語。

2－2. 90％の精度で，翻訳し，人間の介在によって，残り10％を処理する。

2－3. テキストの編集，翻訳結果の印刷までの各行程で計算機が関与する総合システムとする。

2－4. 全コストは，人間が翻訳した場合の30％以上とする。

3.4.2 質問応答システム

〔1〕 概 要

質問応答システムは，システム使用者がシステムに対して質問を出し，これに対してシステム側が適切な応答（これには動作も含まれる）を行うことが基本である。使用者の出した質問内容に不明確なところがあれば，適当な時期に適切な問い返しを行い，最終的に使用者の意図に沿った問題解決を行ったり，それを支援するシステムである。したがってS分科会で提案，

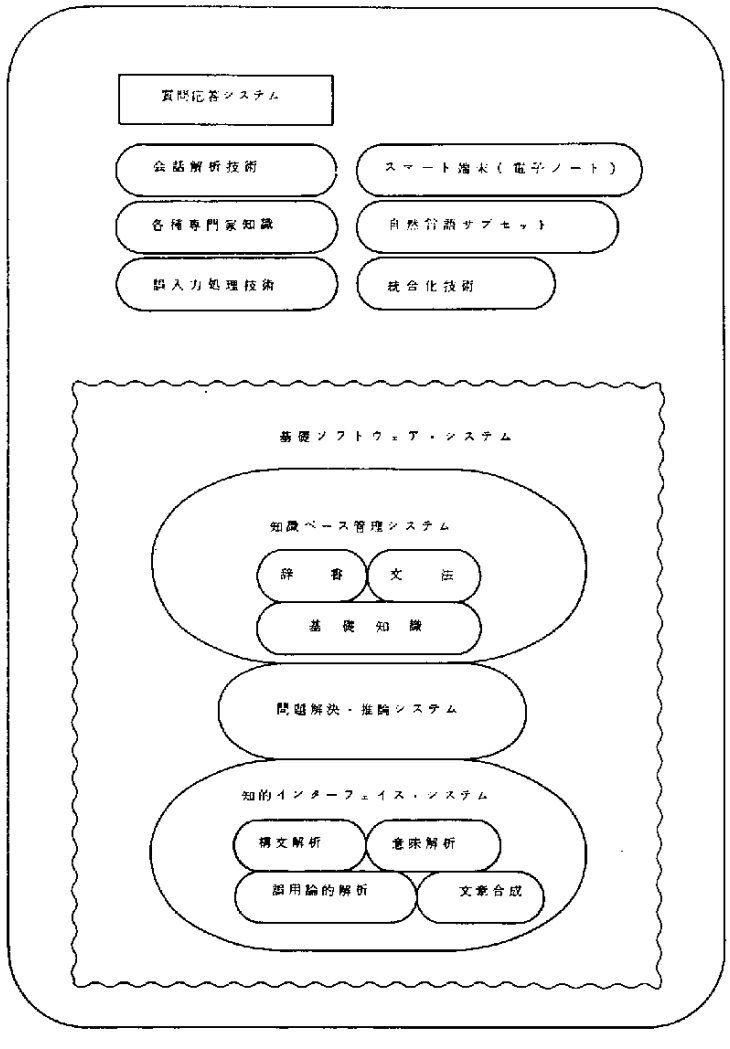


図3－2 質問応答システム

検討されている知的CAE／CADシステム，DSS（意思決定支援システム），知能ロボットや，知的CAC／CAIシステムも，それぞれシステムとしての力点の相違はあるが，システム使用者とシステム相互の間に，質問応答のサイクルが含まれており，広義の意味での質問応答システムであるといって良い。

基本応用システムとしての質問応答システムは，これらの質問応答システムに共通な問題を取り上げ，研究開発を行うものである。したがって，ここで開発されるシステムを中核にして，個々の問題領域に依存した質問応答システムが速やかに設計，製作が可能でなければならない。

第5世代コンピュータのなかで，質問応答システムの果たす役割は，システム使用者に柔軟な対話機能を与え，知識を駆使した高度な質問応答を行う能力をコンピュータに持たせることである。この質問応答システムは，基礎ソフトウェア・システム中の知的インターフェイス・システムをベースに構成されることになろう。さらに，知識ベース管理システムを用いて辞書，文法，その他の基礎および専門知識を作成し，それを問題解決・推論システムを用いて利用しながら，質問の処理が行われるので，基礎ソフトウェア・システムを統合化したシステムとして，質問応答システムが実現されることになる。（図3－2）

〔2〕　必要性

第5世代コンピュータは，「使い易いコンピュータを実現する」という問題意識から発想されている。ここで「使い易いコンピュータ」とは，一般の使用者にとって「使い易い」ということであり，特定のプロにとってのみ「使い易い」ということを意味しない。

現在のコンピュータを使用する場合に，使用者が困難さを感じる最大要因は，使用言語の相違に基づくギャップであろう。使用者が素人になればなるほど，このギャップは大きくなり，現在のコンピュータは，「使い易いコンピュータ」から遠くなる。

第5世代コンピュータの一部として研究開発される質問応答システムは，このような使用言語の相違に関するギャップを解消する必要があり，柔軟な会話機能，すなわち自然言語や音声，図形による会話機能をもたねばならない。

図3－2に示したように，質問応答システムは，基礎ソフトウェア・システムを中核に作成され，柔軟な会話機能と高度な推論能力を持つシステムであり，第5世代コンピュータの一部として「使い易いコンピュータ」実現のための大きな役割を果たすことになる。

〔3〕　内外技術の現状

質問応答システムの内外の情況は，「第5世代の電子計算機に関する調査報告書－自然言語処理－，昭和55年3月，日本情報処理開発協会」の8.2節で詳しく報告されているのでそれを参照されたい。

(1)　国内技術

我が国では，基本応用システムの一つとして取り上げられているような，柔軟な対話機能と

高度な推論機能を有する質問応答システムを目指す試みは少なく，立ち遅れているといわざるを得ない。小規模実験用のシステムとして，電総研のＶＩＳＵＡＬＩＺＥＲ，電電公社武蔵野通研のＭＳＳＳ　７８や音声理解システム作成の試みがある。前者は，積木の世界を対象にした文理解の結果を作図したり，質問応答することができるものである。後者は，ＭＩＴのＳＨＲＤＬＵと同様，文理解の結果を（コンピュータ内にシミュレートされた）ロボットの動作として表示したり，算術文を理解し簡単な計算を行うものである。九大の田町研究室では，気象文を理解して，気象図を作成する研究を進めている。いずれも実用のレベルには到っていない実験システムである。一方，日本ＩＢＭでは，実用を意識した立場から，日本語によるデータ・ベース検索システム「やちまた」を開発した。しかし，これもシステム構成法や応答速度等の問題があり，実用に供されるに至ってはいない。

(2)　海外技術

海外では，1970 年代初頭のＭＩＴの研究，特に南米の地理についての知識を学生に教授するＳＣＨＯＬＡＲ，ロボットに指令を与え積木操作をさせるＳＨＲＤＬＵシステムが著名である。前者は，知識表現の問題と，デフォールト推論の問題に重点がおかれ，入力英文処理は，さほど重要視されていない。後者は，入力英文処理と問題解決推論技術の開発に力点が置かれ，対象領域が制限されれば，かなりの程度の質問応答システムが作成可能なことを実証し，その後の研究を大いに刺激した。

米国のＢＢＮ研究所では，回路図を与え，それに故障要因を付加し，学生に故障を発見させるプログラムＳＯＰＨＩＥや，月の岩石分析用の質問応答システムＬＵＮＡＲを開発している。イリノイ大学のＰＬＡＮＥＳは，海軍の空軍機データベースに対する質問応答を行う。Xerox のＰＡＲＣ研究所では，旅行計画作成システムＧＵＳが，ダートマス大，ＳＲＩ ではデータベースに対する質問応答システムＲＯＢＯＴ，ＬＩＦＥＲが，ＩＢＭのサンノゼ研究所では，同様なシステムＲＥＮＤＥＶＯＵＳ が開発され実験が進められている。最近では，実験システムの作動結果を分析し，本格的な質問応答システムを作成しようとする動きもある。

西独では，政府の援助の下に，ジーメンス社の質問応答システムＣＯＮＤＯＲ等が作成されている。

〔４〕　研究開発内容

(1)　研究開発項目

①　会話解析技術

柔軟な質問応答を行うためには，システムと，システム使用者との間で交わされる会話が，質問の意図に沿ったものであって，会話に首尾一貫性があるものでなければならない。そのためには，質問応答システム側が，

(i)　会話の主題／焦点を抽出し，質問意図の現解を行う

(ii) 常識的な判断を行い，会話の背後に隠れている事実を顕在化する

(iii) 会話の主導権を，システムもしくはシステム使用者のいずれも握ることが可能

(iv) 会話理解のための適切なメモリ・モデルの設定

等を可能にする諸技術を開発する。

② 図形入出力装置

質問応答システムの端末としては，入出力の形態として，漢字を含む自然言語，図形，音声等が考えられるが，色図形の出力はもちろん，入力図に濃淡配色する機能をももつXerox PARC研究所で構想されているDynabookを高度化したスマート端末を開発する。これは，A4～B4程度の画面の大きさを持ち，小型，軽量で，通常のノートと同程度の簡便さで漢字や図形の入出力と，表示画面のハードコピー化が可能なものでなければならない。

③ 専門家知識作成技術

これは，基礎ソフトウェアシステム中の知識ベース管理システムで開発される知識表現，知識獲得方式に沿って，専門家知識を作成するための用具を開発するものである。これにより，専門家知識の更新，削除を容易に行うことが可能になる。究極的には，これは質問応答システムそれ自身によりおきかえられる。知識ベース設計システムの成果を大いに利用する。

④ 誤入力処理技術

質問応答システムの使用者は，プロのみでなく素人も含まれる。したがって，あらゆる使用状況を設定して，それに対処しなければならない。特に大きな問題となるのは，誤入力の処理である。これには，質問文の打ち誤りのレベルから，システム側からの質問意図を，使用者側が誤解するレベルまで，さまざまなものが考えられる。システムは，これらの誤りに対して十分堅牢でなければならない。そのために，誤りの要因を分析し，それに対処する技術を開発する。

⑤ 自然言語サブセットの設計と文法および辞書の開発

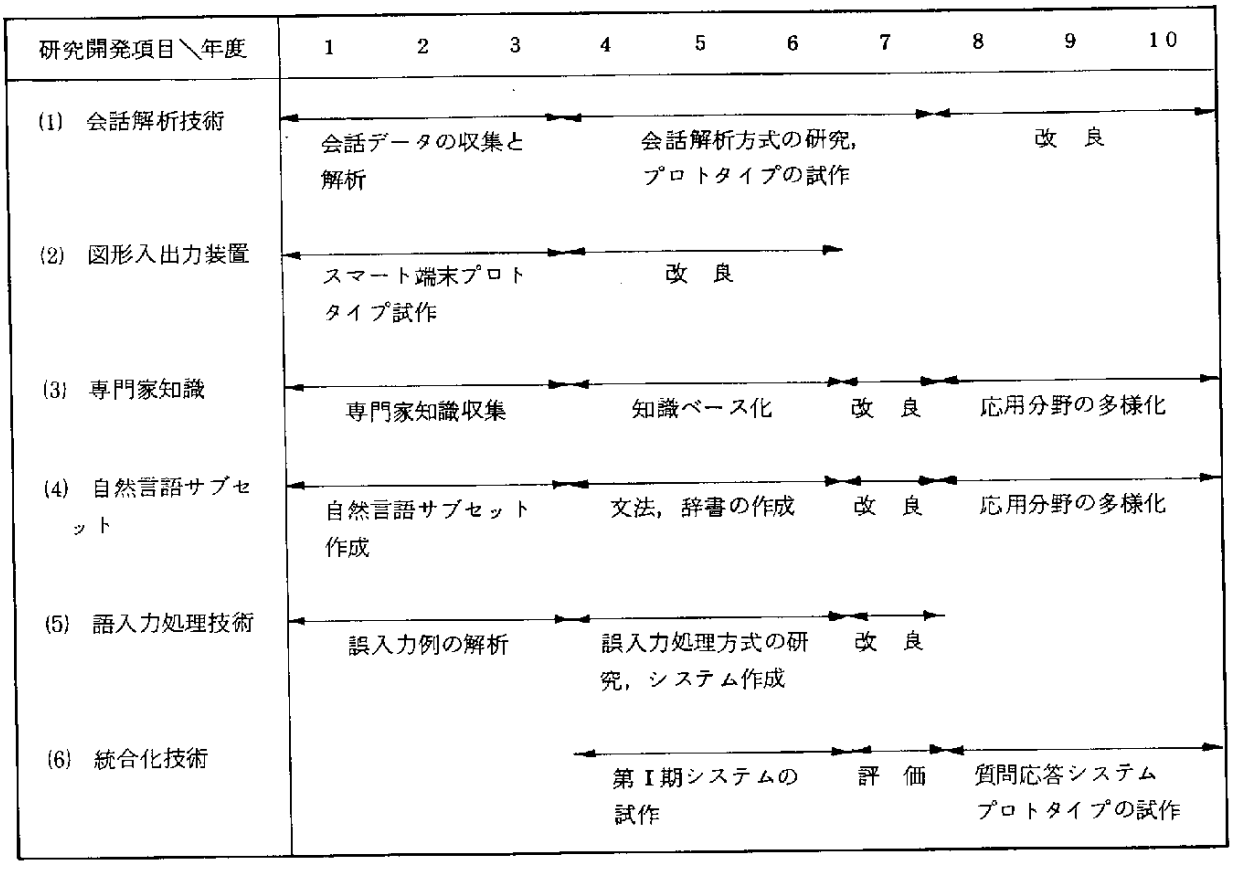
各種問題（専門家）領域に対応した質問応答システムを作成する場合，システムとの会話に用いる自然言語は，一定のサブセットに落ち着くと考えられる。それらを各問題領域毎に設計する。またそれらの文法（辞書）を開発する。

⑥ 統合化技術

上記①～⑤の研究開発成果と，基礎ソフトウェアシステム技術の知識ベース管理システム，問題解決・推論システム，知的インターフェイスシステムを用いて開発される。辞書，文法，基礎知識と，構文解析，意味解析，語用論的解析，文章合成技術を統合化して，質問応答システムのプロトタイプを作成する技術を開発する。

このプロトタイプは，知的ユティリティ・システムの開発に利用される。

(2) 研究開発スケジュール



(3) 研究開発の目標・仕様

① 5年後の中間目標として，一特定専門分野に限定した質問応答実験システムを試作する。

(i) 使用語彙２０００語（日本語）

(ii) 曖昧さ解消のための補助的情報を使用者が与える

(iii) 推論規則数１０００個

② 中間目標で試作した質問応答実験システムを評価し，各種専門分野に対する質問応答システム・プロトタイプを作成する。

(i) 使用語彙５０００語以上

(ii) 推論規則数１００００個以上

3.4.3 音声応用システム

〔1〕 概 要

音声によるコンピュータとの対話が，使い易い形で実用化されると，一般産業機械用やオフィスオートメーションを始め，一般家庭まで幅広く普及することが予想される。基礎ソフトウ

ェアシステムの中の知的インタフェースシステムにおける自然言語・音声系で述べられた音声理解システムが開発されると，その幅広い応用の中で，共通した基本的な応用システムがいくつか考えられる。本節では，機械翻訳の入出力として用いられる汎用音声応答マシン，音声タイプライタ，銀行や電話を用いた問い合せ等で重要な話者認識システム等について説明する。

〔２〕　必要性

コンピュータ等への入出媒体には，音声，キーボード，オンライン手書き文字等がある。工場等の場で，作業者が，コンピュータへ入力するとき，一般にデータ収集者はデータシートに記入し，別の作業者がカード等にキーパンチを行って入力される。一方音声を用いると，データ収集者が直接マイクロホンに発声するだけで，コンピュータへ入力として蓄えられ，オンライン処理が可能になる。また，データの発生速度も，音声入力の方が他の媒体よりも２～４倍速い。音声タイプライタは，音声入力の一つの典型例として取り挙げるもので，自然に発声した連続音声がコンピュータへ入力され（もしくは専用装置によって），日本語の場合は漢字かなまじり文等でプリントアウトするものである。

音声出力は，記録として保存する場合は別にして，出力内容を理解する上で，出力文章を読むよりも容易で使い易いものとなる。また，電話による応答では欠かせないものである。機械翻訳でも翻訳文の音声出力が有用であり，さらに同時通訳作業では必須のものとなる。このような音声出力の基本応用として音声応答システムを開発する。

話者認識は，身分の確認手段として，預金残高の問合せ等の個人情報の検索や各種機密情報の検索，建物への入出時の身分の確認等で用いられることが考えられ，幅広い応用があって社会的ニーズが高いため，基本応用システムの１つとして話者認識システムを開発する必要がある。

〔３〕　内外技術の現状

知的インタフェースシステムの内外技術の現状で述べたように，実用的な音声応用システムとしては，数百単語を識別する音声認識システムや，限られた文章や単語を発声する音声出力システム等が，国内，国外に数例見られる程度である。このような単純なものはこれからも広く普及していくと考えられるが，対象語彙を十分に多くした使い易い応用システムを構成するには，基礎的研究が未だ不十分である。知的インタフェースシステムとしての「音声理解システム」の研究開発の成果を今後利用して，基本応用システムは開発される。

〔４〕　研究開発内容

(1)　研究開発項目

①　音声タイプライタの開発

音声理解システムの研究開発の成果が応用されて音声タイプライタとなる。普通に発声した文章が活字となって打ち出されるシステムであるが，前期は，意味情報等の高度な言語情

報は用いずに，簡単な構文規則と音形規則を利用して単純な文章を認識対系とする。その時，初期誤りを人間が訂正するための能率良い端末が必要とされる。中期・後期は1万単語程度を対象とし，簡単な意味や語用論的情報を用いて，音声認識の誤りを自らある程度修正し，全体としてわかりやすい文章として出力する。

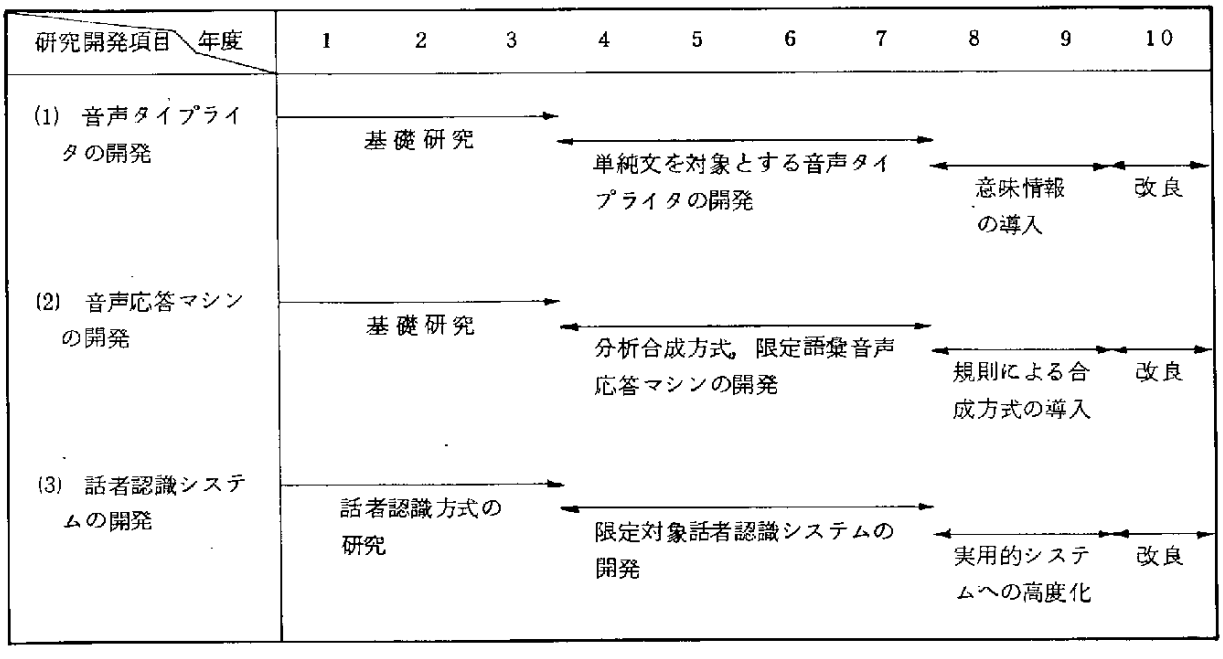
② 音声応答システムの開発

システムに対する質問を理解し，その答を音声で出力するものである。構成は，音声理解部，日本語処理・解答作成部，音声出力部からなり，前2者は他の研究開発の成果を利用してシステムに組み込む。音声出力部は，知的インタフェースの項目で開発した音声出力方式に基づき，分析合成方式の開発や，規則による合成方式の基礎的研究の成果を用いる。前期は対象語彙や分野を小さく限定し，自然な合成音作成技術を開発する。中期・後期は応答内容の意味を把握し自然な会話となるように，イントネーションやアクセント等を適切に取り入れ，対象語彙を1万単語程度に増加する。

③ 話者認識システムの開発

出入管理のように，話者が特定の人物と同一人であるか否かを確認する話者照合と，犯罪捜査のように，話者がある集団の中のどの話者であるかを判定する話者識別がある。話者認識を実用化するには，その認識精度が厳しく要求されるため，特に話者認識用の特徴パラメータの抽出等の基礎研究が重要である。前期は基礎研究および数10名程度の話者を対象としたシステムを作り，中期・後期では数百名程度の対象話者が拡大し，認識時間の縮小を図る。

(2) 研究開発スケジュール



(3) 研究開発の目標・仕様

基本的には，知的インタフェースシステムにおける「音声理解システム」の研究の進捗状況に応じた目標を立てるのが望ましい。

① 音声タイプライタ

前期は，数百～数千単語に簡単な構文情報を利用したシステムとする。中期・後期は，単語数を増やして1万単語程度とし，同時に意味解析を加え，音声認識上の誤りを自ら修正し，全体としてわかり易い文章を出力する。

② 音声応答システム

前期は対象語彙を数千程度とし，分析合成方式を中心とする。中期・後期は，対象単語を1万程度として応答内容の意味を把握し，自然な会話となるように高度化する。

③ 話者認識システム

前期は数10名程度の話者とし，中期・後期に数100名以上の話者に対して実用的な認識時間のシステムを開発する。

3.4.4 図形・画像応用システム

〔1〕 概 要

図形・画像情報を構成的に蓄積し，知識情報処理に利用するために効果的に検索する技術を確立する。

知識情報処理システムの開発の狙いは，使い易いコンピュータシステムを提供することにあると同時に，人間の科学的思考を助長するような知的システムを提供することにもある。

そのために，人間が科学的思考に用いている種々の知識をコンピュータへ蓄え，それに基づいてコンピュータに推論や問題解決を行なわせることによって，人間を知的に支援するシステムの開発を目標にしているのである。

人間の思考過程における図形や画像の役割が非常に大であることは明らかである。人は建物や機械を設計する場合には，図面にそのアイディアを描いて他人とコミュニケートしながら全体をまとめる作業を行う。医者はX線写真を手掛りに患者の病気の診断を行う。われわれの周囲に氾濫する文書にも図や写真が載っていて，読者の理解を助けている。

特に図形や画像を蓄積しておいて利用する場合の効果が大なる場合がある。たとえば，X線による診断に関する医学の教科書を見ると，多数の症例のX線写真に基づいて，その診断手順が記述されている。秀れた教科書を書くためには，良い症例の蓄積が必要であることが判る。

また最近では，気象衛星からの写真がテレビの天気予報で見ることができるようになり，一般の人も雲のパターンと天気との関係に関心を持ち始めたようであるが，過去のいろいろな気象現象を写した衛星写真と，数値予報の結果とを組合せて予測を展開するといった作業におい

ても，画像の蓄積は重要である。

蓄積する図形や画像の量が多くなると，検索の際にいろいろ考慮しなければならない問題が生じる。効率の問題である。これなどは，図形や画像を抽象レベルで記述し，構造化して蓄積することによって解決できよう。また検索の速度やその出力の品質なども，高い性能が要求される。

以上のような応用分野を持った図形・画像応用システムを開発するためには，基礎ソフトウェアシステムの中の知的インタフェース・システムの項で挙げた図形・画像系の研究成果をふまえ，このシステムに特有な技術の開発に重点を置いて課題の設定を行なう。

まず基本的な技術として，図形・画像情報の蓄積・検索方式の研究を行う。図形・画像に固有の構造的記述法を中心に，蓄積・検索のアルゴリズム，品質も高くかつコンパクトな蓄積方式，抽象レベルの取扱いなどの具体的な課題を取り上げる。

検索を行うために都合のよい検索用言語の開発も大切である。これは図形や画像の構造に適合した言語体系を必要とする。その検索を高速に能率よくハードウェアで行なう目的で，図形・画像データベースマシンの開発を行なう。

最終的にまとまった図形・画像応用システムのシステム化を図る技術を研究する。そこでは，本プロジェクトの他の課題で開発された技術を総合的に用いることになるが，特に基礎ソフトウェア・システムの中の知的インタフェース・システムで開発される技術および基本応用システムの中の質問応答システムで開発される技術が主なものである。

〔2〕　必要性

社会の広汎な分野で，図形・画像を情報処理に取り入れたいという要望がある。その最も単純な形態は図形・画像の伝送である。他方，図形・画像処理技術の高度化に伴って，それを高度な情報処理システムの中で利用したいという要望も高まりつつある。

たとえば天気予報のための気象衛星からの画像や診断のための医用画像などは，過去の事例を画像の形で蓄積し，新しい事例から次の現象や新事実を推定しようとするものである。またCAD では図形を介して人間の思考を支援するシステムを目指している。これらはいずれも図形や画像を構成的に蓄積し，うまく検索するシステムを必要とする。

また膨大な図形・画像データを取扱うには，データそのものではなくて，抽象レベルの取扱いが必要になってくる。そのための図形・画像の理解の機能も必要になる。

このような図形・画像情報蓄積・検索システムは，人間の高度な知的活動を強力に支援するものであるとともに，その知的活動の結果の蓄積を可能にするものであり，知識情報処理システムの格好の応用システムである。

〔3〕　内外技術の現状

(1)　国内技術

コンピュータにより画像を処理しようという研究は，国内においても近年広く行なわれているが，現在のところ図形・画像理解のためのいくつかの処理方式が提案されたにすぎず，その方式が本当に複雑な画像の理解に利用できるかどうかわかっていないものが多い。

京都大学では，画像理解のための処理を構造化する研究を行った。

電総研では，対象物体の知識を画像の解釈に利用したシステムを開発した。

シーンの記述法の研究が大阪大学等で行なわれている。

(2)　海外技術

米国では，1975年から始ったARPA(Advanced Research Project Agency)の「Image Understanding」のプロジェクトで，画像理解の研究が行なわれつつある。

スタンフォード大学では；物体の認識に三次元モデルを用いた研究が行われている。

MITでは，対象の記述に詳しさによるモデルの階層構造を採用する方式を提案している。

現在SRI，CMU，USCその他多くの企業の研究グループが画像理解のプロジェクトに取り組んでいる。

〔4〕　研究開発内容

(1)　研究開発項目

①　図形・画像情報蓄積・検索方式の研究

図形・画像情報の蓄積とその検索を効果的に行なうためには，それらの情報が蓄積と検索に適した形態で扱われなければならない。そのための基本的技術を，図形・画像の構造的記述法に求める。

基礎ソフトウェアシステムの中の知的インタフェース・システムの研究開発項目として取り上げている図形・画像の構造化技術の研究を基礎に踏まえ，応用へ発展させる技術の研究を行なうものである。たとえば図形・画像のソート技術なども，この構造的記述法に基づいて行なう方式を開発する必要がある。

次に，図形・画像を物理的なファイルへ蓄積する場合に必要な情報の圧縮，復元，あるいは雑音処理の技法の開発も行う。

この図形・画像情報蓄積・検索システムは知識情報処理システムの一部を成すものであるから，特に図形・画像に付随する各種の知識を，図形・画像のデータと有機的に使える形で知識ベース化する方法を検討しなければならない。これは図形・画像の構造的記述法とも密接に関連するものである。

②　図形・画像情報検索用言語系の開発

図形・画像それ自身，コミュニケーションの媒体であるが，図形・画像の検索の場面を考

えると，言語の助けを借りると具合のよい場合がある。

たとえば，人間は，図形や画像の全体あるいは一部分に着目し，その特徴を表わす何らかの語彙とそれによる表現系を用いて，その図形や画像をコミュニケーションの中とか，思考のプロセスの中へ取り込むことをやっている。

従って，ここでは，図形・画像情報を検索するという立場から，そのための言語系の開発を行なう。認知科学的アプローチも重要な役割を果すものと思われる。

③ 図形・画像データベースマシンの開発

①の図形・画像情報蓄積・検索方式の研究に基づいて，そこで開発された蓄積・検索手法を，高速で実行し，かつ十分実用に耐えられる規模でシステムを実現するのをハードウェア的にサポートする必要がある。

その場合の一つの要になるのは，この図形・画像データベースマシンである。

図形・画像情報は，適宜に構造化されて蓄積・検索されるから，まずその構造を整理して高レベルの汎用性のある処理系が受持つべき部分と，データベースマシンとして専用に設計されたハードウェアが受持つべき部分とを分離しなければならない。

これには，アーキテクチャの技術レベルとのバランスを検討する必要があろう。

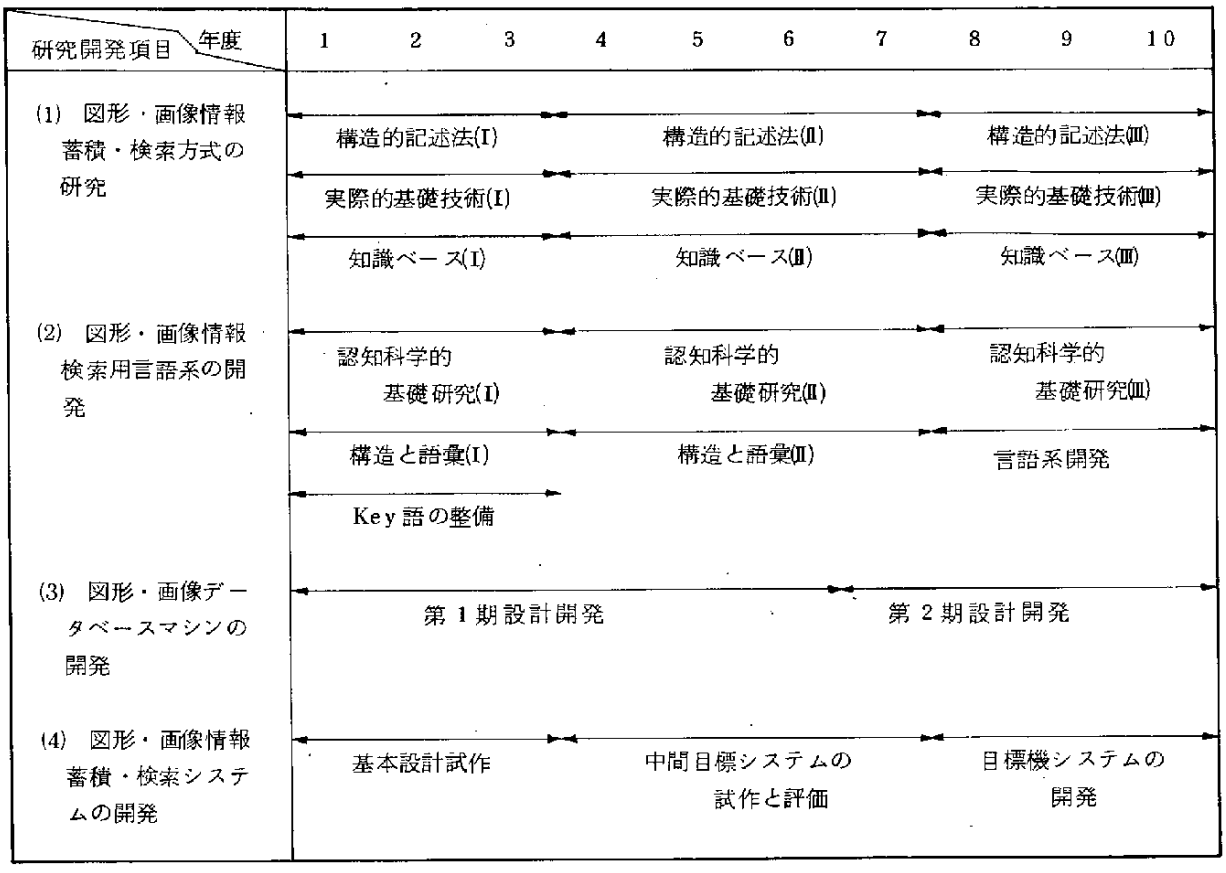
図形・画像ファイルとして，電子化されたファイル，イメージのままでのファイル，あるいはそれらのハイブリッド方式などの可能性について，多角的な検討のもとに設計開発を行うものとする。

④ 図形・画像情報蓄積・検索システムの開発

基礎ソフトウェアシステムの知的インタフェースで開発される技術と，本項目で開発する図形・画像情報蓄積・検索に固有な各種技術とを組合せた，システム化技術の開発を行なう。

技術的には，質問応答システムの技術並びに音声を用いた自然言語の技術を取り入れた，心地よい検索システムの開発を目指す。

(2) 研究開発スケジュール



(3) 研究開発の目標・仕様

図形・画像情報を構成的に蓄積し，知識情報処理に利用するために効果的に検索する技術を確立する。

目標性能は次のとおりとする。

① 検索の対象となる図形・画像データベースの図形・画像枚数は10万枚程度。

② 抽象レベルの記述を含めた図形・画像格納所要時間は数秒以内とする。

③ 図形・画像検索所要時間は，平均100msec以下とする。

④ 中間目標値として，取扱う図形・画像の枚数1万程度，処理速度は目標値の5割程度とする。

3.4.5 応用問題解決システム

〔1〕 概 要

"問題"の記述を入力すると，その"解答"を出力する。しかも，その問題をできるだけ一般的で高度なものを扱えれるようにする。このような問題解決機（システム）の研究は，人工知能研究の中心テーマであった。しかし，扱う問題を一般化すると解決能力は低くなり，解決

能力を高くしようとすれば，対象の特性を分析し，その結果を最大限に利用しなければならないというのが通常の結論である。

ニュウエル等の一般問題解決機（General Problem Salner）に端を発したこの研究は，まもなく，大きく2つの流れとなる。一つの流れは，プログラム言語として，その一般性を保証しようというもので，パターン照合と状態空間の探索を基本機構とする言語が提案，製作された。この流れは，定理証明器，人工知能向言語等を経て，PROLOGという言語に結論付けられつつある。これを充実，発展させるのは基礎ソフトウェアシステムの問題解決・推論システムの中心課題である。

もう一つの流れは，代表的な問題解決を対象に，人間の能力に勝る，あるいは肉薄する位のシステムを作ろうというものである。この流れに，本研究・開発課題は属する。特に対象は，すでに定式化や体系付けがなされているものが選ばれる。最近の知識ベースに基づく専門家システムが，あまり定式化されているものを対象に，多量データと浅い推論を旨とするに比べ，ここでの対象は，非常に深い推論を必要とするものである。対象の代表例が，数式処理とパズル・ゲームプレイングである。他にプログラムを作る，プログラミングという分野があるが，これは，知的システム化支援システムの知的プログラミングの課題である。数式処理システムは，MACSYMA，REDUCEに代表され，人工知能研究の実用化の成功例として良く上げられる。ここでは，少々，趣を変え，公式や法則を知識ベース化し，数式で表わされる知識の一般的な表現システム，すなわち数式理解システムとして研究・開発する。

パズル・ゲームプレイングは，パズルやゲームを何にするかで多種多様であるが，ここでは一番難しいものとされている碁ゲームを取り上げる。碁は，その桁はずれの複雑さのため，単純な探索を高速に行なうということでは，とうてい対処しきれず，局面に対する様々な知識（定石等）を蓄え，利用するというシステムにしなければならない。問題解決システムとして，非常に一般的で高度なものとなる。

〔2〕　必要性

数式理解システムも碁プレイング・システムも，人間がたどりついた分野としては，非常に一般的なものを対象とする。それらを深く研究し，実際に活用できるシステムを開発することは，広く各種応用システムの基本機能が得られるばかりでなく，基礎ソフトウェアシステムが備えるべき要件に対する多くの知見を与えてくれる。

①　数式理解システム

色々な量の間の関係によって表わされる知識を表現するために，人類が永い年月をかけ洗練してきたのが数式である。その性質，数式の意味は，数学の体系として非常に整理，形式化されている。したがって，非常に高い解決能力を持ったシステムを作ることができる。さらに数式で表現される知識は，我々の日常生活で使うものでも，非常に多い。常に目に入っている世

界は，ニュートン力学の法則に従って構成されている情景である。その力学の法則は数式によって表現される等々である。MACSYMA 等の数式処理システムは，現在の数値計算に置き換わる未来の計算システムとして，高度な数式の操作を目的として研究・開発された。したがって，一般的な知識表現システム，あるいは問題解決システムとしては不充分な点が多い。MACSYMA 等では，速度を上げるため，法則や公式はプログラムとして，システム内に書き込まれている。これらを知識ベースとして分離し，法則や公式の追加，変更を容易にし，現在のシステムが不得意とする不等式の処理や方程式の処理をさらに高度にする。すなわち，数式理解システムとして高度化すれば，非常に高い利用価値を持つようになり，各種の質問応答システムのサブ・システムとして非常に有効に利用されるようになる。

② 碁プレイング・システム

パズルやゲームは，問題自体は，簡単に厳密に記述されるが，それとは裏腹に，その解法は非常に複雑になる。問題解決システムとしては，高度な技法を駆使しなければならない。特に高級なゲーム，チェス，将棋，囲碁は永い競技の歴史を経て，正確なランクづけがなされ，能力の評価が厳密に行なえる。いわば，人工知能システムのベンチ・マークと考えられる。

ゲームが高級になればなる程，様々な高度な問題解決の技法を考案しなければならない。しかも，その技法は，一般的なものをより含むようになり，他の各種の応用システムにも適用できるようになる。そこで，人類が考えついた最も高級なゲームといわれる碁をとり上げる。現在，チェスは，米国等での研究により，人間の高段者と競い合えるレベルに達している。その成功は，解法アルゴリズムは単純なものとし，超高速計算機の力によって探索を高速化して得られたものである。ゲームの複雑度を，可能なプレイの生じ方の回数で表わすと，チェスと碁では，控えめにみて，5^{50}，100^{100} 通りとなる。原理的に，チェスの様に単純な探索を高速化するということでは，碁のシステムを作ることはできない。戦略とプランの考えの導入が本質的に必要となり，定石や盤面パターンの辞書化，さらに知識ベース化が必須となる。したがって，システム全体は，非常に一般的な構造を持つことになり，得られる成果は，広く利用できるものとなる。

〔3〕 内外技術の技術

(1) 国内技術

① 数式理解システム

MACSYMA や REDUCE を土台として，数式処理システムとして，高度化し，新しいアルゴリズムを追加したり，実装の方法を改良したりする研究が 4 ～ 5 年前から本格化した。特に理研，東大で活発に行なわれている。他の人工知能システムと結びつける試みは，あまりなされていない。

② 碁プレイング・システム

囲碁の実質的な本拠地でありながら，本格的な囲碁プログラムの研究は，欧米に比べ遅れている。最近，電総研等で研究に着手され，独自な成果が得られるようになった。とかく，我国では，囲碁等は遊びであるとし，正規の情報処理研究よりはずれるとする風潮がある。そのような考え方こそが，自由な発想を封じ，研究全体を魅力無きものとする。

(2) 海外技術

① 数式理解システム

MACSYMAやREDUCEは，10年以上にわたる地道な研究によって，米国において得られた成果である。MACSYMAはMIT，REDUCEはUtah大学で作られ，現在，着実に利用者の輪を拡げつつある。また，システムとしての一般化をねらったものとして，述語論理の定理証明器に数式処理用の推論規則を組込んだシステムも研究・開発され，プログラムの検証システムとしても利用されている。代表的なシステムは，Texas大にある。最近，Edinburgh大でPROLOG上に実現されたシステムは，数式理解システムとしての体裁をととのえつつある。

② 碁プレイング・システム

4，5件のシステムが研究・開発されている。最近では，Reitman等が，碁をパターン認識及びパターンに基づいた推論活動とみてプログラムを作成した。その技量は，27級と評価されている。

〔4〕 研究開発内容

(1) 研究開発項目

① 数式理解基本方式の研究

新しい数式処理アルゴリズムの研究と開発，法則・公式の知識ベース化の方式の研究等を行なう。一般の数式処理システムでは，問題の記述言語は，通常のプログラム言語の形をしている。しかし，その仕様は決して満足のいくものではない。基礎ソフトウェア・システムの問題解決核言語の仕様に合わせ，高度化する研究も行なう。

② 数式理解システムの開発

不等式や方程式に対する処理アルゴリズムや，その他新しいアルゴリズムが組込まれ，知識ベース化された総合的な数式理解システムを開発する。この時，各知識ベースやシステム自体の内部構造を整理して作成し，他のシステムの有効利用を促進するようにする。

③ 碁プレイング基本方式の研究

基本的な戦略やプランの研究，局面パターンの認識方式や評価関数の研究を行なう。同時に，定石データ・ベースや棋譜データ・ベースを作成し，基本データの収集と体系化を行なう。

④ 碁プレイイング・システムの開発

基本方式の成果に基づき，本格的な高能力碁プレイング・システムを開発する。開発に際しては，知識ベース，システム自体の内部構造自体等を整理し，他のシステムで大いに利用できるようにする。例えば，質問応答システムと結び，囲碁指導システム等が作成できるようにする。また，専用の入出力機器等のハードウェアの試作・開発も行なう。

(2) 研究開発スケジュール

研究開発項目＼年度	1	2	3	4	5	6	7	8	9	10
(1) 数式理解基本方式の研究	処理アルゴリズム，公式ベース，等，基本方式の研究				公式ベースの整備と，記述言語の高度化					
(2) 数式理解システムの開発	実験システムの試作				中間目標システム・試作			目標の高機能理解システムの作成		
(3) 碁プレイング基本方式の研究	基本戦略の研究，棋譜データベースの作成				棋譜の知識ベース化					
(4) 碁プレイング・システムの開発	実験システムの試作，基本データの収集				10級システムの試作			初段システムの試作		

(3) 研究開発の目標・仕様

① 数式理解システム

（中間目標）

処理機能としては，現在のMACSYMA程度のものに不等式と簡単な方程式の処理機能を合せたものを，知識ベース化したシステム。

（最終目標）

高度の数式処理アルゴリズムを組み込んだ数式関連の知識表現・問題解決システム。

② 碁プレイング・システム

（中間目標）

アマチュア10級程度のシステム。

（最終目標）

アマチュア初段位のシステム。

4. 研究開発のストーリ

4. 研究開発のストーリ

4.1 研発・開発の指針

その開発規模の大きさ，研究要素の多さからして，これ程の国家プロジェクトは，初めて経験するものであるから，成功裏へ進めるためには，次の二大原則が守られねばならない。

(1) 統合の原則

強力な，研究・開発マネジメント機構が必要であるとともに，各研究者自身の心構えも変革が必要である。細部の独創性を主張するあまり，大局での一致を怠るという傾向は厳につつしまなければならない。本プロジェクトは，多くの新しい概念を作り上げていくという，日本人にとっては，苦手な試みであるから，小異を捨て大同につくという精神は不可欠である。開発ツール（計算機，使用言語，実験用データ）の共通化，標準化が必要で，研究成果の活発な相互利用，相互批判が必須である。これらを保証する物理的な基盤として，研究・開発支援ネットワーク・システムが用意される。

(2) 分散の原則

研究要素が多い，未知の部分が多いということは，一つの課題に対しても，色々なアプローチがあるということである。統合化を急ぐあまり，新しい芽をつみとることのないように気をつけねばならない。非常に多数の相互にからみ合う研究・開発課題を消化していくためには，互いのインタフェースを明らかにし，その仕様を互いに仮定し，それぞれが独立した課題として研究・開発できるようにしなければならない。この仮定したインタフェースを少しづつ精密にし，互いの仮定を一致させていく作業は，研究・開発の進展に従い，注意深く進められねばならない。

知識情報処理システムの研究・開発も，この２つの原則にのっとって進められる。基本応用システム，知的システム化支援システムの研究・開発は，基礎ソフトウエアシステムや他のシステムとのインタフェースを仮定して，それぞれが，専用ハードウエアからソフトウエア・システムまでの開発を含めた自立したシステムを開発するという想定で出発する。ただし，開発ツールは，共通のものが用いられる。プロジェクトの後段部で，最終目標像に向って，統合化が図られるが，独立に発展させる価値のあるものは，専用システムとしても，開発するという判断も下される。

4.2 研究・開発の手順

研究・開発の期間を前，中，後期の３つの期間に分け，それぞれ

前期（３年間）：基礎理論の研究，支援システム上で実験を行ないながら基本方式の策定やデータ収集を行なう。

中期（４年間）：中間目標の試作と最終目標の推敲。

後期（３年間）：最終目標の開発と新理論体系の確立。

を行なう。各グループごとにもう一段詳しくすると次表のようになる。

研究・開発スケジュール

	前　　期	中　　期	後　　期
基礎ソフトウエアシステム	◦第０次仕様（PROLOG＋α） ↓ 支援マシン ◦基礎理論・方式の研究 ◦基本機能の試作 ↑ 支援マシン ◦第１次仕様 ↓ 中間目標マシン	◦中間目標プロトタイプの試作 ↑ 支援マシン（中間目標マシン） ◦最終仕様原案 ↓ 目標マシン ◦最終仕様に合せ支援パソコンの拡充 ◦新理論体系の原図	◦目標マシンシミュレータ ↑ 中間目標マシン（支援マシン） ◦目標システム仕様 ◦目標システムの開発 ◦新理論体系の確立
知的システム化支援システム	◦対話型プログラミング・システム ↓ 支援マシン ◦基礎理論・方式の研究 （仕様記述方式，検証方式） ◦実験用システムの試作 ↑ 支援マシン ◦第１次仕様 ↓ プロトタイプ	◦中間目標プロトタイプ ↑　↓ 目標システム作成 支援マシン ◦仕様記述系，検証系仕様 ↓ 目標システム ◦システム仕様原案 ◦支援マシンの機能拡充	◦目標システム仕様 ◦目標システムの開発 ◦仕様論，検証論の確立
基本応用システム	◦基本方式の策定 ◦基本データの収集と体系化 ◦基本モジュールの試作 ↑ 支援マシン ◦実験システムの試作 ↑ 支援マシン	◦中間目標システムの試作と評価 ↑ 支援システム ◦目標システム仕様の推敲 ◦基礎ソフトウェアシステム，知的システム化支援システムの評価データの収集	◦目標機システムの開発 ↑ 目標マシン （◦単能システムの開発） ◦専用ハードウエアの開発

なお，各グループごとの中間目標の概略は次のようなものである。

基礎ソフトウエアシステム

最終仕様第一次案の決定と小規模プロトタイプの試作。

知的システム化支援システム

研究・開発支援システム上にプロトタイプを作成し，使用実験を可能にする。最終目標システムの仕様原案を作成する。

基本応用システム

研究・開発支援システム上に各基本応用システムの中間目標を個々に実現し，全体とし

ては，機能分散システムとして，ある程度の連携作業を行ないうるものとする。

各課題毎の中間目標は，３章を参照されたい。

各研究・開発課題毎に手順を述べると以下のようになる。

なお，それぞれの課題の各技術項目毎については，３章の各課題毎のスケジュール表を参照されたい。

(1) 前　　期

① 基礎ソフトウエアシステム

≪知識ベース管理システム≫は，知識表現言語を含む表現方式の研究，知識の獲得・利用の手法の研究，分散した知識の管理手法の研究等を行なう。研究の重点は，基礎的な理論についての体系化，基本機能の試作に基づく基礎データの収集におく。知識表現言語などの第一次仕様を決定する。

≪問題解決・推論システム≫は，ＰＲＯＬＯＧを中核とする述語論理に基づく核言語第０次仕様を決定する。これは，研究開発支援を目的とするファームウェア・ベースの述語論理マシンの言語仕様とする。問題解決・推論方式の基本的な研究を行ない，諸記述要素のプログラム言語化を検討し，核言語の第一次仕様を決定する。

知識ベース管理システム，知的インタフェース・システムに用いられる高機能問題解決モジュールの試作・検討を行なう。核言語の仕様の変更・拡張は，ただちに支援パーソナル・コンピュータ上に移され，他システムの開発を援用すると同時に，仕様の当否を決定するデータ収集を行なう。

≪知的インタフェース・システム≫は，自然言語・音声系および図形・画像系に基本的に必要とする機能の方式や基礎理論の　究を行なう。この中には，音素識別方式，音声合成方式構文解析方式，意味解析方式等，さらに図形・画像用の特徴抽出方式，合成表示方式等が含まれる。また，知識表現や問題解決の方式についても基本的な検討を行なう。これらの方式の評価のため，支援システム上に各種システムを試作し，方式の改良を行ない第一次仕様を決定する。

② 知的システム化支援システム

≪知的プログラミング・システム≫は，仕様記述方式，検証方式，合成方式等に関する基礎的な研究を行ない，実験的なシステムの試作を行なう。並行して，支援パーソナル・コンピュータ上に，高機能な対話型プログラミング・システムを開発し，研究・開発の手立てを供すると同時に，今後研究・開発するシステムのデータ収集の体制を作り上げる。基礎ソフトウエア・システムの仕様に整理させた第一次仕様を作る。

≪知識ベース設計システム≫は，メタ知識に関する。表現，推論，検証の基本方式を研究し，実験的なシステムの試作し検討を深める。並行して，支援パーソナル・コンピュータ上に，単

機能な試用システムを作成し，手立てを提供すると同時に，データ収集を行なう。

基礎ソフトウェアシステムの仕様に整合させた第一次仕様を作る。

③ 基本応用システム

≪機械翻訳システム≫は，各種言語の文法の試作，中間言語の研究，文・用語データ・ベースの試作等の基本方式の策定，基本データの収集を行なう。これらは，支援システム上で行なわれ，各種実験システムの試作も行なう。高機能ワード・プロセッシング技術や専用用語データベース・マシンの様な，このシステムに適したハードウエアの検討，設計等も行なう。

≪質問応答システム≫は，会話データの収集と解析や，専門家知識の収集等の基本データの収集や方式の策定を行なう。これらは，支援システム上で行なわれ，実験システムの試作を伴う。スマート端末の様なこのシステムに適した入出力装置等の設計，試作も行なう。

≪音声応用システム≫は，話者認識方式等の基本方式の策定や基本データの収集を行なう。これらは，支援システム上で行なわれ，各種の実験システムの試作も行なう。専用ハードウエア，入出装置等の検討，基本設計等を行なう。

≪図形・画像応用システム≫は，構造的な記述方式，検索方式，検索用言語の基礎研究等の基本方式の策定を行なう。支援システム上に各種の実験システムを試作し検討する。図形画像データベース・マシン等の専用ハードウエアの基本的な検討を行ない，基本設計を行なう。

≪応用問題解決システム≫は，数式理解システムに関しては，処理アルゴリズムの研究，知識ベース化方式の研究を，基プレイング・システムに関しては，基本戦略，局面パターンの認識方式等の基本方式の策定研究を行なう。支援システム上に各種の実験システムを試作し，専用の入出力装置の基本設計も行なう。

(2) 中　　期

① 基礎ソフトウエアシステム

≪知識ベース管理システム≫は，前期に収集したデータに基づき，試作されたデータベース・マシン上に知識ベース管理システムのプロトタイプを試作する。他のシステムの研究・開発用に提供し，評価データを集め，最終仕様の原案を作る。この時，他基礎ソフトウエアシステムとの融合化を目差す。また，新しい理論体系（知識論，知識表現論 etc.）の枠組を描き出す。

≪問題解決・推論システム≫は，支援システム上の使用経験に基づき，核言語の拡張を行なう。意味記述，仕様記述系との接合を精密化する。試作された推論マシン上に処理系を試作し，試用に供し，評価データを集め，最終仕様原案を作る。この時，他基礎ソフトウエアシステムの成果の一体化を目差す。また新しい理論体系の枠組を描き出す。

≪知的インタフェース・システム≫は，開発支援システム上及び試作マシン上にプロトタイプを試作し，基本応用システムの研究・開発に供する。その使用経験に基づき最終仕様原案を作成する。この時，他の基礎ソフトウエアシステムの成果との融合を目差す。また新しい理論体系（認識論，理解論，表現論 etc.）の枠組を描き出す。さらに，能力向上のための種々のＶＬＳＩチップや入出力機器などのハードウエアの仕様を決定する。

② 知的システム化支援システム

≪知的プログラシング・システム≫は，中間目標プロトタイプとして，簡単な検証合成システム，小規模なプログラム・ベース，基本的なプログラム理解システムを作成する。これらは，整理され，支援システムの機能・拡充にあてられる。これらの成果を踏え，基礎ソフトウェアシステムと合せて，総合的なシステムの基本設計を行ない，目標システムの仕様原案を作成する。仕様，検証，合成に関する理論的な整備を行なう。

≪知識ベース設計システム≫は，中間目標プロトタイプとして，小規模なメタ知識表現システムとその推論・検証システム等を含む，設計システムを開発する。試用・実験を行ない基礎ソフトウェアシステムと整合させた総合的なシステムの基本設計を行なう。メタ知識表現論等の理論的な整備を行なう。合せて，支援システムの機能の拡充し，大量な知識収集の実験を行なう。

③ 基本応用システム

≪機械翻訳システム≫は，支援システム上に，中間目標システムとして，使用語彙数 5000 語の小規模な翻訳支援システムを作る。このシステムの開発と評価を通し，基礎ソフトウェアシステムと知識支援システムの仕様の検討データを出し，目標システムの仕様原案を決定する。意味記述辞書や用例集等のデータ・ベースの大規模化，知識ベース化を行なう。専用ハードウエアの試作を行なう。

≪質問応答システム≪は，中間目標システムとして，使用語彙 2000 語，規則 1000 個程の特定専門分野に限定したシステムを支援システム上に開発する。このシステムの開発と評価を通じ，基礎ソフトウエアシステムや知的支援システムの仕様の検討データを出す。他の基本応用システムとの整合性をとり，目標システムの詳細仕様の原案を作成する。各種専門家知識ベースの大規模化，高度化を進める。この中には，第５世代コンピュータ・システム自身に関するものも含まれる。また，スマート端末等の専用ハードウエアの試作も行なう。

≪音声応用システム≫は，単純文を対象とする音声タイプライタ等の中間目標システムを支援システム上に開発する。このシステムの開発と評価を通じ基礎ソフトウエアシステム，知的支援システムの仕様の検討データを出す。目標システムの詳細仕様の原案を作成する。専用ハードウエアの開発も行なう。

≪図形・画像応用システム≫は，１万枚程の図形・画像を対象とした中間目標システムを

支援システム上に開発する。このシステムの開発と評価を通じ，基礎ソフトウエアシステムと知的支援システムの仕様の検討データを出し，他システムの整合性を考慮した目標システムの詳細仕様の原案を作成する。図形・画像データベース・マシン等の専用ハードウエアの試作も行なう。

≪応用問題解決システム≫は，数式理解システムは，ルール・ベース化した中間目標システムを，基プレイング・システムは１０級程の中間目標システムをそれぞれ支援システム上に作成する。このシステムの開発と評価を通じ，基礎ソフトウエア・システム，知的支援システムの仕様の検討データを出し，目標システムの詳細仕様の原案を決定する。ルール・ベース等の高度化を図り，専用ハードウエアの開発も行なう。

(3) 後　　期

① 基礎ソフトウエア・システム

≪知識ベース管理システム≫は，目標機のシミュレータを中間目標試作マシンあるいは支援システム上に作成し，知識ベース管理システムを完結するためのソフトウエアの開発を行なう。この開発の過程においても，適宜，仕様の改良，精密化をすすめる。作成されたシステムは適宜，目標機上に移され，基本応用システムの開発に供される。この過程で得られたデータに基き，より高度な学習機能などの付加を目差し，拡充のための研究・開発を行なう新理論の確立を目差す。

≪問題解決・推論システム≫は，目標機のシミュレータを中間目標試作マシンあるいは支援システム上に作成し，目標機の処理系の開発を行なう。この処理系は，知的プログラミング・システムと一体化され，目標機上に移され，他システムの開発に利用される。得られたデータに基づき，より高度な問題解決機能の付加や，マシンの能力向上を目的とする研究・開発を並行して行なう。さらに新理論の確立を目差す。

≪知的インタフェース・システム≫は，目標機シミュレータを用い，最終目標システムを開発する。これは，専用ハードウエアの開発も含む，知識ベース管理システム，問題解決・推論システムと一体化され目標機上に移され，基本応用システムの目標システムの一部となる。得られたデータに基づき，より高度な理解機能の付加を目差す研究・開発も並行して行なう。さらに新しい理論の確立を目差す。

② 知的システム化支援システム

≪知的プログラミング・システム≫は，自然言語による質問・応答をベースにした，プログラム設計コンサルタント・システムを目標機上に開発する。このシステムには，大規模なプログラム・ベース，改良・修正システム等が含まれる。この開発は，基本応用システムの開発にも一部利用できるよう手順を追って進められる。仕様論，検証論，合成論等の新理論の確立を目差す。

≪知識ベース設計システム≫は，総合的な知識ベース設計支援システムを目標機上に開発する。このシステムには，増殖支援システム，実行支援システム等が含まれる。この開発は，基本応用システムの開発にも順次，利用できるよう手順を追って進められる。また，知的プログラミング・システムとの接合も調整され，両者相まって総合的なシステム化支援システムとなる。メタ知識論，検証論等の新理論の確立を目差す。

③　基本応用システム

≪機械翻訳システム≫は，１０万語彙，多国語間の翻訳をほとんど機械処理できる最終目標システムを目標マシン上に開発する。開発に当っては，基礎ソフトウエアシステムとの整合を計り，各種知識ベースの整理と高度化を行なう。専用ハードウエアの開発・改良も行なう。

≪質問応答システム≫は，使用語彙 5000 語以上，規則１万個以上の各種専用分野に対するシステムを最終目標システムとして目標マシン上に開発する。開発に当っては，基礎ソフトウエアシステムや他の基本応用システムと調和を図り，各種知識ベース高度化と整理を行なう。目標システムには，知的コティリィ・システムも含も。

≪音声応用システム≫は，音意理解機能をもつ，音声タイプライタ等の最終目標システムを目標マシンの高機能入出力機器として開発する。開発に当っては，知的インタフェース・システムとの整合を図る。

≪図形・画像応用システム≫は，１０万枚を対象とする図形・画像データ・ベース及び総合的な検索システムを最終目標システムとして目標マシンに向け開発する。開発に当っては，知的インタフェース・システムとの整合，他基本応用システムとの整合を図る。

≪応用問題解決システム≫は，数式理解システムは，法則，公式等を知識ベース化した大規模な数式処理システムを，基プレイング・システムは初段程度のシステムを最終目標システムとして目標マシン上に開発する。

4.3　研究・開発支援ネットワーク・システム

第５世代コンピュータ・システムの研究・開発には，次節以後に述べるような支援ネットワーク・システムが必須である。ただし，その説明には，理想化した所が何ケ所かある。これらの理想を単なる理想に終らせないのも本プロジェクトの大きな目標である。

なお，後述する支援パーソナル・コンピュータは，研究者のマンパワーに対応する。本プロジェクトに必要とされる研究者数は，前期から中期にかけ，２～３百名，中期から後期にかけ２～３千名と推定される。

4.3.1　構　　成

全国及びヨーロッパ数拠点に多数の研究サイトを設ける。全国の研究サイトは，全国ネット

ワークで結ばれる。全国ネットワークとARPANET，ヨーロッパの拠点は，衛星通行で結ばれ，世界的な規模での研究・開発体制の礎が築かれる。

全国ネットは，出来うる限り全国を隈無くおおうべきである。

各サイトには，数台～数十台の高機能パーソナルマシン（後述）が置かれ，サイト内では，ローカル・ネットワークによって結ばれる。大型の装置は何ケ所かに置かれ自由に共同利用される。大型装置としては，例えば，VLSI試作システム，高精度グラフィック・ディスプレイ及びプリンタ，高速漢字プリンタ，大容量ファイル・システム等が考えられる。超大容量データベース・システムが中央に置かれ，全システム共用の情報バンクとして機能する。

各サイトは，各レベルのネットワークを駆使して，それぞれのサブ・テーマに対し，様々な形態の共同研究のグループを構成することができる。各サイト，各グループは，一定期間内に一定量以上の研究成果の登録・報告の業務を負う。研究成果の登録は，プログラムやデータや論文を中央のデータベースに書き込み，公開することによってはたされる。中央のマネジメント機構は，プロジェクトのマネジメント及びネットワーク・システムの維持・管理を行なう。各サイト，各グループの機器の使用状況，研究の進捗状況は，半自動的にマネジメント機構に報告され，高精度な判断を可能にする。マネジメント機構は，研究交流，成果の相互利用の促進役としても重要な役割を担う。データベースに登録された研究成果を，興味を持つグループに紹介し，成果を活用，追試しさらに成長させるというサイクルを円滑に恒常的に回転させるということも大きな役割である。

いささか，マネジメント機構の管理能力の強力さを強調しすぎたが，これは，このような分散化された研究体制（開かれた研究所）では，研究管理が充分に行ないえないという危惧に対して，むしろ逆に，従来よりはるかに強力で柔軟な管理が可能になるということを示さんがためである。本プロジェクトのように非常に先進的なものに対しては，また，日本の風土を考慮して，大多数の研究員納得の行くような柔軟な管理体制が望まれる。

4.3.2 高機能パーソナル・マシン

ネットワークの基本構成要素となる高機能パーソナル・マシンを何に，どのように定めるかは，本プロジェクトの要となる。その一大方針を以下のように定める。

“研究・開発用のプログラム言語を一種類に定める。高機能パーソナル・マシンは，その言語を対象とする高級言語マシン（ノイマン型）とする”

そのプログラム言語を何に定めるかが次の判断である。代表候補はLispとPROLOGである。言語の安定性という面からは，はるかにLispが優る。成長過程でいえば，Lispは完成期にあり，PROLOGは誕生間もない初期の段階にある。しかし，本プロジェクトの１０年間さらにその後少なくとも１０年間，今後２０年間，この分野の共通言語として使用に耐えるも

のという条件から，Lisp 等を素直に包合しうるという面からも，その発展性という観点からは，PROLOGが優る。そこで次のように定める。

“プログラム言語はPROLOGを母体とし，Lisp 等の機能を包含，融合させたものとする”

そのような言語の仕様，実装方式，アーキテクチャ化の方式等は，1年間程の検討によって，十分確定しうるものと判断される。もちろん，確定するといっても基本部分だけであって，言語自身は，その永い使用過程の期間を通じ，改良・変更されていく。当然のことながら，高機能パーソナル・マシンもその変更に対応できるように設計される。

次にその言語の改良の方向付けとして，次の方針をかかげる。

“この言語は，目標機の言語仕様のサブセットとなる。”

この条件によって，開発支援システム上で開発されたプログラムは，若干の修正をする。あるいは機械的に変換するだけで，目標機上に移殖することができる。

なお，高機能パーソナル・マシンの物理的な条件の主なものを列挙すると次のようになる。

① 主記憶は，数MW以上の実メモリを持つ。

② ローカルなファイル記憶として100MB以上。

③ 高精度なラスタ・スキャン・ディスプレイ装置。

これらは，すべてのマシンについて共通の条件である。利用目的によって，色々な装置や機能を自由に付加できるようにも設計される。

4.3.3 その効果

この開発・支援システムは，単に，第5世代コンピュータ・システムの研究・開発に不可欠であるというばかりでなく，そのもたらす恩恵や利益には，図り知れないものがある。

①プロジェクトにとって

誰もが指摘するとおり，本プロジェクトを遂行するためには，日本中の英知を集めねばならない。その唯一の方法が，この支援ネットワーク・システムである。

②研究者にとって

我が国の大学人，研究所員，誰もが焦がれ，多くの人々が実現を試み，はたせなかったARPANETを，一段進んだ形態で一気に実現することになる。少なくとも，研究設備に関しては日米間のギャップがとり払われる。今だ，日米間で5～10年の差があるといわれているソフトウエア・ギャップを縮める唯一の手段を手に入れることになる。

③メーカにとって

本プロジェクトの目標機が商品化されるのは，10数年以上先のことになるであろう。しかし，支援システム，特に高機能パーソナル・マシンは5～10年以内に相当量の商品化が見込まれる。しかも，商品化にとって重要なソフトウエアの開発は，日本中の英知を集め，自動的

にこのネットワーク上に蓄積されるものを利用することでまかなうことができる。

この支援システム及びその上に開発される本プロジェクトの中間目標の実現は，各メーカが現在，持っている長期研究・開発（５～６年規模）計画をすべて包含しうるものである。したがって，本プロジェクトは非常に革新的な目標をかかげ長期にわたって基礎から積み上げるという画期的なものであるが，目標にいたらねば，成果が得られないというものではなく，プロジェクトの各段階で色々な成果（商品化も含め）が得られるものとして組立てられている。開発・支援システムは，メーカに対する格好のオブラートとなる。

④行政サイドにとって

支援ネットワーク・システムは，第４世代（機能分教システム等）システムのヒナ型である。この第４世代機と目標の第５世代機を結びつけ方向付けることにより，今後のコンピュータ産業の発展の大筋を完全に方向付けたことになる。第Ⅰ，第Ⅱ世代という議論からすれば，第Ⅰ世代では，ＩＢＭ機にコンパチブルにするということで，どうにか規格統一がはかられた。その経緯に比べ，これからの第Ⅱ世代に対しては，世界に先がけ規格統一の方向を示すという大きな判断がまずなされる。

これから本格化する情報化社会においては，情報処理・伝達技術の高度化を一般化（大衆化）を土台にして，あらゆる組織体が色々なレベルでの質的な変化，脱皮をせまられる。この支援ネットワーク・システムを基盤とする研究・開発組織体は，情報化社会における新しい組織体のヒナ型として，来たるべきものを世に知らせることになる。

5. 使われ方のイメージ

5. 使われ方のイメージ

5.1 知識利用，質問応答システムの具体像

(1) MYCINプログラムを用いた簡単な例，知識ベースに基いたコンサルテーションの考え方を説明するために，Stanford大学で開発されたMYCINと呼ばれるプログラムを使うことにしよう。MYCINは病気の診断と療法のためのプログラムである。

このプログラムは伝染性疾患，特に血液感染と髄膜炎疾患の診断を行い，この感染症を治療するための抗生物質による療法を医者に助言する。

MYCINはユーザである医者へコンサルテーションを行うのである。この医者はMYCINを作った医者，すなわちその病気のエキスパートとは別の専門である。エキスパートはルールをMYCINの知識ベースへ入れる人である。ユーザは対話という形で，すなわち最終的には診断と療法に到達するようなインタラクティブなコンサルテーションで，このルールを利用する。コンサルテーションはある特定の型にはまった英語で行なわれる。従って医者はそこで使われるLISPプログラムに関して何も知らなくてよい。コンサルテーションの中で，医者は患者の病気と臨床検査結果（すなわちコンピュータが推論できないような外因的データ）だけを答えればよい。

MYCINのようなプログラムは，結果を導く一連の推論を見つける定性的推論を用いている。下した結論を逆にたどることによって，MYCINは推論の流れをユーザに説明できる。実際，専門家のコンサルテーションシステムは，この説明の機能を持たなければならないというのが基本的な考え方である。そうしないとその領域の専門家はユーザに信頼されないだろう。

If: 1) The infection which requires therapy is meningitis, and
2) The type of infection is fungal, and
3) The patient has been to an area that is endemic for coccidiomycoses, and
4) The race of patient is one of:black asian indian

Then: there is suggestive evidence that coccidiomycoses is one of the organisms which might be causing the infection.

Figure 5—1 A piece of knowledge in MYCIN

もし：1) 治療を要する感染症が髄膜炎であって，
2) 感染症のタイプが真菌であって
3) 患者がコクシディオミコーシス（真菌の一種）にあたる風土病の地域にいたことがあって
4) 患者は黒色，アジアインド系人種であるとすると

そのとき：コクシディオミコーシスが感染症を起した原因菌の有力な候補と考えられる。

図5－1 MYCINにおける知識の例

(2) MYCINの知識

図5－1はMYCINの知識の1つである。MYCINには約500個のルールがある。その内約半分が血液感染病のルールであり，残りは髄膜炎疾患のルールである。それぞれの「プロダクションルール」は「IF」の部分と「THEN」の部分（それぞれ「条件部」と「実行部」と呼ばれることがある）から成っている。「IF部」がもし各項目が真ならばというような適切さを表わす一組の条件を定義し，その後に結論が続く。

図5－1にはエキスパートがルールを入力するのとほぼ同じ形でMYCINのルールが書かれている。逆にディスプレイして医者に示されるルールの形式はここに示したのと全く同じである。この知識は条件が真ならば知識ベースから呼び起され，推論プロセスに組み込まれる。

(3) 不確実さを含む推論

MYCINには，その前提からその結論が導かれるということを，エキスパートがどの程度確信しているかを0.1から1.0までのスケールで表わした方法が備わっている。

たとえば，1.0は問題なく確かな場合，0.9は「非常に有力」な場合，0.6は「かなり明らか」な場合などを表わす。この指標はMYCINの中で非常に簡単かつ理解しやすい関数で計算され，「確からしさの総合的係数」という推論の過程における「確信度」の指標を生み出す。

```
INFECTION-1 is MENINGITIS
  §ITEM-1† E.COLI〔Organism-1 from CSF〕
  §ITEM-2† PSEUDOMONAS-AERUGINOSA〔from clinical evidence
           only〕
  §ITEM-3† KLEBSIELLA-PNEUMONIAE〔from clinical evidence
           only〕
  §ITEM-4† DIPLOCOCCUS-PNEUMONIAE〔from clinical evidence
           only〕
```

Figure 5－2：An example of a MYCIN diagnosis

```
感染症は髄膜炎である
  §ITEM1†  大腸菌〔髄液からの菌体－1〕
  §ITEM2†  緑膿菌〔臨床所見のみから〕
  §ITEM3†  肺菌桿菌〔臨床所見のみから〕
  §ITEM4†  肺炎双球菌〔臨床所見のみから〕
```

図5－2　MYCINの診断

(4) MYCINの診断と治療法

図5－2にMYCINの典型的な診断を示す。このリストを表示した後，MYCINは患者のいろいろなタイプの抗生物質に対する感性とか，ある病源体によって示されるいろいろな種類の抗生物質に対する抵抗力などについて，短いコンサルテーションを医者に行う。そこでMYCINは図5－3に示すような療法の助言を生成する。

My preferred therapy recommendation is follows:
In order to cover for Items §1 2 3 4†:
Give the following in combination:
1) AMPICILLIN
Dose: 3.5g(28.0 ml) q4h IV〔calculated on basis of 50 mg/kg〕

2) GENTAMICIN
Dose: 119mg (3.0ml, 80mg/2ml ampule) q8h IV〔calculated on basis of 1.7mg/kg〕 plus consider giving 5mg q24h Intrathecal
Comments: Monitor serum concentrations

Since high concentrations of penicillins can inactivate aminoglycosides, do not mix these two antibiotics in the same IV bottle.

Figure 5－3：An example of a MYCIN antibiotic therapy recommendation

次の療法を推める：
ITEM1～4† 全てに当てはまるために，以下の抗生物質を併用すること：
1) AMPICILLIN
投与量：4時間おきに3.5g（28.0ml）を静注（静派注射）。〔ただし体重1kg当り50mgで換算〕。
2) GENTAMICIN
投与量：8時間おきに119mg（3.0ml，2mlのアンプルに80mg）を静注。〔ただし体重1kg当り1.7mgで換算〕。
注意事項：薬物の血清濃度をモニタすること。
高濃度のペニシリンはアミノグルコシダーゼを不活性化するので，上記2剤をいっしょに点滴びんに入れないこと。

図5－3　MYCINによる抗生物質の治療例

(4) MYCINにおける推論手順

MYCINによる一連の推論は，臨床試験と患者の病気から，感染性病原体の存在について結論を導くルールの連鎖であるこの一連の推論はバックワード・チェイニング（逆向き推論）によって見出される。一般的な「達成すべき目標」として，「感染媒体」を設定して探索を開始

し，　カテゴリの異なる病原体（たとえば菌類に対してバクテリヤ）を通して逆にたどり，そしてデータへ至る。

説明機能は利用者の要求に応じてその連鎖の指定された部分を説明することができる。コンサルテーションのやりとりの間や，コンサルテーションが終った後の次のようないくつかの種類の質問応答をすることができる。たとえば「なぜ（このような質問をするのか？）」とか，「どのようにして（その結論に達したのか？）」というような質問である。図５－４に質問応答の面白い例を示す。MYCINはこれに答えるために採用された推論のすじ道を保存しておくだけでなく，調べてみて，受入れられないことが判った推論の過程も理由をつけて憶えておかなくてはならない。

USER: WHY DIDN'T YOU GIVE TETRACYCLINE FOR E.COLI IN REC−1?
MYCIN: TETRACYCLINE was discounted for ITEM−1 (RECOMMENDATION−1) because there is evidence that this e.coli is not sensitive to it.

Figure 5−4 An example of MYCIN's explanation facility

ユ　ー　ザ：上記の療法のうち，大腸菌に対しては，なぜテトラサイクリンを投与しないのか。

MYCIN：テトラサイクリンは療法－１には入れていない。というのは，大腸菌はテトラサイクリンに不感性であるからである。

図５－４　MYCINの説明機能

(6)　MYCIN自身に関する知識

MYCINが一連の推論を説明するために，生成する質問，ルール，ゴール等の相互の関係を知っていなければならない。これを知っているという事は，MYCINが「自分自身何を知っているかを知る」ことであり，メタ知識 (Davis & Bucha 1977)を持つことである。MYCINの中で最も重要なメタ知識，ルールの各前提と結論の関数に付随している「テンプレート」である。このテンプレートのおかげで，プログラムはルールを分解して要素を識別しながらルールを読む事が可能となる。このように，MYCINは図５－１に示したルールの第２項が，「感染病のタイプ」というパラメータに関したもので，等号関係を用いており，「fungal」という値とマッチングがとれるということを決められる。この機能は，英語形式のルールから新しいルールを構成し，ルールがなぜ適用できないかを説明し，そしてルールを教える手法のための本質でもある。

(7) MYCINの推論手続き

MYCINの知識ベースを取り去って，その代りに他の問題領域のルールセットに置き換える事が可能である。このことは，知識ベースと推論手続きとは，エキスパートシステムにおいては互いに独立であることと等しい。

MYCINから感染性疾患の診断ルールを取除くと，推論「機関」が残る。これがEMYCINである。

5.2 知的プログラミング・システムの具体像

将来のプログラミングは，知的プログラミング・システムによって，その作業内容が大きく変わり，一方では素人でも容易に計算機が利用できるようになると同時に，他方では専門家あるいは研究者にとっても，より複雑なシステムの開発が容易になるであろう。

ここでは，そのような知的プログラミング・システムが，どのようなものであるのか，また，それが知識利用システムとしてどのように実現されるのか，そして，それがどのように利用されるのか，などについての1つのイメージを描くことにする。さらに，そのようなシステムによって，現在かかえているソフトウエアの諸問題がいかに解決されるかについて予測をしてみよう。

5.2.1 知的プログラミング・システムとは，

知的プログラミング・システムは，プログラムの設計・開発・保守の各段階をサポートするいくつかのサブ・システムから成る。以下に，それらのサブ・システムを列挙する。

(1) 設計支援サブ・システム

設計段階では，要求仕様の記述，それに基づくシステム設計などのサポートが必要となるが，本サブ・システムには，つぎの3つの言語が用意されている。

① モデル化言語

② 仕様記述言語

③ 問題解決言語

①は処理対象の性質を抽象的に記述するための言語で，処理対象が何であるかということと，それらに対してどのような処理が施され，どんな効果が得られるか，の2つの面からの記述を行う。このモデル化が十分整理された形でなされれば，その応用分野を処理するのに適した抽象機械が設計できたことになる。

②は作製すべきプログラムの仕様を記述する言語で，プログラミングの専門家でない人の利用を考えると，究極的には自然言語がそのターゲットである。

③は，問題の解決をgoal-orientedに記述し，それによって，元の問題をより小さな一

連の部分問題へ展開するための言語である。

これら3つの言語は互いに関連している。②は，①でモデル化された対象を用いて，その仕様が記述される。すなわち，処理対象がプログラムによってどのように加工されるのかが述べられる。また，③の記述は，①で求められた抽象機械の基本命令を用いてなされる。

設計支援サブ・システムは，これら3つの言語の他，それらの言語を用いて，どのように設計をしていったらよいかの指針を与えるknow howも，その重要な構成要素である。それらのサポートとしては，自然言語による設計コンサルティング・システムがある。

利用者が自然言語で問題の仕様を逐次与えると，設計のknow howをもったコンサルティング・システムの手助けによって，自然にモデル化がされていくことになる。

(2) 開発支援サブ・システム

開発段階では，設計書に基づいてプログラムを作製し，論理的機能の検証，性能の評価を行う。プログラムの作製は，部品化されたモジュールの組合せにより行う。その際の仕様および解法は，(1)で述べた言語により記述されている（これが設計書となっている）。論理的機能の検証は，プログラムが設計仕様を満しているかどうかのチェックによってなされる。部品化された各モジュールは，その機能が明記されているので，この検証は組合せの効果を中心に調べればよい。開発サブ・システムのより進んだ形態では，これらがすべて自動化されている。

(3) 保守支援サブ・システム

プログラムの保守段階では，システムの性能評価に基づくプログラムの改良，あるいは仕様変更に伴うプログラムの修正などを行う。プログラムの改良あるいは修正を行うには，プログラムの挙動と仕様との関連づけを常に保っておくことが必要である。プログラムを改良するには，仕様を変更せずに性能の向上を図ることが必要である。そのためには，プログラムの性能面での隘路を検出し，その部分を，もっと性能のよい部品で置きかえればよい。一方，プログラムの修正は，仕様の変更がプログラムのどの部分に対応するかを調べ，その部分のみを修正すればよい。

5.2.2 知的プログラミング・システムの仕組

前節で述べた知的プログラミング・システムは，どのような仕組になっていて，どのようにして実現できるであろうか。

ここでは，それが知識利用システムであり，プログラミングに関する専門知識をシステムに蓄積することによって，それが作られるということを主張したい。

前節で述べたモデル化およびプログラミングの作業は，応用分野毎に異なるものであるが，その結果は，各応用分野に関する知識として，システムに蓄えることができる。このようにして，各分野毎の知識ベースが構築される。これらの知識ベースは対象指向のプログラム・ベー

スあるいはモジュール・ベースと考えられる。

このようなモジュール・ベースと目的とするプログラムの仕様から，プログラムを自動作成するには，モジュールの組合せの効果をプログラミングに関する知識として与え，適当な問題解決システムによって，その知識を手掛りとした解の探索を行えばよい。これは知識利用システムそのものである。

プログラムの保守支援サブ・システムにおいても，管理し，把握すべき情報は，そのプログラムに関する知識と考えられ，当然知識ベースで管理される。仕様の変更は，プログラム合成過程での推論の変更を引き起こす。すなわち，ここでは，プログラム合成過程の推論自身とプログラムの対応を知識としておけばよい。これは一種の推論根拠管理システム（Truth Maintenance System）と考えられる。

5.2.3 部品プログラミングの例

ソフトウエアの部品化がどのような単位でどのようにできるかのおおよそのイメージを描くために，部品の例と，結合の例を示そう。ここで挙げる例は，データベース管理システムを作り上げるときの例である。部品は，より基本的なものから，より応用寄りのマクロなものまで，いくつかのレベルに分けられる。それらの部品を以下に列挙する。

(1) 基本部品（記憶管理用）

・動的記憶管理ルーチン

・Hash Table

・Stack

・Symbol Table

これらの部品は，通常は，言語プロセッサに埋め込まれているものである。別の言葉で言えば，これらの部品は，システム記述言語プロセッサを作るときに，その中に組み込まれるものである。たとえばＬＩＳＰには，これらの機能がすべて組み込まれている。

(2) データ管理部品

・Ｂ－treeパッケージ

・逐次ファイル・パッケージ

・ソート・ルーチン

・サーチ・ルーチン

これらの部品は，データベースの核となるファイル管理パッケージを作るのに便われる。ディスクやテープなどの２次記憶管理もこの中に含まれる。

(3) データベース組立部品

・関係代数パッケージ

・質問最適化ルーチン

・データベース定義ルーチン

・データベース更新パッケージ

これらの部品は，データベース管理システムを作るために使われる。部品のレベルは，(b)の部品群に比べて一段高い。

(4) ユーザ・インタフェース

・ＱＢＥコンポーザ

・ＱＢＥコンパイラ

・自然言語アナライザ

これらの部品は，(3)よりもさらにユーザ寄りの機能を実現するための部品である。言語レベルは，(3)が手続き的であるのに比べて，(4)では非手続き的である。

(5) ディスプレイ管理

・ディスプレイ・エディタ

ディスプレイ・エディタは，ＱＢＥや自然言語入出力のためのディスプレイ装置の画面管理に用いられる。

図５－５に，これらの部品間の結合例を示す。部品を□で表わし，インタフェースを⬭で表わすことにする。また横方向の結合が水平結合で，縦方向の結合が垂直結合である。水平結合は関数の合成に相当し，垂直結合はデータ抽象ルに代表されるモジュール結合に相当する。

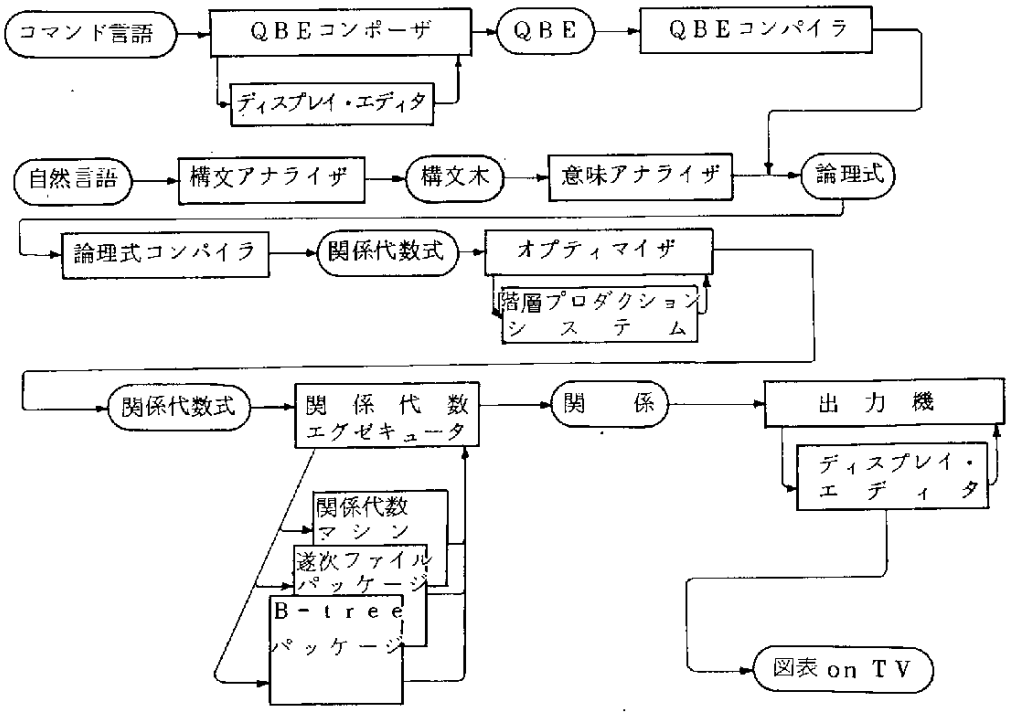


図５－５　関係データベースの組立て例

図５－５には，記憶管理用の基本部品がどこにも表われていないが，それは各部品の下に垂直結合されているものと考えてよい。それは，これらの部品がＬＩＳＰなどの基本部品を含んだ言語によって作られているからである。

5.2.4　ソフトウエアの諸問題への対処

ソフトウエアの問題は数多く，すべてを列挙することはできないが，ここでは，つぎの４つの点について考える。

(1)　大規模ソフトウエアの開発が困難なこと．とくに，虫とりあるいは，仕様の変更に伴うプログラムの修正作業は，プログラムが大きくなればなるほど困難となる。

(2)　正しいプログラムを作ること．あるいはプログラムの質を保証すること（品質管理）が困難であること。プログラムの質は，論理的正しさと，プログラムのからくりの適切さによって決まる。論理的に正しく，しかも簡単なしかけのもので，計算時間や記憶容量の無駄使いをしないものがよい。これらの検証は容易でない。

(3)　素人によるプログラミングが困難なこと。

プログラミング言語は一種の言語であり，それを使いこなすには，外国語を修得するのと同様，相当な訓練が必要とされる。とくに現在の手続き型の言語では，計算機の働きに対する一通りの理解が必要で，計算機への指令書としてのプログラムを書くことが要求されている。

(4)　現在のソフトウエアは柔軟性が乏しく，決められたことしか実行できないこと。ＣＡＤやＤＳＳなどの分野では，データやデータ間の関連の利用のし方が問題によって大きく異なってくる。従来のプログラムでは，データなどの利用のし方を中心に記述するが，そうすると異なる利用方法の数だけのプログラムが必要となる。

これらの問題が，知的プログラミング・システムで，いかにして解決され得るかを見てみよう。

(1)の素人によるプログラミングの問題は，設計支援サブ・システムに関連している。素人がコンピュータによって問題の解決を図りたいときに一番困るのは，その問題を，コンピュータ言語によって表現しなければならないことである。そこでの困難性は，実は２種の要因が考えられる。第１の要因は，問題自身が十分に理解されていないことが多いことである。設計支援サブ・システムは，自然言語による対話型の設計コンサルテーション・システムをその一部として持っているが，それは，この問題の分析手法のknow howを知識として持っており，利用者のもっている断片的な要求を元にして，問題の整理を手助けする。このようにして，問題のモデル化が進行する。第２の要因は，解法の記述に用いる従来のコンピュータ言語が，人間の思考に適合していないことである。この点は，設計支援サブ・システムの言語の１つである問題解決言語によって，大巾に改善され得る。これは一種の思考型言語であると考えられる。こ

の言語によって，プログラミングのスタイルが，従来の手続き型言語によるスタイルと全く変ってくる。従来の言語では，解法は，コンピュータ上に投影したモデルの操作のための一連の操作例として記述され，そのプログラムの各命令がどのような意味を持つのか，あるいは全体として何をやるのかといった事柄は，付随するドキュメントにしか記述され得ない。ここに現われるプログラムと仕様とのギャップが，素人を最も悩ます元凶である。プログラムの働きを理解するためには，プログラムの行間をじっくり読み，コンピュータ上に投影したモデルの動きを追ってみなければならない。

一方，思考型言語では，プログラムが何をやるか，またその解法がどのようなものかを直接記述する。解法が直接記述できることは，その言語によって解法自身を考えることができることをも意味する。すなわち，その機能によって問題の分析が進み，問題のモデル化にも有効である。

(2) プログラムの質の向上を図る問題は，開発支援サブ・システムに関連している。論理的な正しさの検証は，モジュール化技術によって著しく容易になる。とくに垂直方向のモジュール化技術によって，プログラム全体の証明が抽象レベルのプログラムの証明と，抽象機械の実現モジュール群の証明とに直交分解され，それぞれの証明は，分解前に比べると非常に簡単になる。モジュール化に伴って実行速度等の性能は多少落ちるが，その分は，他の利点が十分にカバーする。さらに，このモジュール化は標準化にもつながり，ハードウエア化も可能となるので，性能面での欠点も補われるであろう。プログラムの他の面の品質は，つぎに述べる柔軟性や可変性などと関連する。それらは，いずれも，プログラムの構造によって決まるものである。そして，そこでは，(1)で述べた思考言語と，ここで述べたモジュール化の２つが，重要な働きをすることが分かる。

(3) ソフトウコアの柔軟性の追求は，保守支援サブ・システムに最も関連が深いが，それは，作られるプログラムの構造の問題でもある。(1)で述べた思考型言語によるプログラムの構造は，この点，非常に優れている。従来の手続き型言語のように，データの利用の仕方を中心に記述する方式を採れば，異なる利用方法の分だけのプログラムが必要となる。たとえば，建築でのＣＡＤで，２次元上の点の集合を与えて，その中に正方形がいくつあるか，矩形はいくつか，あるいはＬ字型はいくつかを求める問題があったとする。すると，この問題毎に，２次元上の点の集合を調べる別々のプログラムが必要となる。一方，これを思考型言語の１つであるPROLOGで記述をすると，点の集合が水平線分あるいは垂直成分を成しているかどうかを調べるモジュール群によってこれら各プログラムが簡単に記述できる。さらに，それらの各モジュールは各々独立しており，それらの追加，削除，修正は，データベース中のデータと同様に容易に行うことができる。たとえば，利用者がプログラムの利用時に，Ｔ型の検索も必要になったとすると，プログラムの変更は全く必要とせず，上記の水平・垂直を調べる基本モジュールを用いた主プ

ログラムを書いて，それを単純に実行しさえすればよい。プログラム部分の結合は実行時にパターン照合を基にして行われ，実行前にコンパイルあるいはリンケージ操作を全く必要としない。これは，従来の手続き型の言語では全く不可能である。

(4) 大規模ソフトウエアの保守の問題は，(3)と同様，保守支援サブ・システムに関連していると同時に，プログラムの構造に大きく依存している。あるいは，保守支援サブ・システム自身，プログラムの構造が鍵になっているとも言えよう。

いずれにしても，この問題は，保守支援サブ・システムの目差している機能そのものであり，これ以上の説明を要しない。

ソフトウエアの問題は難しいと一般に言われている。その進歩は到底望めそうもないと断言してはばからない人も多い。しかしながら，以上述べたように，将来知的プログラミング・システムの実現によって，事態は大幅に改善されることが十分に期待できる。言語に思考の助けとなることは，数学を見ても明らかである。新しい言語の発明が，ソフトウエアの世界でも質的な転換をもたらすと言えよう。それは，設計の段階にも及ぶのである。新たな言語は，新たなプログラムのスタイル（パラダイム）を持っている。プログラミング方法論の進歩は，新たなパラダイムの発明によってなされてきた。Fortran などの原始的な言語の提供し得るパラダイムは，くり返し計算のみと言ってよいが，その後，ＬＩＳＰ で再帰計算が実現され，さらにco-routine，非決定プログラミング（PROLOG），プロダクションなどの強力な計算機構が発明されてきている。

新しい発明は，現在のところ，確かに性能の面で，従来のFortran に代表されるプログラミングに追いつかないかも知れないが，プログラムの開発を含めたコストを考えれば，確実に新プログラミングが追いつき，追い越そうとしている（Fortran自身，アセンブラと比べると実行速度は劣るが，生産性でそれを補っている）。さらに，複雑なソフトウエア・システムの開発は，Fortran などでは，すでに不可能になっている。

新プログラミング・スタイルに適したコンピュータの開発は，新しい方向をさらに加速することになるであろう。

5.3 知的ＣＡＤ／ＣＡＭシステム

5.3.1 設計プロセス

一口にＣＡＤ／ＣＡＭと言っても対象の範囲は広く個々のケースにより設計手法も異なるのが普通である。たとえば電子回路の設計，機械的形状設計，システム，材料設計など，すべて設計と呼ばれているが，その内容は相互に非常に異なるように見える。しかしさらに考察を進めれば，これらの間に，対象の相違を越えて設計という一つの共通の概念が左右することも明

らかである。

以下ではこの共通概念にある設計というプロセスに，いかに計算機が関与するかについて考えてみる。

まず対象の相違を越えた設計の本質は，

(1) 現存しないものを創造するプロセスであることである。一般には，設計され，将来実現されることが期待される（仮想の）ものについて，それが満たすべき条件（要求仕様）が与えられ，それを満たすものを創り出すことが設計者の仕事であるが，このプロセスは物が存在する以前に，それに関する情報が作られることに特徴がある。この設計プロセスは

(2) 設計対象のモデルが設定され，そのモデルが与えられた要求仕様を基準に評価され，修正されるということの繰り返しである。ここで前述の対象による相違はモデルとその評価方式の表現の相違と考えれば，すべてこの形式を満たしている。モデルやその評価手順はともに情報であり，この意味で設計とは純粋に情法処理のプロセスである。設計されるべき対象によってモデルの表現が異なるので，モデルの表現手段が重要になる。従来，この表現法は多くの経験を通して，対象ごとに最も適した表現法が用いられている。またこれは特定の対象についても一種類ではない。要求仕様に与えられる様々な面に関する条件に対応して，多面的な評価は必要であるが，個々の評価目的に適したモデルの表現が作られる。例えば形状設計の場合の概案図，三面図，線図，構造図，部品図等である。

次に設計というプロセスが多くの場合複数の人間によってなされるという点も留意する必要がある。この場合モデルは単に設計者のイメージの表現であるだけでなく関係者間のコミュニケーションの手段でもある。

設計プロセスにおいて，モデル表現のみが変化するように見える。しかし時には評価の部分に実験等が含まれ，その結果に応じて人間の知識そのものが増大し，評価方法に影響し，さらには要求仕様の変更ということも生じ得る。この意味で広義の設計とは

(3) 人間の知識開発の可能性を含むプロセスとするべきであろう。

要求仕様を満たすモデルに到達した後，実際にこれを製造することになる。この時モデルの表現はこれを実際に製造するマシンを制御する情報に変換されねばならない。以前のようにすべてのマシンを人間が制御する場合，人間間のコミュニケーション手段としてのモデル表現がそのまま製造時のマシン制御情報をも与えたが，マシンの自動化が進んできた現在，この表現間の変換が必要となった。

このように設計・製造というプロセスは最終段階の情報から物への変換（製造）を除けば情報処理プロセスであり，これに計算機を利用するのがＣＡＤ／ＣＡＭである。他の多くのアプリケーションの場合と同様，このプロセスを一貫して計算機用処理できることがCAD／CAMにおいても目標とする姿である。途中での入出力とそのための変換は極力少なくすべきであり，

このために計算機向き設計プロセスの開発が必要である。ここで最も基本的な問題はモデルの表現である。

5.3.2 ＣＡＤ／ＣＡＭに課せられる条件

設計という仕事が（試行錯誤を行ないながら）モデルを要求仕様を満たすように変更するプロセスとすると，当然モデルの表現がこの中心に存在する。現状はこの表現を対象に応じ，図面，言語，記号，数式等で表わしており，モデルが変更される都度，人手によって新らしいモデルを表現せねばならずここに多くの時間と労力を要していた。この改善がＣＡＤ／ＣＡＭにたいする第１の条件である。

モデルが変更される度にその評価が行なわれるが，設計対象が複雑になるにつれ，この評価プロセスに多くの時間と人手を要する。この改善がＣＡＤ／ＣＡＭにたいする第２の条件である。なお一般には評価は多面的であり，それらが別のグループによりなされ，しかも相互に関連している。たとえば航空機を設計する時，性能面での要求を満たすために，機体形状，エンジン推力，機体重量などにたいする変更が出されると，これは構造，強度に影響し，エンジン側からは形状，構造に対する変更要求が出る。構造強度を確保するために形状・材料等の要求が出され，これは性能に影響してくる。このようなプロセスを収斂させる上で，評価を高速で行なうことが必要である。

新しい製品を目指して設計が進められる時には，既存のデータや知識のみで評価を行なうことができず，実験や理論解析を行なって新しい知見を得ることがしばしば必要になる。CAD／ＣＡＭのシステムはこのような知識獲得のプロセスを援助し，これと設計プロセスとを容易に結び付け得ることが望ましい。これが第３の条件である。

評価を行なった結果，モデルが修正される。モデルの修正は自動的になされる場合もあるが基本的には人間が行なう。このためにモデルの表示と，操作というマン・マシン・インタフェースは十分強力でなくてはならない。人間の操作には言語による指示，図形表現による指示,あらかじめ定義された操作群のメニュー方式による指示など多様な対象に最も適した方式が準備されねばならない。これは第４の条件である。

5.3.3 モデル表現

モデルはそれを通して評価がなされ，またそれに人間が変換を施こすことによって，要求仕様を満たす方向に変更される。この意味で設計プロセスの中心であり，その表現は評価・変更が容易であることが必要である。このモデル表現に課せられる条件は次のようなものである。

(1) 記述力が大であること。

設計に際しては設計者の創意が十分に発揮されることが重要であり，モデルは設計の過程で

生じる可能性をすべて表現できるだけの記述力を持たねばならない。

計算機による表現の方式は手続きを含む形式によるか純粋に非手続き的方法によるかに大別される。現状では手続きを含むデータすなわち抽象データ型が記述力の点で優れていると考えられている。しかしこれまでの技術の範囲では手続きの定義が非常に明確でありこれに関する研究が広範囲に行なわれ，詳細に分析されてきたのにたいし，非手続き表現はその多様性から容易に把握しにくく，非手続き情報の構造やその表現形式とセマンティクスの関係，ひいてはその記述力に関する研究が極めて不十分な域に留まっているということは銘記する必要がある。最近，データベースの研究が進み，普及したこともあって，モデルにデータベースを利用するという試みもなされているが，データベースの構造自体は極めて拘束の多いものであり，非手続き的表現法の可能性のごく一部であることも認識する必要がある。この意味で非手続き的表現の可能性について今後十分な検討が必要であろう。このことは次の表現の柔軟性の問題と深く関わってくる。

(2) 動的な変化に対応し得る柔軟性

モデルは設計の全過程において絶えず評価され，修正される。このような動的なプロセスに耐えるために，モデル表現は十分に柔軟性に富むものでなくてはならない。モデル表現は一般には変数を含む構造情報であり，この変更は単に変数の値の変更のみでなく，構造の変更を含む。この意味で，モデル表現に要求されるのは構造を値とする変数を含む表現と言うことができる。(1)における記述力とはこのような意味である。

モデルは評価と修正のための操作という異なった二種類の目的にたいする処理対象となる。ＣＡＤシステムをより汎用化するためにはモデル表現と処理手順を分離し，それぞれ独立な定義がなされることが望ましい。またこの両目的のための処理手順も独立であることが望ましい。

モデルに対する評価・修正のプロセスについてみるとモデルにいかなる操作を施こすかは，各時点でのモデルをその時点に到るまでの履歴によって定まり，また処理に際して，既成の標準的な構造情報とモデルの構造との部分的なマッチングを取っていかなる処理を施こすかを定めるといった手順が必要とされる。この点に留意して，(1)で述べた２種類の表現形成－手続きを含む表現と手続きを含まない表現を比較してみよう。

前者については，その最大の欠点は構造のマッチングのとりにくさにある。これはモデルの動的な処理を拘束する。他方，構造のマッチングを避け，モデルに依存する形式で処理手順を定義すると汎用性を損なうことになる。したがってこのような表現形式は設計対象を特定の範囲に絞った専用システム向きのものとなる。

一方，後者に関しては前述の如く表現法は極めて多様である。この一例としてデータベースを用いることを考えてみよう。データベース自体はデータ独立の概念が前提にあり，汎用性の条件は満たしているが，現存のデータベースの構造は極めて堅く，この構造を動的に変えると

いうことは困難である，事実上マッチングを取ることも難かしい。この意味でやはりＣＡＤ／ＣＡＭにおけるモデル表現としては不向きである。

以上のことから，現在用いられている方法はいずれもＣＡＤ／ＣＡＭにおけるモデル表現としては不十分であり，新しい方法の開発が必要とされる。これは恐らく非手続き的方法によることになろう。

(3) モデルの定義のしやすさ

モデルの初期設定および修正は人間の指示に従う。したがってモデルは人間とのインタラクションが容易に行なえる表現形式であることが望ましい。モデルに関する人間の発想は多くの場合，言語，図面，記号，数式等人間の日常用いる表現形式により行なわれる。これらは非手続き的表現であり，この意味でもモデル表現は非手続き的なものであることが望ましい。

(4) マシン制御情報の生成

設計の終了時にモデル情報はマシンを制御するための制御情報に変換されねばならない。この制御情報は手続き情報である。したがってモデル表現はこの変換に適したものでなくてはならない。

以上を総合して，モデル表現に要求されることは

(1) 人間の用いるモデル表現形式からマシンへの変換が容易である。

(2) 記述力が豊富である。

(3) 動的な構造変化に適する。

(4) マシン制御情報の生成が可能である。

ことである。

5.3.4 評価・修正操作の手順

評価やモデル修正操作に関してもこれらが満たすべき条件が明確にされねばならない。これらはいずれも原則として手続きである。設計プロセスにおいては状況に応じて多面的な評価や修正がなされる。手続きで表わされる評価や修正の手順は，そこに含まれるパラメータによる変換はあるが，既してその視点は固定される。したがって評価・修正の多面性に対応する多数の評価・修正手続きが準備される。このどれが適用されるかはその時点のモデルに依存するから，これら手続き群を管理し，動的に選択する管理システムが必要になる。個々の評価手続きは一般に複雑で，これらは前もってプログラムしておくことが必要であるが，管理システムは目的に応じ，これら手順群の適用条件をユーザが任意に記述できる形式とすることが望ましい。このためのマン・マシン・インタラクションは容易でなければならない。一方，適用条件はモデル表現に依存するという性格上，モデル表現と類似のものであることが望ましい。この意味で(1)適用条件の記述は非手続き的に行なえること，それは人間の用いる表現からマシンへの変換

が容易なものであることが要求される。

他方，このような記述に基づいて，適宜，評価・修正手続きを選択し，実行させるための高レベルの汎用手続きが存在せねばならない。これは適用条件とモデルとのマッチングをとり，その条件に対応する手順を呼び出す。

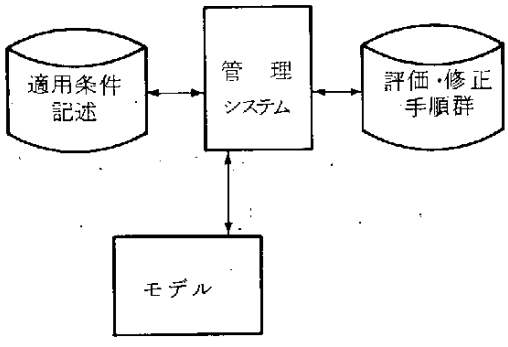


図５－６　知的ＣＡＤ／ＣＡＭシステムの概念構成図

5.3.5　ＣＡＤ／ＣＡＭシステムの使われ方

上述の条件を満たすＣＡＤ／ＣＡＭシステムが実現した場合の利用法は次の通りである。

(1)　評価・修正プログラム群の作成－通常のプログラミング言語により記述，この部分はプログラムの可搬性を良くし，プログラムの流通を図ることにより，その都度プログラミングする手間を出来る限り省く。

(2)　適用条件の記述も自然言語に近い言語，図面等による記述。

(3)　モデルの記述（(2)と同様）

(4)　管理システムをランさせるシステムは評価結果を表示。

(5)　ユーザはその結果をみて，修正指示を(3)と同様の手段で行ないランーシステムは修正されたモデルを表示する。

(6)　ユーザはその結果ＯＫなら(4)へ，さもなくば再度(5)。

(7)　設計条件を満たしたら，マシン制御情報生成を指示。

5.4.　知的ＣＡＣ／ＣＡＩシステム

5.4.1　知的ＣＡＣ (Computer Aided Consultant) システム

我々は，社会生活を行なう上で必要な全ての分野に精通している訳ではない。問題が発生すれば，個々の問題に応じた専門家に相談して解決するのが普通である。知識情報処理システムの一つの眼目は，計算機に各種の専門分野の知識を知識ベース化して，それに対して自然言語等による質問応答相談を行なうシステムを実現し，計算機を我々の身近な専門家として利用す

ることである。

ＣＡＣの分野としては，法律相談，教育，税務・税理相談，医療相談等各種専門家分野が考えられるが，彼等を代行しうる知的ＣＡＣシステムの使われ方のイメージは自明のことと思われる。そこで以下では一つの典型的なＣＡＣのサンプルとして教育に焦点をあてたＣＡＩ（Computer Assisted Instruction）システムについて，その現状と将来を展望してみる。

5.4.2 Computer Assisted Instruction （CAI）

最近のＣＡＩの研究開発は，コンピュータ技術の発展と強い相関関係をもち，この傾向は今後一層強まることが予想される。海外では，米国，英国，カナダ等でＣＡＩ（英国ではCAL: Computer Assisted Learningとよぶ）の研究開発が，大学，企業，政府等の協力により１９６０年代から活発に続けられてきている。米国では，フロリダ州立大，ダートマス大，スタンフォード大，イリノイ大，カリフォルニア大アーバイン分校，テキサス大，ＭＩＴが中心的役割を果している。ＭＩＴを除く以上の大学では，大学・高校のカリキュラムに実際にＣＡＩを組み入れて，ＣＡＩによる学習効果を定量的に測定する作業が進められている。それによれば，時期尚早であるとの意見もある。学習効果は，ＣＡＩのプログラムを作成する教師の質，科目の種類，ＣＡＩの周辺装置の種類等により差異がある。またＣＡＩを導入しなかったクラスと比べて，ＣＡＩを導入したクラスの方が一般に学習効果は優れている，ということが報告されている。

ＭＩＴでは，Intelligent CAI （ICAI）の研究を進めている。これは後に述べるように次世代のＣＡＩを目指すものである。こうしたアプローチに興味を持つ企業の主なものをあげるとすれば，ＢＢＮとXeroxであろう。

英国では，政府の援助を受けた大規模なＣＡＩプロジェクトがあり，現在研究情況は活発である。主な大学としてエジンバラ大，ロンドン大，chelsca大，University Collegeが，最近では，オックスフォード大や，オープン大が参加している。

こうした海外の情況に対して，我が国のＣＡＩ研究開発は立ち遅れているといわざるを得ない。スッペによれば現在の日本におけるＣＡＩの研究レベルは非常に低い。これは会話型計算が日本で立遅れていることと無縁ではないと思われるが，次の１０年に日本ではＣＡＩの分野で大きな変化が起り，２０年後には米国と並ぶだろうと予想している。

〔第５世代CAI〕

１９７０年代の研究成果は，１９７０年代初頭の予想を下まわるものであったといえる。その原因は，ハードウエア価格，ＣＡＩを組み込むプログラミング言語，教師との協力と，他のＣＡＩサイトとの相互協力の必要性等が考えられるが，これらは今後技術的に解決しうる問題であろう。しかし，ＣＡＩが教師の代用をするものである以上，ＣＡＩシステムそれ自身，膨

大な知識を持たざるを得ない。したがって，第５世代のＣＡＩは，曲型的な知識情報処理システムとして実現されねばならない。換言すれば，知識情報処理システムの格好のモデルとして，第５世代ＣＡＩシステムをとらえることができる。前節のＩＣＡＩは，こうした方向を目指すもので，人工知能の研究との関り合いも深い。

第５世代ＣＡＩシステムで特に重要視される項目を重複をいとわず列挙するとすれば以下のものである。

① 自然言語による会話（応答理由の説明機能をも含む）

② 音声による応答

③ 図形のパターン認識

④ 学生の思考・学習状態のモデル化とMixed Initiative

⑤ ビデオディスク等の新しい周辺機器の導入

⑥ 知識ベースの構築

⑦ デフォールト推論手法の確立

⑧ 柔軟な画像表示装置（電子ノート）

⑨ 高性能低価格パーソナルコンピュータ

これまで，大学，高校等の学生教育を対象としたＣＡＩシステムについて述べてきた。しかし，コンピュータ産業のように日進月歩の企業内での技術研修や，語学研修低開発国に対する援助等で，我が国でも本格的な第５世代ＣＡＩシステム実現に取り組む必要があろう。

以上の項目を組み入れた第５世代ＣＡＩシステムが実現したとしても，特に学校教育に対しては，教育そのものの観点から批判がなされるであろう。その主なものは，教師との人間的ふれ合いのない教育の是非，画一的教育の危険性，権力によるコントロール等々であろう。これらの問題は，他分科会，特にＳ分科会で議論しておくべき問題であると思われる。

第５世代ＣＡＩシステムの実現には，教育者，心理学者等との学際的な研究協力の場が特に必要になろう。

6. マシンのイメージ

6. マシンのイメージ

(1) アーキテクチャは

知識情報処理プロセッサは，機能のどの面を強調するかによって色々なマシン・イメージを与える。処理機能から見れば推論マシン，管理機能から見れば知識ベース・マシン，対話機能から見れば自然言語・音声マシン（図形・画像マシン）というイメージを与える。具備すべきアーキテクチャの要件を列挙すると図６－１の様になる。これらの諸要件を統合した結果，どのようなマシンに結実するかは，もう少し研究の進展を待たねばならない。

(2) 何　故

何故，プログラム言語（問題解決・推論システム），データベース（知識ベース管理システム），マン・マシン・インタフェース(知的インタフェース・システム）の進化が，新しいマシン（アーキテクチャ）を求めるかである。進化にともなって，既存マシンの枠内では効果的な処理が不可能になり，新しいマシンが求められるというのがその筋書きである。図６－２に従い説明する。

まず処理機能，プログラム言語に関してである。繰返し計算と静的データ構造はFORTRANという記述系にまとめられ，既存マシン上で実に効率良く実行される。次の再帰呼び出しと動的データ構造はLispによって乗り越えられた。ALGOLやPL／Ｉは系としてはその記述要素を持ちながら，実用的といえるまでの再帰呼び出しやリスト処理を実現することができなかった。Lispの既存マシンへのハード化の要求はスタック位で，最近のLispマシンも，その基本的な計算の機構までも変更しようというものではない。次の非決定的処理とパターン照合はPROLOGによって乗り越えられた。μ-PLANNERをはじめとする人工知能向言語は，これに失敗した。PROLOG自身は，まだ誕生まもなくでその成長は，今後の研究成果に負うところである。

LispもPROLOGも，その障壁を乗り越えるにあたっては，その身を清め，基本スタイルを一新するという努力をした。その努力のかいがあったからこそ，既存マシン上でも効率良く働くことができた。それでは，まだ既存マシンでは達成されていない非決定処理のほとんどや並列処理は，また新しい基本スタイルを持つ言語を考案すれば，既存マシン上で乗り越えられるのであろうか。明らかに，もはや不可能である。今度はマシンの方がその基本スタイルを変えねばならない。その新しいスタイルが駆動型（データ・フロー）アーキテクチャである。

次に管理機能，データベースに関してである。プログラムとその処理対象となるデータ群とのインタフェースを整え，互いに独立して扱えるようにしたいというのが，データベースの発想の源である。まず，ファイル・システムの完備により，ファイル記憶の物理的性質から解放され，さらにその論理的性質からの解放を目差して，データベースの研究・開発が本格化する。

MARKⅣ，ADABAS等の商品化が行なわれ，CODASYLによって，共通言語としてまとめられ，COBOLのデータベース機能が完成を見る。さらに，世の中の様々なデータの意味を出来るだけ自然に表現したいという望みは，データモデルの研究となり，コッドの関係モデルによって，一つの目標に達した。このモデルによる，関係（言語）データベースの実現例の一つがＱＢＥである。関係データベースは，高機能であるだけに，規模が大きくなると，既存マシン上にプログラムで組み上げるという方法では，すぐ限界となり，関係データベースマシンという新アーキテクチャを模索する動きとなる。

データベースには，もう一つの流れがある。人工知能における知識表現の研究としての流れである。知識は，単なるデータだけではなく，規則や定理や手続きなどデータをどう見るかという高次のデータをも含むものである。データとプログラムの一段高い次元での融合化が図られる。関係モデルをさらに一段と高度化したものと見ることができる。したがって，新しいマシン，知識ベース・マシンの発想が必須である。

最後に，対話機能，マン・マシン・インタフェースに関してである。マシンと人間との共同作業系として情報処理システムをとらえる。単純作業，ただし高速・多量データ処理はマシンが，高度な判断は人間が行う，両者は，柔軟な対話機能によって結ばれる。その境界は，情報処理技術の進歩にともない，少しづつ人間の側に移され，より密接なものとなる。これが，本質的なシステムの姿である。マン・マシン・インタフェースの重要性はますます高くなる。文字ディスプレイ・タイプライタによる対話は，時分割システムの普及（日本では，全く不十分）によって一般化し，定着した。ただし，対話の形態は単純な交信にとどまる。次に，日本語端末，音声入力出装置，図形・画像入出力装置が，端末のインテリジェント化，さらにコンピュータのパーソナル化によって実用化され，広く普及しつつあり，交信の形態も多様化しつつある。このような高機能な入出力装置の実用化，さらに低廉化には，種々の専用ハードウエア，プリ・プロセッサが開発された。この段階では，対話機能は，あくまでも入出力装置，入出力システムの問題である。次に，理解機能をともなった知的会話の形態に入る。もはや単なる入出力の問題ではなく，マシン・アーキテクチャ全体にかかわるものとなる。

さらに重要なことは，上記の三つの基本機能の関連の仕方が，進化にともなってますます密になってくるということである。そのことが，第五世代コンピュータという一つのマシンを想定する裏付けともなっている。

(3) 推論実行速度について

新世代コンピュータ（＝推論マシン）の性能表現の一助として「推論実行速度」の設定を提唱する。

この単位は，平易に言えば，「三段論法」による推論を１回行なう操作である。

より具体的に言えば，例えばレゾリューション方式において，ユニフィケーションを含み，

新しいレゾルベントを1つ作る操作である。手続き言語的に言えば，パターンマッチングを行って手続きを呼び出す操作に相当する。

この操作は，現行のコンピュータにおいてインタープリタ的に行うとすれば，100～1000ステップを要するものである。

略号として，

LIPS（Logical Inference Per Second）

を採用するとすれば，

LIPS＝100～1000 IPS

いま，1 Mega LIPSのマシンを想定する。このマシンを現行マシンでインタープリタ的に実現するとすれば，100～1000 MIPSマシンが必要である。

コンパイラを採用すると（コンパイル時間等を無視するとして）10～100 MIPSマシンが必要である。

さし当り，新ノイマン的アーキテクチャを採用するとして（直列マシン），専用アーキテクチャを採用すれば，そのマシンの複雑さは，現行の1～10 MIPSマシンに相当しよう。

1～10MIPSマシンは10年後にはパーソナル化されると予想されているから，1 Mega LIPSの推論マシンは，10年後にはパーソナル化可能な線であろう。

(4) 処理速度は

別の角度からイメージを具体化するため，処理速度（機能）関する大まかな目標を上げる。

① 100M～1G LIPSの処理能力を持つ。

当然，並列計算を前提とし，レゾリューションを平均1 nsec～10 nsecで実行することになる。これは，あまりに基本すぎるので，もう一段マクロな目標を上げる。

② 述語論理の証明（ユニット・レゾリューション）で数10クローズ（中間で生成されるクローズが数千）の証明を数10μsec以内に行なう。

現在の大型機では，数100 msecを要するものである。もう一段マクロな目標として

③ 1万の規則（プロダクション・ルールあるいはクローズに換算して）を用いて高度な推論を数sec以内に行なう。

現有のシステムでは，たかだか数100に対し，数sec～数10 sec かけある程度の推論を行なっているのが現状である。

(5) 如何にして，

目標マシンのイメージとして，高機能な革新的アーキテクチャを設定した。当然のことながら，これは多くの研究要素を含むもので，すぐに設計にとりかかれるようなものではない。まず，第一ステップとして，PROLOGマシン等をノイマン型アーキテクチャとして設計・試作し，評価機構の内部メカニズム，パターン照合，記号処理等，知識情報処理の基本技術を体得

し，蓄積することが急がれる。

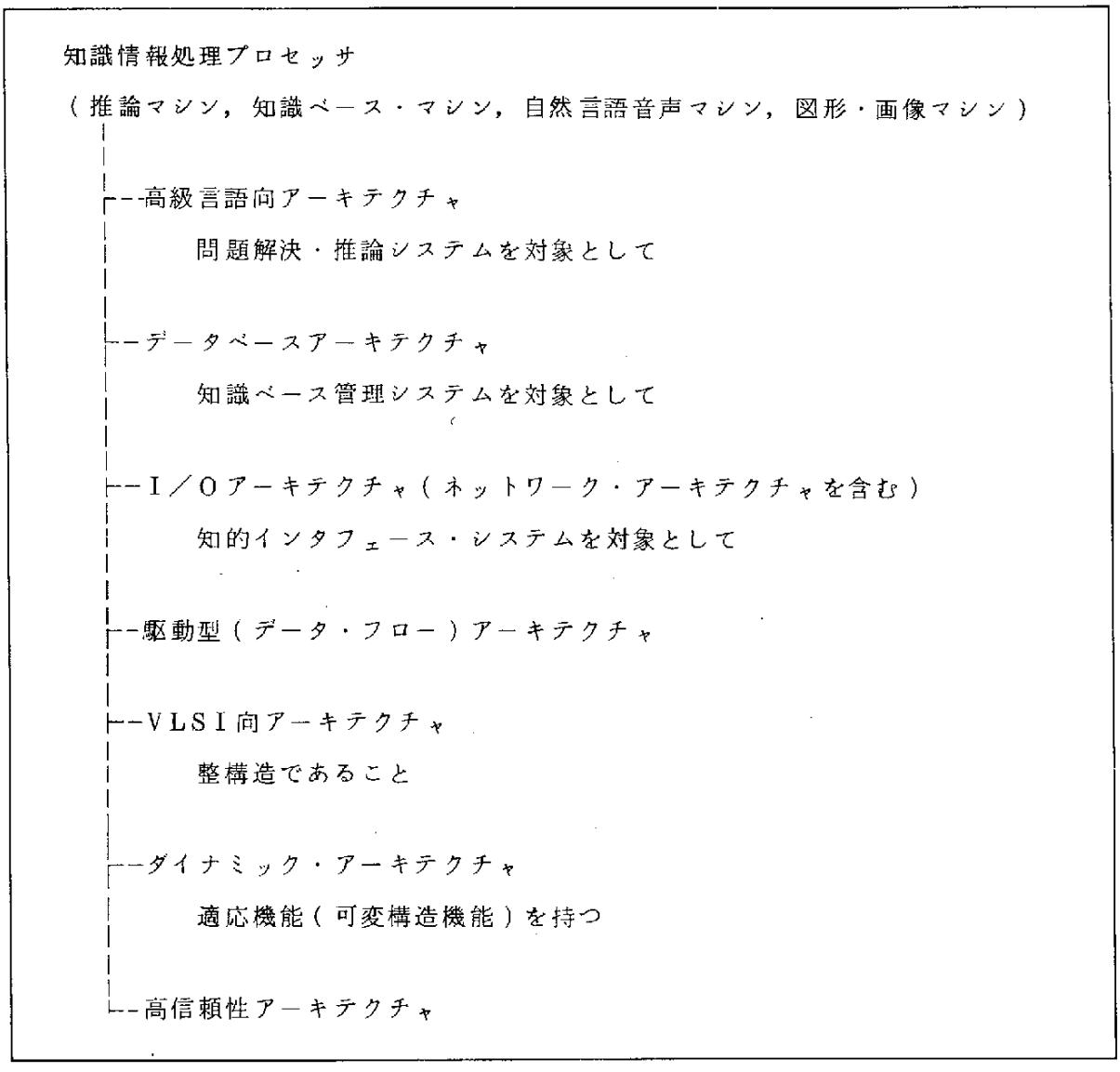


図６－１　知識情報処理プロセッサの要件

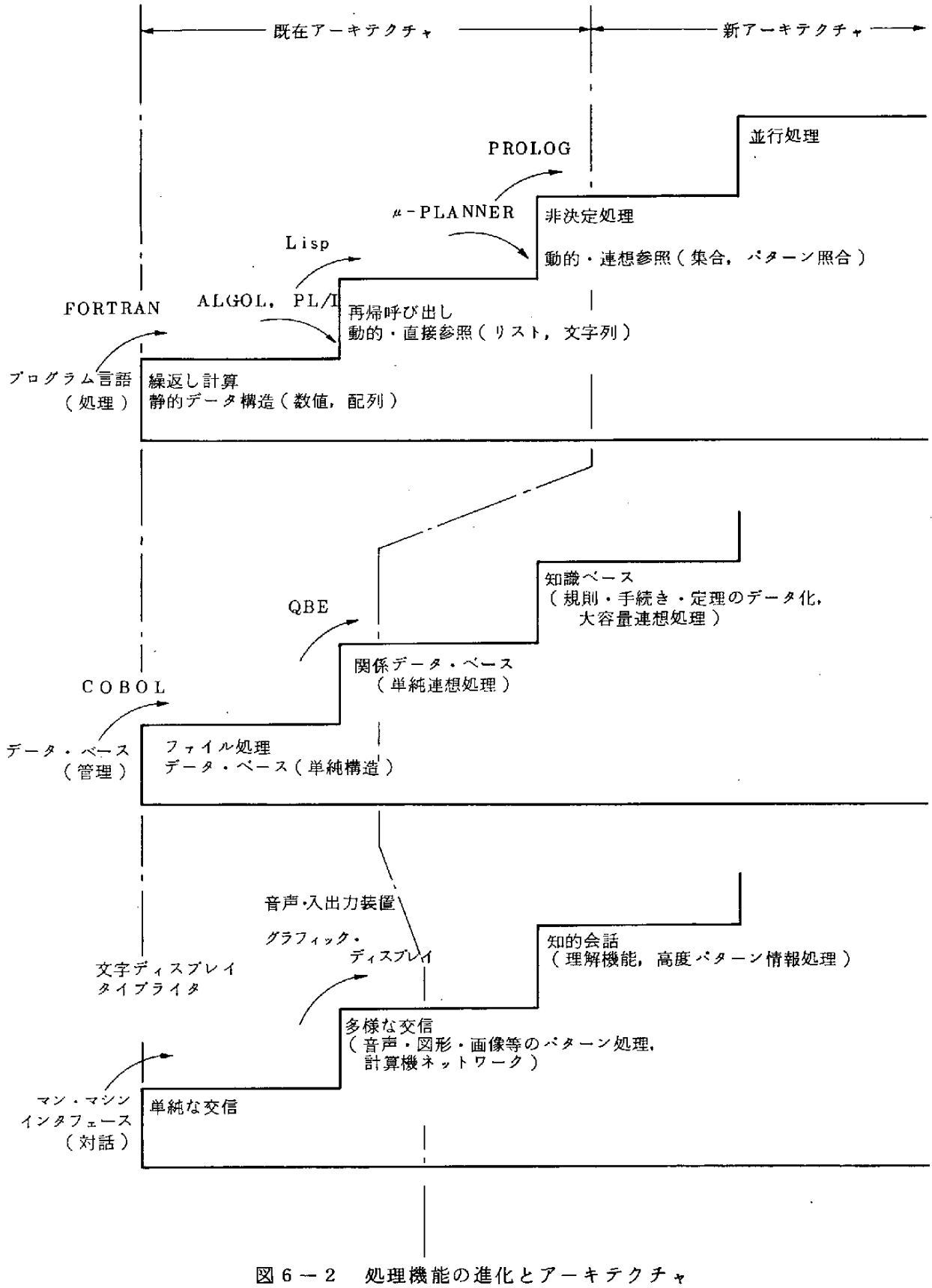


図６－２　処理機能の進化とアーキテクチャ

付　録

1. 各論エッセイ

〔1〕 合金設計システム

〔2〕 混合計算と変換機械

2. 各WG報告

〔1〕 並行及び並列システムの形式的記述

〔2〕 PROLOGの拡張に関する若干の考察

〔3〕 並列定理証明機の一構成法

〔4〕 代数的仕様に基づく段階的詳細化
－HISPを例として－

〔5〕 高速リスト処理アルゴリズム

〔6〕 記号処理データフローマシン

〔7〕 並行処理の記述と管理

1. 各論エッセイ

(1) 合金設計システム

1. はじめに

合金設計システムは，知識ベースシステムの，実例の一つである。これと計算機との関わりを示す前に，この例そのものの意味・背景について簡単に述べる。

この例は材料設計と呼ばれている分野の中で，合金設計に関する問題である。従来，多種類の合金が開発されてきたが，これまでの合金開発の方法は多分に経験的であり，職人的であった。すなわち，目的とする性質をもった合金を体系的に作り出す方法が存在せず各技術者がそれぞれ断片的な物性的知識と経験とから，なにがしかの方向を定め，それ以後は試行錯誤により，多分に偶然に期待する方法で行なう他なかった。近年，物性論等の知識の蓄積，電子顕微鏡による結晶構造の分析などの手法が進み，これらを背景に合金の開発を体系的に行なおうとする動きが出，合金設計という分野が形成されている。このためには合金の開発に関する知識やノウハウを集積し，目的に応じてそれを選択し，利用する技術の確立が要求される。従来はこれに近いことを各個人の頭の中で行なっていたが各個人単位では習得知識が限定的かつ偏在すること，知識利用の一様性が保証されないなどによる限界があり，知識やノウハウを顕在化し，利用技術を確立することが必要になってきている。

以上合金設計という実例に則して述べたが，全く類似の事情が機械設計，医療診断，経営意思決定など広範囲の分野に存在する。そして重要なことは，これら異なった分野において，用いられる知識の内容そのものは異なるが，それを集積し，目的に応じて選択・利用するという情況は全く同じであることである。このことは知識表現の形式を確立し，知識そのものと，それを処理する処理系とを分離することにより，処理系は汎用のものになり得ることを意味する。このような処理系および知識表現が満たすべき条件は後に述べるが，このような処理系が汎用の知識ベース・システムに相当する。

2. 知識ベース・システムの特徴

本論にもどり，この実例は実際には必要な諸規則の集合の僅かの部分に過ぎない。しかも，その最も単純な部分である。実際には数百あるいはそれ以上の規則やノウハウから成る知識体系が作られるが，要求が出された時，その要求を満たすために，どの知識が利用されるかは要求内容に応じて定まり，前もって定めることのできないものである。また，各知識の多くは一般性のあるもので，多数の対象に適用できる形であり，表現として変量を含む。要求に応じて，この変量への代入が行なわれ，要求を満たすための最も適切な形に変換せねばな

表１　合金設計における知識の例

(注)　以下の記述には〝ベースの金属（base material ） Mに原料（alloying element）Aiを添加して溶解する際〟が含められる。

1* 合金添加元素の薄める割合が少ない程，偏析も少ない。

参考：溶解過程において外部より振動を加えると偏折が減少する。

：凝固過程において急冷する程偏折は少ない。

2 合金において重量%の小さな方が合金添加元素である。

⋮

7 金属間化合部は割れやすい。

8 共晶合金は割れやすい。

⋮

10 共晶合金の融点は各元素の融点より低い。

⋮

12 融点の低いものは溶かしやすい。

⋮

14 A，B二元合金においてA／Bが一定であれば元素間の結合が強い。

⋮

18 溶かしやすい合金は母合金として適する。

⋮

21* Mの蒸気圧をP_1，Aの蒸気圧をP_2i，MとNの化合物の蒸気圧をP_3，MとOの化合物の蒸気圧をP_4，AiとNと化合物の蒸気圧をP_5i，AiとOの化合物の蒸気圧をP_6iとすると

$\{\{(\forall i)\ P_1 \geqslant P_2 i\} \cup (P_1/P_2 \leq 10^{-2})\}$ でかつ

$P_1 \ll P_3, P_4, P_5i, P_6i \quad P_2 \ll P_3, P_4, P_4, P_5i, P_6i$

なら普通の条件（条件C）で溶解が可能である。

⋮

23 ファイル2は炉のタイプ#A，るつぼのタイプ#Bで，ベース金属Mに材料Aiを添加して入熱量Xで時刻T_1からT_2まで，#nameの人が，溶解した時，$T = T_2 - T_1$で得られる合金Maについて，T_1におけるAiの重量%n_1であり，T_2における重量%n_2であり．さらにM＋AiのT_1における重量とT_2における重量の比がλであることを表している。

A	B	M	Ai	X	T	n_1	n_2	name	λ

ファイル２

24* 蒸気圧Pは次式で与えられる

$\log P\,(\mathrm{mmHg}) = A/T + B \log T + CT + D$

ここではTは温度A，B，C，Dは材料Mat.とそのコードSと共にファイル３に記されている。

Mat	S	A	B	C	D

ファイル３

らない。また知識利用の順序も要求内容によって定まる。

このように知識利用の際の一つの特徴は

(1) 知識利用計画は要求内容に応じて動的に作成されることである。

次にこの例の中にいくつかのものはきわめて曖昧な表現になっている。これは人間のもつ知識そのものの曖昧性による。多くの場合，人間のもつ知識は不完全なものである。人間は新しい情報を収集することにより，不完全な知識をより完全化するという努力を意識的・無意識的に行なっている。この意味で知識というものは一般的には不完全なものであるが，それにもかかわらず，その不完全知識がなにがしかの情報を保有していることは明らかであり，不完全情報を組み合せることにより，それらから目的にたいし有用な情報にすることができる。前記の如く各種応用においてはこのような情報の利用は不可欠である。このことから知識ベース・システムの第二の特徴として

(2) 知識は一般には曖昧な（あるいは不完全な）ものである，ことが指摘される。このことは知識表現として，①不完全情報の表現が許されることのみでなく，②知識の完全化のプロセスを実現できることが要求され，処理系としては，③不完全知識を含め，知識に含まれる情報を損なうことのない利用方式を確立すること，④曖昧さの評価を行ない得ることといった条件が課されることとなる。

次に表1のいくつかの例が示しているように，知識がデータベースと関連して定義されることがある。このことは前記の(2)の特徴とも関連するが，知識は常に情報収集を通して完全化され，あるいは生成されるものであることを示す。知識が多くの場合，体験（実験等を含め）を通してその存在が示唆され，分析により抽象化され，一般的なものになるというプロセスを経るものであり，この意味で入力情報の意味的定義を与えることが知識システムには必要である。この例では入力情報あるいは生データがデータベースの形で与えられるということを前提に知識が表現されているが，この機能は一層拡大されねばならないであろう。すなわち知識ベース・システムの第三の特徴として

(3) 入力を含め未処理情報との相互変換の可能性

があげられ，①データベースおよび一般的なファイルの記述が同一表現形式でなされると同時に，そのアクセスが同一処理のもとで行なわれること，②前記不完全知識と関連し，生データのレベルからより抽象化された，一般的な知識へと帰納的に変換し，かつ不完全知識をより完全化する機能を必要とすることも指摘されねばならない。

知識ベース・システムにはこれ以外にも多くの特徴があることはいうまでもないが，より詳細な議論は後に回す。問題はこのような機能を従来の計算機技術のもとで実現することが，上記の諸条件から困難なことである。個々の知識をプログラムにより表現する試みによっても，データベース化する試みによっても，このように動的であり，不定形であり

曖昧でありかつ学習的な進化をも許す表現と処理系を実現することは困難である。反面，このような条件を満たすものとして知識型システムが実現すれば，前述のような先端的，専門的分野だけでなく，逆に日常的な問題をも扱う使い易いシステムが実現することになる。

知識型システムの発想はこのように，従来の技術による限界がはっきりしたこと，いいかえれば，従来は既成計算機の技術の範囲内で応用分野が定められていたが，それを超えた新しいニーズがすでに発生していることによる。このようなニーズを喚起した緒はハードウエア技術の進歩と低価格化に伴い，情報処理用のファシリティが数年前に比して飛躍的に増大しており，今後この傾向が続くと予想されるからである。

3. 知識システム実現の可能性

新しい技術に関し，それが実現可能か否かという問題に関して，少くとも三つの段階で考える必要がある。すなわち

1. 理論的可能性
2. 技術的可能性
3. 経済的・社会的可能性

知識システムは基本的にはソフトウエアの問題であり，その背景に深い理論的洞察が必要である。これに関しては未だ十分な理論の確立にまで到っていないが，これまでの経験と共に基礎的な概念が固まりつつあり，近い将来，この理論的根拠が定まることが予想される。この意味で1.の理論的可能性は充たされる。

知識システムの理論を実現する技術に関しては，従来の技術と大巾に異なることが予想されるが，1.の理論的検討に際して，知識システムにとって必要な基本的な，あるいは論理的な機能を明確にしてゆけば，これを実現する基礎技術はすでに十分に高いレベルにある。一方物理的な性能に関しては未だ必ずしも十分な検討がなされていない。知識システムの基本動作が，本来，探索ー処理の組合せであることから，従来の計算機の考え方のもとでは非常に非能率的である。したがって知識ベース・システムの実現に際しては知識という，いわば世界のモデルを表現する複雑な構造を持った情報を扱う一方，処理系の構造は極力単純化する努力が必要となる。これは処理対象の構造と処理系の構造の完全な分離を要求する。従来のプログラム方式では，ともすると対象の構造が直接的に処理系の構造に反映されたり，そうでなくても相互の強い関連が認められるが，知識表現と処理系を分離することによってこの構造の依存関係を切りはなし，処理系の構造自体は極めて単純化してチップ化することが不可欠と思われる。これは再び知識システム構築に関する理論研究に反映されるが，これを実現する可能性は十分高い，これが可能であることは，たとえば数学という十分に形式化さ

れた分野において，記述される対象の世界の構造を前もって確立しておくことにより，記述系ならびに処理系自身はたかだか有限階の高階論理の範囲で実現し得るといったことに範を求めることができる。

技術的問題に関してもう一つ重要なことは，たとえ処理系自身はチップ化できたとしても，探索に際し，Ｉ／Ｏが頻繁に要求される状況であってはならないことである。このためには一定量の主メモリ容量が必要である。この容量は，たとえば知識システムを特定の分野に適用した場合，そこで用いられる用語を仮りに2,000とし各語が表わす概念の直接的関係を表現するために，各々５０Byte を必要とすると，このために１００KB，処理系および各種サポート用プログラム領域として２００KB，作業場所として２００KB，その他の入出力処理部分や表の類の記憶領域を含め約７～８００KByteのメモリがあればほぼ十分である。知識ベース自身は外部メモリに置き，必要に応じてその一部を主メモリに移して用いれば十分である。補助メモリとしては知識ベースの他，データベースあるいはその為のバッファ用として数十MByteが適切であろう。現在ＶＬＳＩを用いた新しいマイクロコンピュータでこれらの条件で満たすものがすでに発売されており，この点で問題はなく，この意味で技術的実現可能性は十分大である。

第三に経済的および社会的可能性はどうであろう。経済面ではある技術が成立するか否かは費用と効果との関係であるが，前述の如き応用にたいし効果は十分大であると考えられ，一方，ハードウエア，特にメモリ価格が急速に低下していること，この傾向が今後少くとも１０年間は続くと予想されることから，前記の規模のハードウエアをパーソナル計算機として所有する時代になる可能性は十分高い。この意味で経済的な可能性は十分大である。

他方，社会的可能性としては，まず従来の蓄積された計算機技術の成果すなわちプログラムとデータとが利用できなくなる状況が生じた場合，技術の革新に対する抵抗が非常に大なることが予想されるので，知識ベース・システムの実現に際してこれは留意する必要がある。単に情報のみでなく，ファシリティそのものも，少くとも残存耐用年限の間は共用できる形で知識システムを実現することも考慮せねばならないかもしれない。しかしこのことは知識ベース・システムの概念と反するものではない。知識ベース・システムにおいても処理の最終段階において，現在の計算機によるプログラムの実行に相当する処理が必要であり，この部分を新しいアーキテクチャ（たとえばデータフロー・マシン）で行なうか否かは，前段階の，知識処理部分とは論理的に独立な部分である。したがって手順としては従来の計算機のフロント・エンド・サブシステムとして知識システムを導入し，後にこの計算機処理部分を新しいアーキテクチャでおきかえるといったプロセスが必要であろう。

社会的な問題としてはこの他に教育の問題その他があるが，ここではこれ以上は立ち入らない。

4. 知識ベース・システムの要求

知識ベース・システム設計に際して，その設計指針とすべき基本方針を確立することが重要である。

〔知識と処理系の分離〕

知識ベース・システムはまず第一にその汎用性を確立するために，扱われる知識とその処理系とは明確に分離されねばならない。これを大前提とする。知識と処理系の分離はさらに知識を随時付加・削除・更新する上でも必要であり，また対象の構造と処理系の構造とを分離し，処理系を単純化し，ハード化することによって高速化し，必要な速度を得る上でも重要なことである。知識表現が満たすべき条件は後に記すとして，上記基本枠組の中で知識ベース・システムの機能を分析する。

〔1〕 入　　力

知識ベース・システムのもたらす効果の一つは，人間と機械とのインターフェースが非常に容易になり，使い易いシステムの実現が可能となることである。これは知識やノウハウを人間の通常の表現法に近いものとして受け取り，これを処理する能力があることによる。

知識ベース・システムの利用に際しては人間は自己の知識を計算機に与え，また要求を発することが基本的方法であり，多くの場合，目的のためのプログラムを書く作業から開放され，しかもきまりきった作業でない，広い範囲の問題の解決にあたることができる。

このために知識ベース・システムに要求される機能は人間の表現にできるだけ近い形で入力を受け入れることである。このような表現としては自然言語，図形，画像，音声などが主たる手段になる。このことは記述される知識が少くとも入力の段階では非手続き的表現であることを意味する。コミュニケーションの立場からは，問題を記述し，その解を求める上で必要な情報の量は表現法にはよらない。このことは知識システムに関し，二つのことを意味する。第一は知識ベース・システムによって問題が解決され得るなら，与えられた問題と，その解を得るに関連して用いられる知識が表わす情報の量は，従来のプログラムにより，問題をその解法の記述によって表現する場合と同等であることである。しかるに，知識ベースシステムでは知識の表現形式を定め，それを常時，システム内に蓄積しておくことにより，問題発生時に入力する情報量は著しく少くて済み，コミュニケーションに要する人間の努力が最小に保たれることである。第二は知識そのものは非手続き的表現であるが，これが手続きと等価なものである以上，この両者の間に変換が可能であるべきことである。現に，後述するごとく，知識ベース・システムはある意味ではプログラム生成システムといっても良い性質をもっている。

以上のことから逆に知識ベース・システムに関し，次の如き条件が課せられる。

(1) 人間の用いる情報表現を受け入れること。

人間の用いる情報表現としては，前述の如く自然言語，図形，画像，音声が主要なものである。厳密には自然言語の表現は音声，手書きもしくは印刷文字，それ以外の（コード化を伴う）機械的文字入力装置を通しての表現とに分けられるから，入力という立場からは，これらと，図形，画像とするのが良い。これらのうち機械的入力文字装置と一部の図形入力装置による入力はコード化された入力であるのにたいし，この他のものは生データの入力であり，これを前者のレベルに合わせるために認識機構を必要とする。現段階ではこのような認識機構の実現は多くの困難を伴う。また認識を十分に行なうためには意味解釈が必要であり，そのために知識利用が必要となる。すなわち粗い認識－コード化－知識利用－精細な認識といったサイクルが必要になる。したがって開発の手順としては前記コード化入力を第一とし，他のものは当面マン・マシン形式で人間がコード化を援助することにより実用化する他ない。

コード化入力によっても，自然言語および図形を完全に処理することはなお困難であるし，仮りに可能であったとしても，ごく例外的に生ずる表現，しかも代替表現が可能なものまでも含むためにシステムを複雑化することは得策とは言えない。したがって，経済性を損なわない範囲で有用なシステムを実現するためには，入力の形式を限定することが必要である。たとえばこのためには利用できる自然言語のサブセットを定義することが非常に重要な仕事である。このようなサブセットは，(1)その中の表現のみを用いて必要なすべての知識が記述できるだけ十分表現力があり，しかも，ユーザが特別な訓練なしにそのようなサブセットの定義を理解しかつシステムを容易に利用できるものでなければならない。このような言語のサブセットは任意に定められるものではなく，それによって記述される知識をシステムが受け入れるものでなければならない。すなわち，これは知識表現および処理系の設計に関わってくる問題である。

表2 Basic Patterns of Sentence

1	S-V-DO		
2	S-V-to-Inf.		
3	S-V-N-to-Inf.		
4	S-V-N-(to be)-Comp.		
5	S-V-N-Inf.	DO	:direct object
6	S-V-N-pres. P.	Inf.	:infinitive
7	S-V-O-Adj.	Comp.	:complement
8	S-V-N	past P.	:past particle
9	S-V-past. P.	Pres. P	:presentparticle
10	S-V-Adj.	N	:noun
11	S-V-that clause	V	:verb
12	S-V-N-that clause	S	:subject
13	S-V-Conj. -to-Inf.	Adj.	:adjective
14	S-V-N-Conj. -to-Inf.	Adv.	:adverb
15	S-V-Conj. -Clause	O	:objective
16	S-V-N-Conj. -Clause	Conj.	:conjective
17	S-V-Gerund	Ind. O.	:indirect obj.
18	S-V-DO-PREP. -PREP. O.	Adv. Adjunct	:adverbial adjunct
19	S-V-Ind. O-DO		
20	S-V-(for)-Comp.		
21	S-V		
22	S-V-Predicative		
23	S-V-Adv. Adjunct		
24	S-V-Prep. O.		
25	S-V-to-Inf.		

このような自然言語のサ

ブセットを定義する一つの方法は複雑な構文を生成する原因となるような接続詞，関係詞等をある程度制限することである。意味を表わす基本構造である単文の種類を制限することは好ましくない。しかし意味記述に必要な単文の基本セットとして，英文の例で表2に示す程度の構文を許せば記述上ほぼ十分であり，かつこれに対応するように知識表現とその処理系を構成することは困難なことではない。

(2) 半永久的な記憶に適した知識表現

コミュニケーションの立場から，知識システムの利点が知識を記憶しておくことにより入力時の人間の負荷を軽減することができるということは，逆に半永久記憶に適するように知識表現を定めることと，急激に不要情報が増大し，有用な知識の利用が阻害されたり処理時間が長くなることを防ぐ機構を要求する。特に後者に関しては利用頻度による知識ベースを管理する方法の導入が必要である。

(3) 入力処理

知識ベース・システムは入力（知識の記述，要求の記述）を受け取り，これを知識表現に変換する。入力形式が前記のように定められるとしたら，この変換アルゴリズムは変換されるべき形式－知識表現－に依存する。知識表現の形式は内部処理の要求から定まるので，変換アルゴリズムはそれに応じて定められねばならない。

〔2〕 処　　理

知識という形で表現された情報は知識ベースを形成しこれが要求された問題の解決に用いられる。ユーザが創造的作業の補助手段としてこのような知識ベース・システムを利用する際には，仮説という形で問題を入力する。与えられた問題にたいし，知識ベース内のある一部分が関与するが，この選択は問題の表現を手掛かりに処理系が行なう。知識ベースはこの選択が効果的に行なわれるように構成されねばならない。この選択は問題との意味的関連に基づいて行なわれることが重要であり，たとえ形式が異なっても意味的な等価性が成り立つ場合，それが必ず用いられるということが保証されねばならない。このために異なった表現間に存在する意味の等価関係がexplicitに表現され，知識ベース内に貯えられていることが必要である。すなわちここで意味の等価関係と呼ぶのはexplicitに表現された知識ベース内の知識に基づいて定義されたものを意味する。たとえ人間がある異なった表現間に意味的関係があると思っても，それが表現されて知識ベースに含まれていないため，意味的等価関係は成立しない。

所与の問題は，知識との意味的等価関係が成り立つことがシステムによって証明されることのみにより解決する場合がある。これは単純な質問応答型の問題の場合であるが，現実の問題は必ずしもこのように単純なもののみでなく，最終的な結果を得るために，一連の演算を行なうことを要求するものが多い。すなわち知識や問題の記述は非手続き的になされるか

ら入力から出力に到る処理段階のどこかでプログラムが生成され，実行されねばならない。この代表的な場合は問題がデータベースへのアクセスを要求する場合である。データベースはその操作手順すなわちプログラムが確立されて始めて利用可能になるから知識からデータベース操作命令列への変換が行なわれる。もちろん一般的にはさらにこれに各種演算が施された結果が問題の解を作る。

このように知識ベース・システムとは本質的にプログラム生成システムであるともいえる。ではどの段階でプログラムへの変換を行なうべきであろうか？すでに述べたようにプログラム化された情報は動的な性質を失なうから，プログラムが生成された後でさらに知識ベースを用いた意味変換を行なうことが困難になる。したがって，プログラム生成は必要な意味的変換が行なわれた後である。

以上のことから，処理系で行なわれる問題処理の手順は

① 知識ベースを用いた入力の意味的変換（推論）

② ①の結果に基づくプログラム生成

③ 生成されたプログラムの実行

であり，これを遂行する機能が処理系に要求される。

以上は既存の知識の体系が知識ベースとして存在する時，それを創造的プロセスに利用するための条件であるが，これと別に知識ベースをより完全なものに整備するプロセスが必要である。したがって

④ 知識の変更，挿入，削除が必要である。

(1) 知識ベースを用いた意味的変換

意味的変換は前述のようにexplicitに表現された情報間の意味的等価関係を用いて入力を変換しつつ解を得るか，もしくはプログラム生成部に与える情報を生成する部分である知識による表現は，原理的には前述のようにプログラムによる表現と等価であるが，処理系にたいしてはこれを実際上に等価なものにすることが要求される。すなわち知識ベース内に必要な情報が存在するにもかかわらず，それが見落とされることがないことを理論的に保証することが必要である。これは意味的変換は論理処理に対応させた時，推論処理に相当する部分であるが，これが論理的に完全であることを要求する。これは知識表現にたいする要求でもある。

(2) プログラム生成

推論処理を繰り返した結果，もはやこれ以上意味的変換を行なう必要がないと判断された時点で，推論処理の結果がプログラムに変換される。推論処理は所要の入力を知識ベース内の知識を用いて等価な別の表現に変換するプロセスの繰返しであるが，この繰返しの回数は入力と知識ベース内容とから定まるもので，前もって計画がたてられるも

のではない。したがってこの変換を行なう処理系を簡単にするためには，変換というプロセスに対し，知識表現が閉じていること，変換前後の情報が同じ表現形式のものであることを要す。すると推論の終了を，どのように判定するかが問題となる。この判定は得られた結果の構造からでなく，それを構成している表現要素による。一般に，どのような表現形式によるにせよ，文章が単文の集まりとして構造化され，その意味が単文の意味を基本に組立てられていることに対応して，知識も単文に対応する基本の意味の表現単位（要素）の複合として表現される。この表現要素のうちあるものはその意味が手続きにより表わされ，したがって対応する手続きを与えることができる。したがって知識の基本要素はこのような手続きに対応するもの，すなわち非手続き的なその表現が，対応する手続きの名前と解せるものと，それ以外のものに分類される。推論はこの後者のような要素が表現内に残っている限り続けられ，すべての要素が手続きに対応するものに到って終了する。このようにして作られた結果はプログラムに対応する。プログラム生成部はこれを使用するプログラム実行機械に適した形式で実際のプログラムに変換する。

(3) プログラムの実行

この部分は従来の計算機と本質的に変わらない。人間がプログラムを与える代りに，前記プログラム生成部で生成されたプログラムが与えられる。

(4) 知識の変更・挿入・削除

知識ベースは人間の知識の一部を移植したものに相当する。人間の知識は変化するからそれに応じて知識ベースも変化し得るものでなくてはならない。この際留意せねばならないのは，次の諸点である。

(i) 知識体系の無矛盾性

(ii) 知識の段階的発展過程の表現

(iii) 人間の知識の開発の援助

(i) 知識ベースは多くの独立な知識の集まりである。一つの新しい知識をそこに加えるということは，問題解決能力の面では単にその知識の分だけ能力が増すのではない。問題解決能力は知識の組合せに依存するから新しい知識と既存のすべての知識との組合せ論的な程度で増大することが期待される。この際用いられる知識体系が矛盾を含むものであってはならない。したがって知識の変更・挿入のプロセスにおいて，無矛盾性のチェックが機械自身により行なわれる必要がある。ただしこのことは知識ベース全体が常に無矛盾であることを意味しない。創造的プロセスは常に未知の領域に入り込む関係で，仮説が作られ，知識ベースにはいくつかの，時には矛盾した仮説が与えられるかも知れない（複数の人間の共同作業では人により思考方法が異なるか

も知れない）。したがって少くとも一時的には矛盾しているかも知れない知識が存在すること自体は許すことが必要である。要は，問題の解決プロセスにおいて，人間の制御のもとで矛盾している知識が同時に選択されないようにする機構である。

(ii) 人間の知識は外からのデータにより漸次的に進歩してゆく。この知識の発達段階をそのまま知識ベースに反映することができなくてはならず，このための知識表現の形式と利用時の推論アルゴリズムが作られなくてはならない。

(iii) さらに重要なことは人間の知識の発展自体を機械が援助し得ることである。2.で述べたように，これは発見のプロセスと密接に関連し，多くを人間の直感にたよる他ない偶然による事象の発見は別として，事象の関連の発見に際しては仮説を作り，それを検定するという方法が用いられる。仮説は既存知識や観測データを用いて検定されるから，機械がこのプロセスを援助する他，時には検定に必要なデータを人に要求し，さらにはそのための実験計画の作成等を行なうことが理想とされる。

〔3〕 出　　力

出力処理は入力および内部処理に比し，従来技術との隔たりは小さいし，実用上，人間工学的な面で様々な考慮が必要であるが，特別に新しい方式は必要としない。自然言語出力生成に関しても，入力時に利用できる構文の範囲を単純なクラスに限定していることから，出力にたいしても特別に複雑な構文を用いなければ，知識表現からの逆変換として特に困難はない。

5. 情報の表現形式間の変換

知識ベース・システムの基本形式，利用方法に関する上述の考察から，知識とその処理系という分離が行なわれる時の知識表現の満たすべき条件が求められる。知識表現は非手続き的な情報の表現であり，知識ベース・システムの中心に位置するものである。これを通して知識表現（これをKと表わす）以外の情報形式すなわち，(1)人間の用いる表現 ── 自然言語(サブセット)，図形（以下Eと表わす），(2)プログラム(P)，(3)データベース(D)との間で変換がなされる他，この表現内で意味処理が行なわれる。このことをまとめて表わすと，知識ベース・システムが最小限可能でなくてはならない表現間の変換は表3のようになる。知識表現としてはこれらの条件を満たすべく設計されねばならない。これらのいくつかの条件についてはすでに

表3　表現間の変換

(1)	E→K	文脈解析　パタン分類
(2)	K→E	出力生成表示
(3)	K→P	プログラム生成
(4)	K→D	データベース記述・アクセス
(5)	D→K	データ一般化
(6)	K→K	演繹的推論
(7)	K→K′	知識の再構成

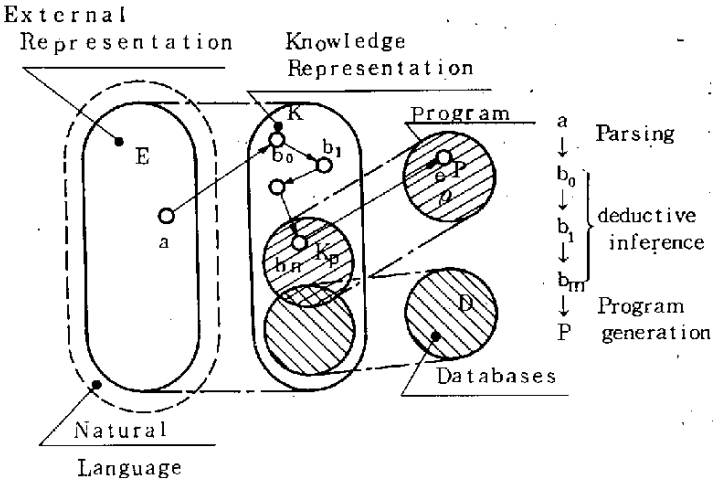


図1　各種表現形式間の関係

述べたので簡単に補足する。

なお図1はこの表の関係を図で表わしたものである。

表3で(1) E→K と(2) K→E はマン・マシン・インターフェースにおける入出力に相当，(3) K→P (4) K→D はそれぞれプログラム生成，データベース・アクセスに相当する。厳密には，通常のデータベースにアクセスするにはそのデータベースを管理するデータベース管理システム（ＤＢＭＳ）ごとに定められているデータ操作言語を用いる必要があるため知識表現からこの操作言語への変換がK→Dに相当する。(6) K→K は意味の等価変換部分（推論）である。以上の5種類は，既成の知識ベースを利用する部分である。

表3の(5)と(7)とは知識の獲得部分である(5)はデータの集積からその中に規則性を抽出し，法則化する部分である。このような帰納的な機能は人間の能力に負う所が多い。しかし，統計処理等，その人間の能力を援助することが重要であり，(5)のD→Kはこの部分を表わす。(7)は人間の知識の増進に対応し，計算機内の知識も同様に変更されねばならないこと，そのために知識表現としてはこのプロセス自身を表現できねばならないことを示している。

6. 知識ベース・システム開発の現状

知識ベース・システムの技術はここ数年で急速に発展してきた。現在米国を中心に多数のシステム開発が進められている。特に医療の分野での知識ベース・システムの利用が積極的で，中でもMYCYNとEXPERTなどと名づけられたシステムが良く知られている。前者は抗生物質の利用に関し，後者は医療診断に関し，医療に対するコンサルティングを目的とするシステムである。医療関係外でも知識ベース・システムを利用する試みは活発化しているが，現在の米国の傾向は実用性を重んじているため問題領域を定め，ある程度その領域に限定された利用を前提としたシステム，あるいはその問題に拘束されたシステムが多いことである。

知識ベース・システムは当然設計目標に依存するからこれら専用化の傾向をもつシステムは，当然，汎用性を重視するシステムとは種々の面で異なってくる。その中で最大の相違は処理系と応用との分離の程度である。現在の多くのシステムではこの両者を完全に分離すること，すなわち前節までに挙げた諸条件を完全に満たすことが困難であるという理由で応用

依存になっている傾向が強い。これが困難な理由のうち最大のものは処理対象である問題や知識の表現を非手続き的に行なう点にある。これは単に表現法のみの問題ではなく，必らずそれに対応する処理アルゴリズム（推論系）が定義されねばならない。このため知識表現は一方では複雑な対象・知識を表現できるだけの記述力を有することが要求されると同時に，他方では汎用の推論アルゴリズムが定義され，それが効率よく働らける程度に単純な構造のものであることが要求される。この相反する要求を解決するためには十分な理論的根拠を必要とする。

このような研究の動機は単にそのような可能性があるということにあるのではない。今日の社会の仕組み，それを支える技術がより複雑化し，人間の創造性がより広汎な分野で必要とされているという現実があり，今や切実なニーズとして存在しているのである。

このような目的をもった計算機システムを実現するために，本稿では創造という機能のうち，人間に委ねるべき部分と，機械化し，それによってより一層効果があがると期待される部分を求め，そのような機械を実現する上で満たされるべき条件を考察した。このような機械として，人間の知識を一定の形式で表現して貯え，人間の要求に応じて，人間のガイドのもとでその知識を利用して問題解決にあたる知識ベース・システムを提案した。このようなシステムは決して仮想のものではなく，すでに実験的には実現の可能性が十分確かめられたものである。現状は未だすべての条件を満たしている訳ではなく，一層の研究・開発が必要であることはいうまでもないが，近い将来，これが実用化することは決して夢ではない。

〔参考文献〕

1) 大須賀節雄：知識の表現と利用 — 知識システムの満たすべき条件，情報処理，Vol. 19, №10 pp. 944－951(1978)

2) 大須賀節雄：プログラムおよびデータの特性指標とその仮想記憶システムの特性への影響 情報処理，Vol. 20，№3，pp. 256－264(1979)

3) 大須賀節雄：意味処理と知識利用のシステムについて，日本語情報処理シンポジウム前刷 (1978)

4) 大須賀節雄：知識構造に基づく機械的推論規則について，情報処理，Vol. 17，№2，pp. 100－109(1976)

5) 大須賀節雄，山内平行：推論能力を備えた情報検索方式について，情報処理，Vol. 18，№8，pp. 789－798(1977)

6) 宇田川佳久，大須賀節雄：知識システムの設計問題への応用について，情報処理学会第20回全国大会予稿集，pp. 695－696(1979)

7) 山内平行，大須賀節雄：知識システムにおけるデータベース・アクセスについて，情報処理学会第２０回全国大会予稿集，pp. 697－698(1979)

8) Ohsuga, S. :Toward Intelligent Interactive Systems, Proc. The IFIP W. G. 5. 2 Workshop Seillac Ⅱ on Methodology of Interaction, North-Holland (1979).

9) Ohsuga, S. :Semantic Information Processing in Man-Machine Systems. Proc. 1977 IEEE Conf. on Decision & Control, pp. 1351-1358(Dec. 1977)。

10) Ohsuga, S. :Theoretical Basis for a Knowledge Representation System, Proc. Int'l Joint Conf. on AI (1979)

東京大学宇宙航空研究所
大須賀　節雄

〔2〕 混合計算と変換機械

—— 将来の計算機設計に向けて ——

本資料は，１９８０年１０月のＩＦＩＰに来日したＩＦＩＰプログラムチェアマンであるエルショフ氏との意見交換会（Ｔ分科会のワーキング・グループ）での講演および討議内容を要約したものである。

要　旨

未来の計算機の方式設計とプログラミングは，情報処理の本質と原言語の諸特性を理論的・数学的に考察することによって決定されるというのが最近の傾向であると思います。混合計算もそのようなものの良い例です。

これまでの"データの上に働らく"機械から，"プログラムの上に働らく"機械というものを考えたいわけです。そのためには，"混合計算"の概念が重要です。混合計算は，いわば"未完結な情報処理"とでも言えるもので，その出力は部分的に処理された情報が中に含まれた"剰余プログラム"です。多くの実用的な情報処理：プログラムの実行，プログラムの翻訳，プログラムの作成，プログラムの最適化，アセンブリ，マクロ処理，編集，ライブラリ，またＯＳにおけるプログラムの生成，プログラム合成などが，混合計算以外の何物でもないと言えます。混合計算には２つの接近法があります。１つは演算的接近法で，それは剰余プログラムの命令を生成するステップを織り込むことにより言語の演算意味を一般化する方法です。他方は，変換的接近法でいくつかの基本変換法則を定めて，それらの適用により剰余プログラムを得る方法です。多様な混合計算を記述する方法を与えてくれること，およびプログラム変換と関係づけてくれるという点で，変換的接近法がより重要であります。計算に対するこのような統一的な接近法が変換機械の概念をもたらしてくれるのです。以上が本論の要旨です。

2つの変数X，Yを持ったプログラムp（X，Y）が関数f（X，Y）を計算するものとします（X，Yの値をx，yで示します）。これは万能計算過程Vを仮定して，V（P，X，Y）と書けます。V（p，x，y）＝p（x，y）～f（x，y）となるわけです。ここで変数Xをしばれば，つまりX＝xのとき，新しい関数fx(Y)が得られますが，これに対応するプログラムPx(Y)を組織的に作りたいのです。言いかえれば，別の万能計算過程Mを導入して

M（p，x，Y）＝Px(Y)

とすることです。Mのことを混合計算と言うわけは，Mが通常の計算の他にプログラムの生成も行うからです。もう一度確認しますと，Mは次のような条件を満している必要があります。

M（p，x，y）＝V（p，x，y）＝p（x，y）
M（p，X，Y）＝P（X，Y）
M（p，x，Y）＝Px(Y)；Px(Y)＝P（x，y）

このような考えが有用なのは入力XとYが，一方のデータは他方に比して異なった値をとる度合が少ない，いいかえれば，一方がパラメタであるという時です。このときは，パラメタの値について一般的なプログラムを適応させることができるのです。この講義の間使用するために例を書いてみます。

$y = x^n$ を評価するプログラム

```
y :=1 ;
while   n > 0   do
  while   even   n   do
    n := n / 2 ;
    x := x²
  od
  n := n − 1 ;
  y := y · x
od
```

もちろん，例えばn＝5のときはこの一般的なプログラムを使う必要はなく，もっと単純に，

n＝5のとき

y：＝x；　x：＝x²；　x：＝x²；　y：＝y・x

を使えば良いわけです。この例のような簡単な場合でなく，汎用プログラムがライブラリにあるような場合には，こういう特別な場合のプログラムを使うことができなくて，汎用プログラ

ムを使わざるを得ず，よって，効率を失うということになります。組織的に特別な場合のプログラムを得ることができれば話は変ります。この特別な場合が汎用プログラムから変換によって得られることはあとで示しましょう。

M（p，x，Y）＝p_x(Y) と書いたとき，xをしばられた変数，Yを自由なあるいは凍らされた変数，またPx(Y) を剰余プログラム，あるいはプログラムpのxの上への射影と呼ぶことにします。次に，翻訳の過程の本質を混合計算が見事に表現していることを，いくつかの式を示すことにより示しましょう。

混合計算の有効性

L＝（D，P，V）を実現言語であり同時に目的言語とします。ここにDは対象領域，Pはプログラムの集合，VはLにおける演算意味，つまりV：P×D→Dです。同様にI＝（Ξ，Π，S）を原言語とします。

ここで実現言語において万能混合計算プログラムmixをすでに持っているという重要な仮定をします。つまり，

自己射影：mix（p，x，Y）＝Px(Y)．

mixを自己射影というのは，それ自身が書かれている言語にプログラムの射影を作成するからです。

次に，原言語の演算意味を解釈する実現言語（＝目的言語）におけるインタプリタを持っていると仮定します。つまり，

int（Π，ξ）＝Π（ξ）

次に，それぞれ定義により

目的コード	$ob_\pi(\xi)=\Pi(\xi)$
コンパイラ	comp（Π，Ξ）＝ob(Ξ)
コンパイラ・コンパイラ	cocomp（S，Π，Ξ）＝comp（Π，Ξ）

を得ます。すると，次の式が成りたつことがわかります。

mix（int，Π，Ξ）＝$int_\Pi(\Xi)$＝ob(Ξ)　（Ⅰ）

mix（mix，int，（Π，Ξ））＝mix_{int}（Π，Ξ）＝comp（Π，Ξ）　（Ⅱ）

mix（mix，mix，（S，Π，Ξ））＝mix_{mix}（S，Π，Ξ）＝cocomp(S,Π,Ξ)（Ⅲ）

mixとintだけにより目的コード，コンパイラ，コンパイラ・コンパイラをも実現できるということは重要です。公正のためにここで次のことは言わねばなりません。

この関係式（Ⅰ），（Ⅱ），（Ⅲ）の一部は日立製作所のフタムラヨシヒコ氏により発見されたも

のですが，（ここで，二村氏はここにいますの返事に講師“オー”）原論文*は広く読まれることなく，少なくとも２度は再発見されています（それぞれTorchinとErshovによって）。

重要な点はこれらの関係が単に事実を言っているに止まらず，目的コードとコンパイラを得る効率の良い方法を示していることにあります。詳しく述べる時間はありませんが，式(Ⅰ)，(Ⅱ)を用いて，目的コード，あるいはコンパイラを得る例をあげます（図１参照）。これは，私にとって，原言語とインタプクタから混合計算によって目的コードを得た事実上の最初の例であります。

入力プログラム

```
read  n ;
i ：＝0 ;
while  i ＜n  do
        i ：＝i＋1 ;
        read  x , y ;
        if  x ＜0  then
                z ：＝x＋( 5×y ) ;
                write  z
        fi        od ;
write  z.
```

目的コード

```
        in(n) ;
        i ：＝0 ;
H1 ：r1 ：＝i ＜ n ;
        if ¬r1  then  goto  M2  fi ;
        r2 ：＝i＋1 ;
        i ：＝r2 ;
        in(x) ;
        in(y) ;
        r3 ：＝x ＜ 0 ;
        if ¬r3  then  goto  MH1  fi ;
        r4 ：＝5×y ;
        r5 ：＝x＋r4 ;
        z ：＝r5 ;
        out(z) ;
MH1 ：goto  MH2 ;
MH2 ：goto  M1 ;
 M2 ：out(z),
```

図　1

*）電子通信学会論文誌，システム・コンピュータ・制御　Vol.2，No.5，１９７１年

2番目の例（図2参照）はインタプリタからコンパイラを得た例です。ここではwhileに対する意味サブルーチンを取り上げました。混合計算がインタプリタのサブルーチンをコンパイラのサブルーチンに変換している例です。

```
proc WHILE=(ref while L) void;
    begin M1:RELATION(heading of L);
        if ¬value of heading of L then goto M2 fi;
        SERIES(body of L); goto M1; M2: end.
```

```
Nee oopa3 B TpaHCJLATope:
proc WHILE=(ref while L) void;
    begin nM1:=copy(M1); nM2:=copy(M2);
        M1:T(nM1:); RELATION(heading of L); n1:=value of
        heading of L; T(if ¬n1 then goto nM2 fi;);
        SERIES(body of L); T(goto nM1;);
        M2: T(nM2:) end.
```

図　2

ここですこし観念的な話をしましょう。プログラムを構文木に解析したとき，端点にデータがあるわけです。混合計算は，その度合いによって，一定のところまでこの木を刈り上げたことに対応します。プログラムの実行をコンパイルと実行の2つに分けて示したものが図1です。混合計算はこの全てのスペクトルをカバーしているといえます。勿論この図は非常に観念的なものですが，どこかに最適な点というのがあるに相異ありません。

休憩にはいる前に，与えられたプログラムから剰余プログラムを得るヒントを差し上げましょう。最初の x^n を計算するプログラムに戻り，n＝5のときの跡をたどってみます。それは図3に示すようになります。

入力プログラム

```
y:=1; while n > 0 do while even n do n:=n/2; x:=x² od;
                                  n:=n-1; y:=y・x od
```

n＝5：x－frozenのときの実行

```
y:=1; n>0⁺; even n⁻; n:=n-1; y:=y・x;    (n=4)
      n>0⁺; even n⁺; n:=n/2; x:=x² ;     (n=2)
            even n⁺; n:=n/2; x:=x² ;     (n=1)
            even n⁻; n:=n-1; y:=y・x;    (n=0)
      n>0⁻.
```

剰余プログラム

```
y:=1・x; x:=x² ; x:=x² ; y:=y・x.
```

図　3

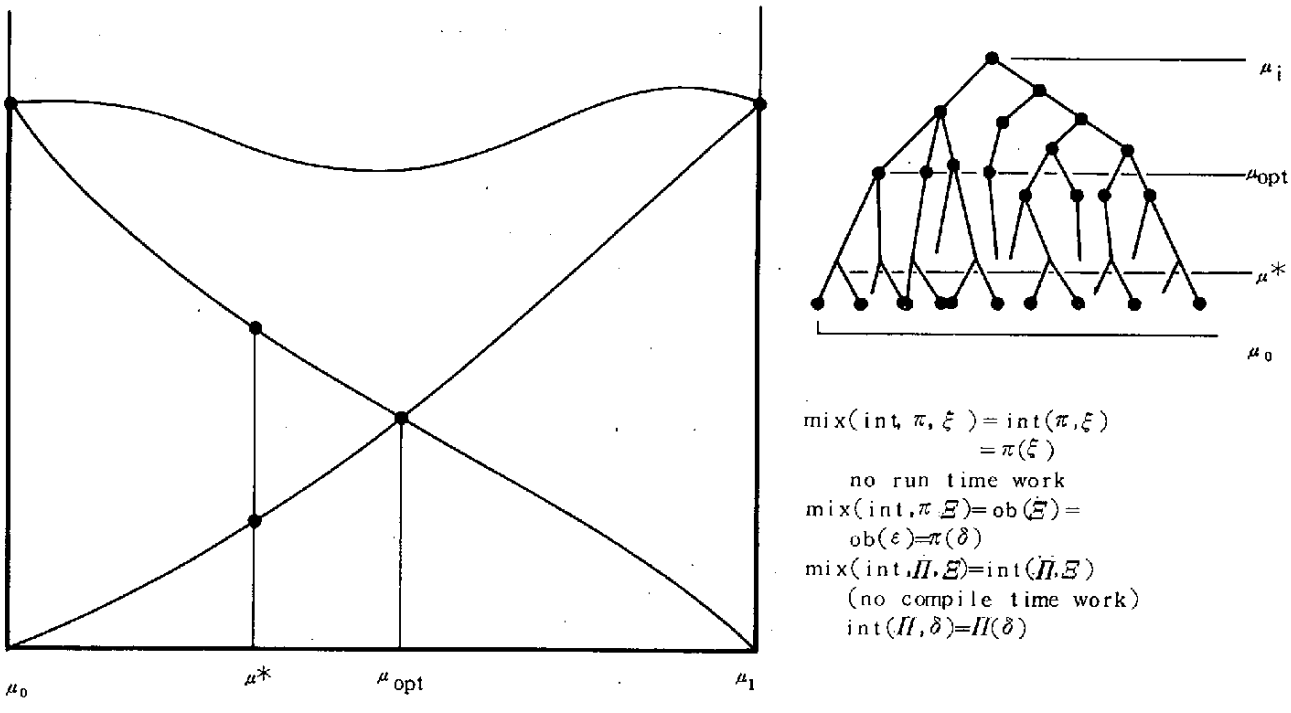


図 4　コンパイルと実行時間の兼ねあい

ここで混合計算モードで，このプログラムを実行します。すると，ｘを含んだ文，あるいはｘに依存した部分の実行は後まわしにして，剰余プログラムに送りこまねばなりません。このような機構を実現すれば，剰余プログラムとして，ｙ：＝1・ｘ；　ｘ：＝x^2；　ｘ：＝x^2；　ｙ：＝ｙ・ｘ　を得ます。最終的なプログラムはこれに通常の最適化を行うことにより得られます。これは演算的接近法の例です。

さて，変換的接近法について述べましょう。私には，アルゴル類似の言語と再帰型言語の２つに対する基本変換規則がありますが，これについて細かく説明する時間はありません。図５にアルゴル型言語に対する基本変換規則が示してあります。どれも単純なものばかりです。ただ式⑤でＤｘ とあるのは，プログラム中でｘが代入文の左辺に現われたところから始めて，次にｘが代入される前までの間でｘが右辺に出現している。その出現をｘの動作域と呼びＤｘで表わしています（従って式⑤は情報的に関連のあるｘの出現を残らず別の文字ｙで置きかえてよいことを意味します）。

Basic Transformations for
Algol like programs (System M)

① Term reduction. Constant expansion

$$\frac{\vdash T = C}{P(\tau) \sim P(c)}$$

② Reduction by true Conditioning by true

if true then s1 else s2 fi ~ s1

③ Reduction by false Conditioning by false

if false then s1 else s2 fi ~ s2

④ Loop expansion. Loop reduction

while Π do s od ~ if Π then s; while Π do s od fi

⑤ Domain separation. Domain merge

$$\frac{\neg(y \text{ occurs in } P)}{P(Dx) \sim P(Dy)}$$

⑥ Substituting by source. Expression economy

$$\frac{\exists ! x := E \text{ in } P}{P(x_a) \sim P(E)}$$

⑦ Unloading. Loading

$$\frac{\neg(x_a \text{ occurs in } P)}{P(x := E) \sim P(\varnothing)}$$

図5　アルゴル型プログラムの基本変換規則

この7つの規則はあらゆるプログラム変換，つまり混合計算において完全系であります。

最初の例をこの規則によって変形し，この規則の"力"を見てみましょう（内容は省略，参考文献参照）。

このようにして，この7つの規則だけで，もし全データが与えられていれば，プログラムの実行が，また，一部のデータしか与えられていないならば剰余プログラムが生成されます。

結論に移りましょう。良い実現言語はこれらの基本規則を含んだものでなくてはならず，それによってすでに述べましたあらゆるプログラム変換，つまりプログラムの実行，最適化等が一様に記述できるものでなければなりません。この考えにより，変換機械を考えることができます。その命令語は上の変換規則です。これらをハードウェアで実現することも考えなければなりません。このような考えの例は，ほかにＩＢＭのCocke氏もやっています。ノヴォシビルスク大学ではプログラムモデルを作っています。Ostrovskiiと私は，ＬＲ(１)パーサをPascal言語に射影し，それに最適化を行うことにより，もとのＬＲ(１)パーサより4.5倍高速になることを見ました。

高速で高能力の将来の計算法の基本設計には，このようなプログラム変換の統一的な接近法

およびそのハードウェア実現が有用であると納得していただけたと思います。また理論的・数学的な考え方が計算機の基本設計の諸決定に対して有用であることも示しています。このような考えが計算機の基本設計に対する数学好みの接近法の1つだと考えてはいけないと思います。私の同僚のKotov氏がこのＩＦＩＰの招待講演で言語を考慮した並列機械の基本設計を述べますが，これも理論的な立場から計算機の基本設計を考えたもう一つの例だと思います。

質問：どのような機能が将来の計算機の基本設計に重要と考えられますか。
答：一般に情報は図4の木構造の端節に置かれます。このような木構造において根から端節に高速で到達する機構が重要だと思います。

〔参考文献〕

A.P. エルショフ：混合計算 ― その応用の可能性と研究問題（ロシア語；この英訳はNorth Holland 社から出版される予定（1981）である）

Corresponding member of Academy of
Sciences of USSR
計算センタ （ノヴォシビルスク）
A.P. Ershov
（電総研 パターン情報部 数理基礎研究室
宮川正弘 訳・編）

2. 各 W G 報 告

〔1〕 並行及び並列システムの形式的記述

1. 概　要

本報告では並行並列システムを記述する理論モデルのいくつかを取り上げて，その記述形態，正当性の証明機構，記述能力等について解説する。

まず，第2章でSequentialなProgramに対する理論モデル，検証系，証明系について簡単に触れ，証明手段，検証手段，としてどういうものがあるかを点検する。理論モデルとしてはFloyd-Hoare流の公理系主義と，Scott流のmodel主義が二大潮流であるのでそれらについて触れる。公理主義の立場に立つと証明系をそろえることが主眼となる。model出義の立場に立つと，プログラムconstructに指示物（denotation）としてsetを与えることが主眼となるが，どちらもプログラムの証当性を証明できる機能をもっている。

第3章では並行プログラミング言語，並列プログラミング言語，を発展過程に則して調べ，基本的な計算モデルのあり方，言語側のプログラムconstructとしてのとらえ方を見る。計算modelとしては，共有変数を媒介としてsequential processが通信を行うmodel＝並行プログラミングmodelと，各processがdataを直接交換するmodel＝並列プログラミングmodelがあるが，並行プログラミングでは共有変数へaccess制御を記述するプログラムconstructの発展過程，並列プログラミングmodelではpure modelからside effectを取込んでゆく過程を追跡する。

第4章では公理論的立場から非決定性並列プログラムとその理論を数学的にきちっと定式化できる体系としてSuzuki-FlonのapproachとPrattのDynamic logicについて述べる。

並列プログラムの性質について論証できる公理系としてはDynamic logicが形式的に最も整っている。公理系を設定した場合の研究目的の一つにその公理系の完全性，決定問題の可解性，無矛盾性等の証明があるが，それらについても一通り述べる。

第5章では，並列プログラムを入出力ポートを通じて直接交信するプロセスのネットワークとしてとらえるMilne modelについて述べる。プロセス概念はHewittのactor等との関連を考えるとき，またData Flow Networkとの関連を考えるのも興味深い。Scott modelを拡張したPlotkin model上でprocessをとらえると並行プログラムで重要なdeadlock等の問題について論証できない。そこでMilnerはprocess間に新らたにinductiveな等価性の定義を与えることによって問題を解決している。

2. Sequential プログラムに対する正当性の証明

Dijkstraがソフトウェアの危機を訴えて以来，ソフトウェア技術の一つの大きな目標は，いかにしてソフトウェアの生産性を向上させるかという課題にある。近年ハードウェア技術の長足の進歩により，計算機システムの価格に於けるハードウェアとソフトウェアのしめる費用の割合は1対10に広がって来ており，ハードウェアに比してソフトウェアの技術進歩の遅れが目立っている。ソフトウェアの生産性を向上させるには，

1. module化，構造化
2. 仕様記述の明確化
3. プログラミング言語の高級化
4. Debug systemの高級化
5. 自動プログラミング

等が叫ばれてきた。その中でプログラムそのものに反省がなされ，プログラムというものを数学的にきちんと定義し，出来上ったプログラムの正当性を証明できるシステムの開発及び仕様記述からプログラミング迄，その正当性を検証できる形で展開する方法の開発がなされてきた。

プログラムの意味を数学的に厳密に定め，その正しさを証明しようという試みはFloyd，Naurのinvariant assertion methodにより一つの方向が打出されたといって良かろう。

Hoareがこれを一つの公理系にまとめたことから，Floyd Hoare logicと呼ばれる。

いま

$$x = 0 \Rightarrow$$

$$< \text{while} \quad x < 5, \quad \text{do } x \leftarrow x + 1 > x = 5$$

なる文を考える。＜　＞内がプログラム，＜　＞の外がassertionである。$x = 0$ならばwhile……なる文を実行したのち$x = 5$が成立すると読む。

$$P(x) \equiv x \leq 5$$

とおくと$P(x)$はプログラムの実行の全ての時点で成立するinvariantである。

また　　ⅰ）　$x = 0 \Rightarrow P(x) \wedge x < 5$

ⅱ）　$P(x) \wedge x < 5 \Rightarrow P(x+1)$

ⅲ）　$P(x) \wedge x \geq 5 \Rightarrow x = 5$

が成立するからwhile文の意味を

$$(P \wedge C \Rightarrow < S > P) \Rightarrow (P \Rightarrow < \text{while} \quad C \quad \text{do} \quad S > P \wedge \neg C)$$

とすると，$P = P(x)$，$C = x < 5$，$S = x \leftarrow x + 1$とおくことによりⅱ）を使って，

$$x < 5 \wedge P(x) \Rightarrow < \text{while } x < 5 \text{ do } x \leftarrow x + 1 > P(x) \wedge x \geq 5$$

ⅰ）とⅲ）から　$x = 0 < \text{while } x < 5 \text{ do } x \leftarrow x + 1 > x = 5$

が導びかれ，この文が正しいことが証明される。

ここに示した例がinvariant assertion methodの一つのひな型である。この方法はな

かなか有効であるが，invariant を発見するのがしばしば困難であるといわれている。証明をなるべくmechanical なレベルへもっていける方法としてA．J．Milner 等による Logi-c for Computable Function が開発された。

ＬＣＦについて述べる前に我々はもStrarchy によるプログラミング言語の形式的意味論について述べる必要がある。universal な計算体系として知られているものに

1） Turing machine

2） Post system

3） Churchs λ - Calculus

4） Kleenes General Recursion

等がある。そのうち λ - Calculus は全ての対象を一変数の関数と見なし，計算を代入と見るというunique な観点に立っている。このため高階な関数をも普通の関数であるように取扱えることになる。これはプログラム言語の定義のように高階なプログラム construct を記述するには極めて便利な点である。

例えば

$1 = \lambda a \lambda b \quad a(b)$

$2 = \lambda a \lambda b \quad a(a(b))$

としよう。即ち1はaとbという二つの関数を入力してa（b）即ちbにaをapply した結果得られる関数を出力する関数である。一変数の関数と見なすときはaを入力してλ b a(b) なる関数を出力するものと見なすのである。

このとき

$S = \lambda a \lambda b \lambda c \quad b((a(b))(c))$

という関数を考えると

$S(1) = \lambda u \lambda v \quad u(((\lambda a \lambda b\ a(b))(u))(v))$

$= \lambda u \lambda v \quad u(u(v))$

$= \lambda a \lambda b \quad a(a(b)) = 2$

即ち

$S x = 1 + x$

となる。

ところで関数 $\lambda x(x(x)(\lambda x\ x(x)))$ もλ-Calculus の意味で関数でなければならない。

しかるに代入計算をほどこしても

$$\lambda x\ x(x)(\lambda x\ x(x)) = \lambda x\ x(x)(\lambda x\ x(x)) = \cdots\cdots$$

となってλ x x(x)は関数としての値が定まらない。この場合 λ x x(x)(λ x (x(x)))をど

ういう関数と見なすべきかに疑問を生ずることになる。この点に関して明確な解答を与えたのがScott modelである。λ－Calculusの関数はøとω（自然数全体）をそれぞれ 最小元，最大元とする連続ラティス上の連続関数であると考えると矛盾のない解釈が得られるというのである。

例えばScott model では

$$\lambda\ x\ x(x)(\lambda\ x\quad x(x)) = \emptyset$$

である。（applyの解釈の仕方で実は任意のλ－式にできることがわかっているがøにもできる）

$$Y = \lambda\ u\ (\lambda\ x\ u\ (x(x))\ )(\lambda\ x\ u\ (x(x))\ )$$

なる関数を考えると

$$\begin{aligned} Y\ u &= (\lambda\ x\ u\ (x(x))\ )(\lambda\ x\ u\ (x(x))\ ) \\ &= u\ (\lambda\ x\ u\ (x(x))\ )(\lambda\ x\ u\ (x(x))\ ) \\ &= u\ Y\ u \end{aligned}$$

となりYuがuの不動点であることがわかる。Yuはlatticeの順序に関して実は最小であって従って，

$$Yu = u(\emptyset) \sqcup\ u(u(\emptyset) \sqcup \cdots\cdots \sqcup\ u^n\ (\emptyset) \sqcup \cdots\cdots$$

であることがわかる。

この関数YはParadoxical combinatorと呼ばれrecursionを定める。これを使ってwhile文のsemanticsを規定してみよう。

簡単なstate transition modelを考える。

S：state

exp：state → Boole

statement：state → state

即ち，machineは stateをもち，expはBoolean expressionで0 1 valueをとる。

またstatementはstate transitionを引起すのみとする。

すると

$$\text{while}(\text{exp})\ \text{do}(\text{statement}) = \lambda\ s\ Y_{\lambda u}\quad \text{Cond}\ (\text{exp}\ (s),\ u\ (\text{statement}\ (s)),\ s)$$

によりwhile文の意味が規定できる。

実際

$$\begin{aligned} &\text{while}(\text{exp})\ \text{do}\ (\text{statement})\ (s) \\ &= Y_{\lambda u}\quad \text{Cond}\ (\text{exp}\ (s),\ u\ (\text{statement}\ (s)),\ s) \\ &= \text{Cond}\ (\text{exp}\ (s),\ \emptyset,\ s) \\ &\sqcup\ \text{Cond}\ (\text{exp}\ (s),\ \text{Cond}\ (\text{exp}\ (\text{statement}\ (s)),\ \emptyset, \end{aligned}$$

statement (s)), s)

⊔ Cond (exp (s), Cond (exp (statement (s)),

Cond (exp (statement (statement (s))), ∅,

statement(statement (s))), statement (s)), s)

であるから exp (statement (statement(s)))=1

ならば　　=statment (statement (s))

となって意味が合う。

プログラムの意味というものが極めて抽象的にしかも厳密に定義し得ることはプログラムの正しさを証明するといった作業に於いて重要な意義をもつ。

$\neg <$ while E do $\Gamma > E$

を示してみよう。

while E do Γ (s)

= Cond (E (s), ∅, s)

⊔ Cond (E (s), Cond (E (Γ (s)), ∅, Γ (s)), s)

⊔ Cond (E (s), Cond (E (Γ (s)), Cond (E (Γ^2(s)),

∅ Γ^2 (s)), Γ (s)), s)

⊔

⋮

であるが

$n_0 = \min \{ n \mid E(\Gamma^n(s)) = 0 \}$

とすると　while E do Γ (s)

$= \Gamma^{n_0}$

従って

$\neg E(\Gamma^{n_0}(s)) = \neg E$ (while E do Γ(s))

$= <$ while E do $\Gamma > \neg E$

から

$<$ while E do $\Gamma > \neg E$

が成立。

ＬＣＦはプログラム construct を λ − Calculus で定義すると同時にその Construct を用いて書いた問題のプログラムと λ − Calculus でそれとは独立に直接書いたプログラムとの等価性を調べることにより，プログラムの正しさを証明しようという試みである。

λ − Calculus に於ける計算は等価変形であるから２つの異なる λ − 式を変形していって同一の式が得られればもとの２つの式が等しいことがわかる。この手法を検証手段として用い，

適用範囲を広げたものにAbstract Data Type によるプログラムのspecification 及び検証システムがある。

Abstract Data Type の考え方に従うと全ての世界はData と Data を Data へ移す演算からなっている。Data はそれにaccess する演算によって定義され，その実装詳細について知る必要がない。その意味でまさにAbstractである。

例えばstackは三つのobject (or sort)

stack , item , Boolean , Λ

からなり

newstack : $\Lambda \rightarrow$ stack

undefined : $\Lambda \rightarrow$ item

push : stack × item → stack

pop : stack → stack

is new : stack → boolean

top : stack → item

なる演算が定義されている。

我々はstack が通常の意味でstack として機能するには

is new (newstack) = true

is new (push (s , i)) = false

pop (newstack) = newstack

pop (push (s , i)) = s

top (newstack) = undefined

top (push (s , i)) = i

が成立しなければならない。

これをシステムの公理とする。

いまこの等号＝を→に置き換えると一つの書換システムが得られる。書換システムはPost が示したようにuniversal な計算機能をもつから上記システムと見たてることができる。

例えば

pop pop (push (push (s , i_1) , i_2))

= pop (push (s , i_1))

= s

である。

pop pop (push (push (s , i_1) , i_2)) をプログラムであると思えば上記の式変形は計算である。逆にpop pop (push (push (s , i_1) , i_2)) = s を知って

いれば公理系の正しさを確めていることになる。

Abstract Data Type による計算システムではプログラムを与えてその正しさを証明するというよりもシステム全体を抽象的に記述し，その正しさを確めるという考え方になっており，それがまた仕様記述言語といわれる所以になっている。

Abstraet Data Type のこの考え方に近いものとしてPROLOGに代表されるLogie programming がある。Logic Programming ではプログラムしたいシステムの機能をHorn Clauseという一階述語論理のsubset で記述し，プログラムはそのシステムで更に或る条件がかされたとき(入力)，何が成立するか（出力）を問うこととなる。実行は定理証明そのものであるから最初の記述が誤っていなければ得られる結論は常に正しいわけでプログラムの正当性の証明そのものが不用でないかとも思えるが，逆に自動証明が困難であることも想起する必要がある。

例えばFactorial のプログラムをHorn Clause で書くと

$$\begin{cases} F(0,1) \\ F(n,n\cdot x) \Leftarrow F(n-1,x) \end{cases}$$

となる。

0！＝1，n！＝n・（n－1）！を意味している。

以上示したようにプログラムの正当性を証明しようという目標に対して実にいろいろな角度から努力が払われて来ていることが理解できよう。

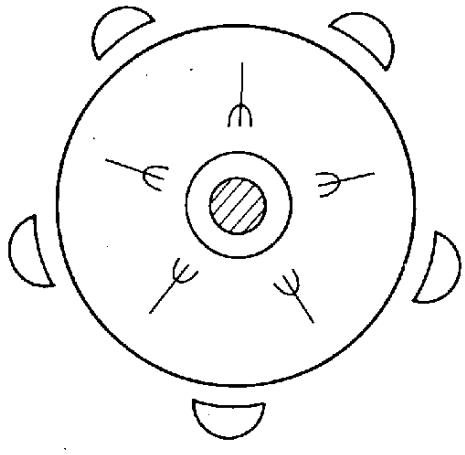
3. 並行並列プログラミングシステム

ＴＳＳシステムの開発の結果，そのＯＳは多重プログラミングシステムをsupportすることとなり，同時に巨大化することとなった。

ＭＵＬＴＩＣＳを代表とするＴＳＳシステムでは後に述べるような並行プログラミング言語を使用することが出来なかったので大変な開発日時を要した。この経験から多くのプロセスのinteractionを可能とする並行プログラミング言語を構築することがソフトウェア工学の課題として浮び上ってきた。また並行プログラミングに於いてはresorce sharing，deadlock，livelock等のsequentialプログラミングには表われない問題の重要性が認識され，deadlock，livelockを避けるためのresorceへのaccess制御を行うプログラムconstructの研究及びdeadlockが生じないことを証明するための様々な数学的形式化の研究が行われるに至った。

簡単なdeadlockの問題としてDijkstraによるDining Philosopherの問題を取上げよう。

いま５人の哲学者が思索にふけっては食事をとるという行為を繰返すものとする。彼等は同じ食堂で食事をとるものとし，食堂には５つのイスと真中に１つの丸テーブルがありイスとイスの中間の位置のテーブル上にフォークが置いてあるものとする。



テーブルの上には従って５つのフォークが置いてあり真中に１つの皿があってスパゲティが盛ってある。スパゲティをフォークでとるには１本では足らず２本のフォークを要する。哲学者は腹がへると食堂に来て適当なイスにすわり，まず左手のフォークをとり次に右手のフォークをとってスパゲティを食べるものとしよう。ここでもし５人が同時に食堂に来てイスにすわり，２本のフォークをとろうとすると，これは実行不可能である。これがdeadlockと呼ばれる現象である。

O.S－Virtual Memory system の構築をきっかけとしmulti－processor systemというHardwareの環境の中で育ってきた並行プログラミングとは多くのプロセスが共有変数を媒介とし，deadlockを回避しながら進行する計算モデルを前提としている。

3.1. 並行プログラミングの系譜

並行プログラミング言語の構築に当って最初に考えられたのが多数のプロセスを発生させ，またそれらの同期をとる命令である。

```
Folk      par begin
  ⟩         ⟩
Join      par end
```

これらのprocessは並行実行後にDijkstraがguarded commandなるものを発明した為に，非決定性並列実行が重要性を増すことになった。

```
Guard1  :   command1
  ⟩           ⟩
Guard n :   command n
```

～のcommandのうちGuardの条件が満たされたものが実行に移る。

共有変数保護のためのプログラムconstructとして最初に提案されたのがDijkstraによるsemaphoreを用いたP，V－operationである。これはもちろん表現として十分であったが，primitiveすぎたので次にConditional Critical regionがBrinch Hansenによって考案された。これは共有変数を臨界領域に入るときの条件を各プロセスの定義に於いて明示するものであった。Hoare及びBrinch Hansenはこれを更に発展させて，共有変数の定義に於いてそれにaccessするprocessとその同期関係をまとめて記述し，Monitor手続きと命名した。又，これはBrinch HansenによるConcurrent Pascalに実装されている。Monitor手続きはque変数に対するdelay操作により手続き間の同期をとっているが，これをより記述的に表現しようというのがCambell and Harvermanらの順路式（path expression）による同期条件の表現である。これは各プロセスが共有変数にaccessする仕方をそのaccess系列によって表現しようとするものでその手段として正則式が使用される。

message bufferのmonitorによる記述の例をあげる。

```
type page = array ( 1‥length ) of integer ;
type buffer =
monitor
var contents : page ;
        empty : boolean ;
        sender , receiver : que ;
procedure entry send ( message : page ) ;
begin
```

```
            if not empty delay(sender);
            contents = message ;
            empty    = false   ;
            continue ( receiver ) ;
        end ;
        ..........
        begin .......... end ;
```

これをpath expressionを用いて書くと

```
        type page = array (1.. length ) of integer ;
        type buffer =
        var contents : page ;
        path ( send | recieve )* end ;
        export procedure send ( message : page ) ;
            begin
                contents = message
            end ;
        ..........
            begin
                :
            end  ;
```

となる。

3.2. 並行プログラミング

現在進行中のＶＬＳＩ化はハードウェア，ソフトウェアの両面に於いて大きな変革を引起しそうである。一つは超密結合マルチプロセッサ・システムが実現する可能性がある。第２はその前提に基づいてそれを利用するプログラミング・システムとして並行プログラミング言語が開発されるというものである。

並行プログラミング言語の前提とする計算モデルの大きな特徴はsequential processが共有変数を媒介にして計算を押し進めるというものであった。このため

1. 共有変数を媒介とするやりとりをuserが指定する必要がある。
2. 共有変数にまつわる問題があるためプログラムの正当性を確めるのに困難が生ずる。
3. 細かいレベルの並列性を書くのに共有変数とそれを中心とする同期機構という大掛りな装置を必要とする。

といった点であろう。

共有変数によらないプロセスの同期通信機構として様々なモデルが提案されるに至った。

1. Hoare のＣ．Ｓ．Ｐ．
2. Hewitt のActor model
3. Dennis の Data Flow model

HoareのＣ．Ｓ．Ｐ．ではプロセス間の通信機構としてＩ．Ｏ文を用いDijkstraのGuarded CommandのGuard文に入力文が書けるようにし，入力文が実行可となったときにGuardが解けて実行できるものとし，これによってプロセスの同期をとる機構とした。

Ｃ．Ｓ．Ｐ．によるメッセージバッファプロセスは

```
X : :
buffer : portion ;
in : bit ;
in : = 0 ;
*〔 producer ? buffer → in : = 1
 □ in = 1 ; consumer ? more (  ) →
   consumer ! buffer ; in : = 0 〕
```

により定義される。ここでXはプロセス名，＊は繰返し文，producer ? bufferはproducerから入力が可能であればbufferに入力する。

A→BはAがGuardでAがtrueになるとBが可能となる。

Hewitt の actor model はミクロレベルの並列プログラムを書く言語を設定したという点で並列実行テデルとしては大きな前進であった。

actor model の設定には

1. Abstract Data type 型の記述形式を実際の並行プログラムに於けるような side effect が記述できる形に拡張する。
2. Denotational semantics による go to 文の記述に於いて発明されたものに continuation という概念があるが，continuation の送受信及び continuation の生成という現象がミクロレベルの計算を支えている。
3. プロセスの起動にはメッセージに対するパターンマッチングによるものとし従って，caller に対する return adress の格納といった操作を必要としない。

といった考え方がベースになっているようである。

いま go to 文の semantics を考えてみよう。go to 文では行き先き即ち文番号が go to の引数となる。文番号とはプログラムの位置指定であるがこれは実行の状態が与えられるとその位置から end にいたるルート全ての集まりを決める。このルートは更にそのルート を通ってプログラムが走るときに得られるプログラムの出力でおきかえられる。

以上から文番号を実行状態 s が与えられたときにその後の出力列を与える関数と見なせる。これは Denotational semantics の用語で continuation と呼ばれる。通常のプログラムでは statement の実行毎に文番号が変わり従って continuation が移る。このことは

statement : Continuation → Continuation

であることを意味する。

この解釈のもとで while 文の semantics は次のように書き換えられる。

while exp do statement

$= \lambda c \lambda s \quad Y_{\lambda u}$ Cond (exp (s)

statment (u (c)) (s) , c (s))

これは次の図に示すように

while exp do statement ; C end .

↑
S

$$= \begin{cases} \uparrow_{S} \ C \quad \text{end} . & \text{exp} = \text{true のとき} \\ \uparrow_{S} \ \text{statement} ; [\ \text{while exp do statement}\ ] \ C \ \text{end} . & \text{exp} = \text{false のとき} \end{cases}$$

状態 S で while 文に入り C により決まる出力系列をへて end するプログラムは S で exp が

false ならば statement を評価せず単にCにより決まる出力系列をへて endにいたり，またSでexp が true ならば statement を評価しwhile 文を評価し，Cによる出力列をへて end に至るということを意味する。

このcontinuationという概念を用いるとfactorial のプログラムは

$$factorial = \lambda\, c\, \lambda\, n\, Y_{\lambda u}\ Cond(n=0,\ c(1),\ u(\lambda\, k\ \ c(n \times k))(n-1)).$$

となる。

stateはnでtactorial を行った結果state がfactorial（n）へ移ると考えられる。すると上式はstate n＝0でfactorial 実行後のcontinuationがc のときstateが1に移ってその上でcがstartする。n≠0のときは，stateがn－1でfactorial実行後のcontinuationが$\lambda kc(n \times k)$であるときと同じ出力行動をとる。

上式を

$$factorial = \lambda\, cont : c\ \lambda\, arg : n\ .\ Cond(n=0,\ c \Leftarrow result : 1,\ factorial \Leftarrow cont : c',\ arg : n-1)$$

但し $C' = \lambda K\ .\ C \Leftarrow result : n \times k$

と書き直す。

そしてfactorial，continuation，c，c'をcomputing agent と見なし，actorと名付ける。又，⇐をmessage の受信と考える。するとactor factorial はcontinuationとargという2つの入力port及びresultという出力portをもちcontinuationはresult という入力portのみをもつactorと考えられる。そして計算がこれらactor間のメッセージの送受信というマイクロレベルの計算に分解される。actor model ではλ－Calculusで全てのobject が関数であったのと同じように全てのobject がactorであると考える。

actor model による並列計算の例としてfolk－join behaviorをとりあげよう。

関数fをf（x）＝h（g_1（x），g_2（x））でg_1 とg_2 をconcurrentに評価するものとしよう。

$$f = \lambda\, arg : x\ \lambda\, cont : C.$$

$g_1 \Leftarrow arg : x,\ cont : C_\ell$ ； $g_2 \Leftarrow arg : x, cont\ \ C_r$

但し $C_\ell = \lambda\, result : r\ .\ joiner \Leftarrow left : r$

$C_r = \lambda\, result : r\ .\ joiner \Leftarrow right : v$

$joiner = \lambda\, cases$

left : u →

λ right : v . h ⇐ arg 〈u , v〉, cont : C

right : v →

λ left : u . h⇐arg : 〈u , v〉, cont : C

のようにfを定義する。

まずg_1 にarg xとcont Cℓ , g_2 にarg xとcont Crが送られる。g_1 はresultをCℓに送る。Cℓはresult,rを受け取ってjoinerにleft･rを送る。g_2 crについても同様。

joinerに先にleft･rが送られるとそれはright v_1 の到達を待ってペアarg〈u, v〉をつくりcont Cとarg〈u, v〉をhにおくるactorに変身する。right : vが先に到着した場合も同様。

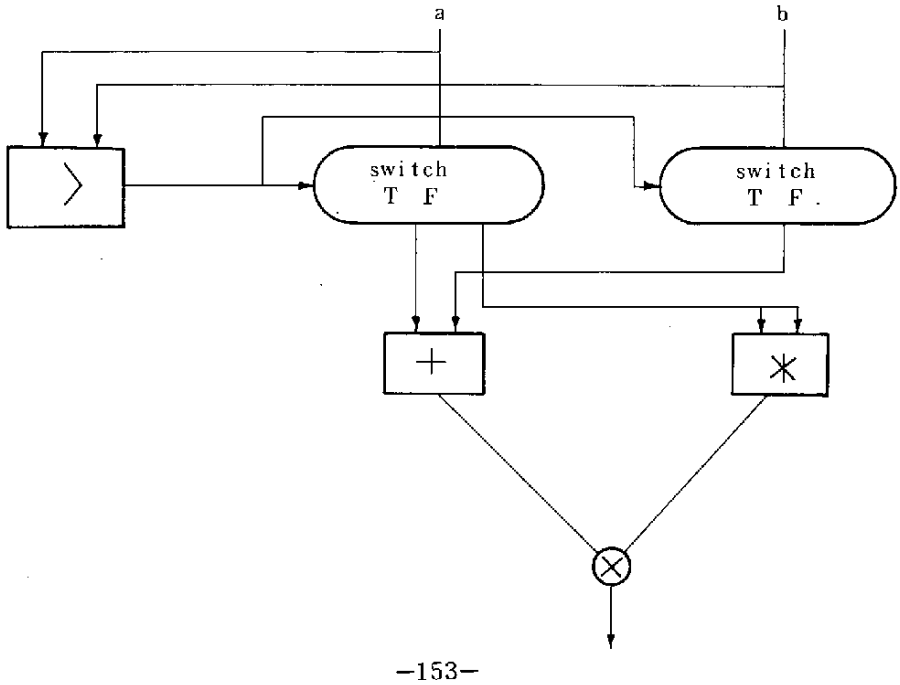
actorが内部にメモリーをもつ場合をimpure actor,そうでない場合をpure actorと呼ぶ。impure actorを用いるとcorcurrentなprocessを表現できる。これらについては文献にゆずる。

Hewittのactor modelはPLASMAという言語に結実したがそれをsupportするHardware Architectureについては未だ手掛りがついていない。DennisらのいわゆるData Flow modelの重要性の一つはそれを実現するHa ware Architectureにいくつかのプランがあることである。

Dennisはプログラム言語のcompilerのうち特にoptimiserを作る際，Flow graphを作るが，これを並列プログラミング言語として採用できないかという点から出発したそうである。そのmodelはペトリネットといわれるものに良く似たものでデータをになうtokenと呼ばれるものが演算elementを流れるうちに加工されて出てくるといったものである。演算elementの起動はデータの到着によるものでデータdrivenと呼ばれる。

例えばプログラム

if a 〉 b then a ＋ b else a ＊ a



なるデータFlow graphに置き換えられる。まずaとbがcomparator [>] へ到着するとその結果が2つのswitchへ流れていって，そのT，Fの値に従って，蛇口をひねる。Tならばaとbが[+]に流れついて結果がa＋bとして出口から出てくる。＊についても同様である。

データFlow modelについてはプログラミング言語としてもまたその金物実現を考える場合でも多くの課題があった。

1. データFlow graphの書式化
2. Do - loop，Block，recursionの作り方
3. side effectのとり込み

等がsoftware的な課題であったといえるであろう。

これらの問題はIrvineの言語Idに於いてほぼ完全に解決されているといって良かろう。まずtokenはdataを運ぶだけでなくそれと一緒に（u．c．s．i）Pというtagを運ぶ。ここで

u ： procedure applicationのcontext

c ： tokenのいる手続名

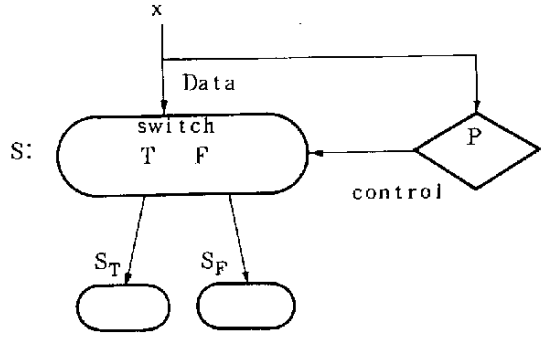
s ： tokenのいるその手続内の演算名

i ： iteration count

p ： tokenのいる手続内の演算のport名

である。

いま



とするときswitchは

$$input = \{ < x,\ u.c.s.i > data,\ < b,\ u.c.s.i > control \}$$

$$output = (b = true \rightarrow \{ < x,\ u.c.s_t.i > \},\ b = false \rightarrow \{ < x,\ u.c.s_f.i > \})$$

なる働きをする演算である。即ちtoken＜x，u．c．s．i＞dataが入口data より又token＜b，u．c．s．i＞controlが入口controlよりswitch sへ入ったとするとbのtrue，falseに従ってoutput tokenがそれぞれSt或いはSfへ流れる。

同様にloopに入るときにtokenのtagを初期設定する演算としてLなる演算子を考えた。

input ：{＜x，u．c．s．i＞}

output ：{＜x，u'．c．t'．1＞}　　但しu'＝(u．s．i)

これはstatement名sとiteration count iをcontextとして格納しinteration count を1に初期設定し，statment名を次のstatement名で置き換えたものをtagとするtoken をはき出すものである。

iteration countを1つ増やすoperationとして次のDが定義できる。

input ＝{＜x，u'．c．t．j＞}

output ＝{＜x，u'．c．t'．j＋1＞}

これらに対して後始末のためにD^{-1}, L^{-1} という2つのoperatorが定義される。

D^{-1} ：　input ＝{＜x，u'．c．w．n＞}

output ＝{＜x，u'．c．w'．1＞}

L^{-1} ：　input ＝{＜x，u'．c．w'．1＞}

output ＝{＜x，u．c．s'．i＞}

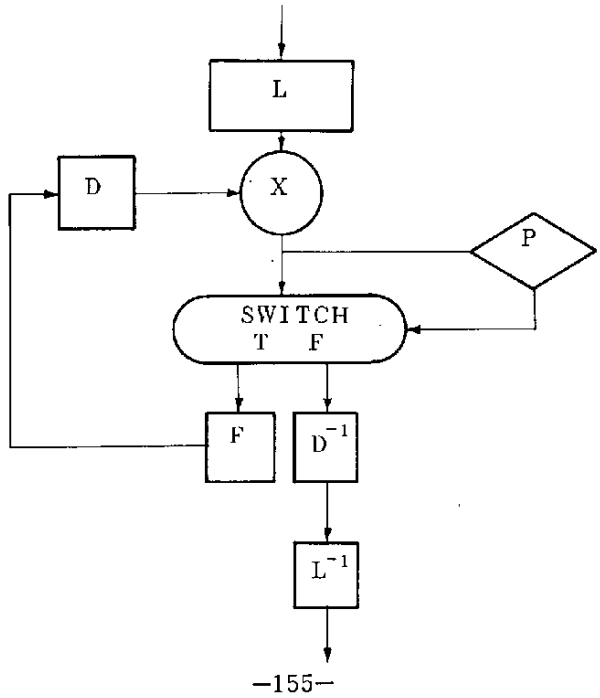
但しu'＝(u，s，i)

即ちD^{-1}はitreration countを1にもどし，L^{-1}はもとのcontextを回復する。

プログラム

```
initial  x← a
while  p (x) do
    new  x← f (x)
return  x
```

は次のdata flowグラフで実現できる。



以上でData Flowでloopを実現する方法がわかったがBegin Blockの実現等も同様にしてできる。次の問題はside effectをどういう形で実現するかである。

tokenの列で最後がestで終るものをsteamと呼ぶ。このsteamの履歴に従って実行すべき演算の種類を変えることにより（いわゆるHistry sensitivityの実現である。）side effectを実現しようというのがその解決法である。

まずcounterというstreamは次のプログラムにより生成される。

```
A ← ( intial i ← 1 ; next ← 1
    for counter from 1 to k do
      new i ← nexti
      new nexti ← i + nexti
    return all i )
```

ここでreturn all iはk個のtekenを一まとめにし，最後にestというtokenをつけたものとして出口から流す働きをする。

stream上の演算を実現するにはまずstreamの表現を決めねばならない。

$$<\!\!\not<X_k,\ k\!\!\not>,\ u\ .\ c\ .\ s\ .\ i>_p$$

によりstreamのk番目のtokenを表現する。

そしてstreamをばらす演算Eを

$$input = \{<\!\!\not<X_k,\ k\!\!\not>,\ u\ .\ c\ .\ s\ .\ 1> \mid 1 \leq k \leq \#X\}$$

$$output = \{<X_k,\ u\ .\ c\ .\ s'\ .\ k> \mid 1 \leq k \leq \#X\}$$

またstreamを作り出す演算E^{-1}を

$$input = \{<X,\ u\ .\ c\ .\ s\ .\ i>\}$$

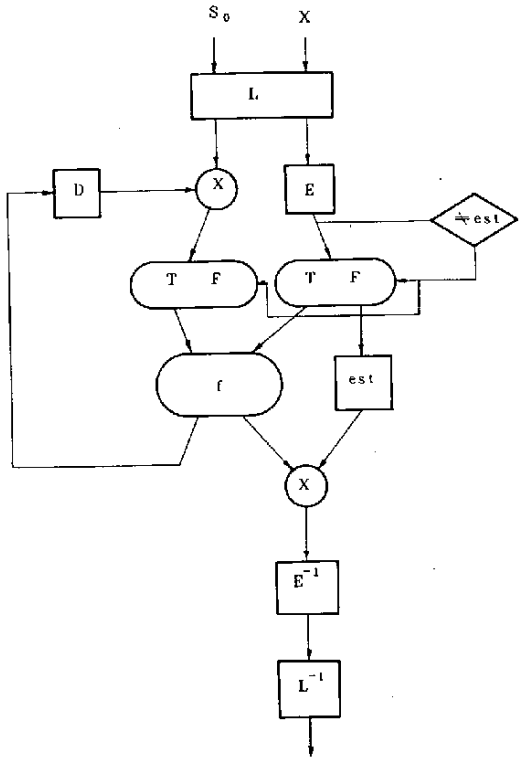
$$output = \{<\!\!\not<X,\ i\!\!\not>,\ u\ .\ c\ .\ s'\ .\ 1>\}$$

と定める。

これによりsをresourceの使用状態とする簡単なresource manager

```
mdl ← manager( s0 )
  ( entry X do
      Result ← (initial S←S0
      for each x in X do
      < new s , answer > ← f (s,x)
      return all answer )
    exit Result )
```

は



となって実現される。

但しD，Lはstream用に若干変更する。

以上並行並列プログラミングの動向を見てきたが

1. 共有変数を媒介とするcommunicationを前提とし，共有変数に対するaccessの同期の形式を整える。

といった並行プログラミングの研究から

2. 共有変数を媒介とせずdataを直接交換することにより通信を行い従って同期の問題を避けると同時にミクロな並列実行を可能とし，そのメカニズムの上にresourceの共有できる言語をimplementしていく。

という並列プログラミングの研究へと発展してきている。

このように並列プログラミングに関して一つの明確な方向が打出されてきていると見られるわけで，その意味論を構築する前提が整ったと考えられる。また並列プログラムの意味を考える場合，それは

1. プロセスの同期に関する命題を議論できる。
2. 非決定性並列演算が取込まれている必要がある。

といった条件を満たす必要があることがわかる。

4. 公理的方法

並行プログラミング言語が設定されるとその構文に対応した証明技術がFloyd － Hoare 流の立場から開発されてきた。ここではそれらを個々に取上げるのではなく，並行プログラムをそれと等価なDijkstraのGuarded Command に近い形のnon － deterministic プログラムに変換し，それを一般形としてそれにHoare 流の公理系を設定するというFlon － Suzukiの理論をまづ紹介しておく。

いま

cobegin $x \leftarrow x + 1$ // $x \leftarrow x + 1$ coend

というプログラムが与えられたとする。これに対応するDijkstra の Guarded Command 言語を用いたnon － deterministic program は

do

$p_1 = 0 \rightarrow t_1 \leftarrow x \ ; \ p_1 \leftarrow 1$ //

$p_1 = 1 \rightarrow x \leftarrow t_1 + 1 \ ; \ p_1 \leftarrow 2$ //

$p_2 = 0 \rightarrow t_2 \leftarrow x \ ; \ p_2 \leftarrow 1$ //

$p_2 = 1 \rightarrow x \leftarrow t_2 + 1 \ ; \ p_2 \leftarrow 2$ //

od

である。ここでp_1，p_2は2つのプロセスに対するプログラムカウンタの役割を演じており，最初は$p_1 = p_2 = 0$でこのプログラムに入るものとする。

これに対しGuarded Commandが満たされなくなると休止し脱出のためにはexit という command をもつ構文を考える。上記のプログラムを

rep

$p_1 = 0 \rightarrow x_1 \leftarrow x \ ; \ p_1 \leftarrow 1$ //

$p_1 = 1 \rightarrow x \leftarrow t_1 + 1 \ ; \ p_1 \leftarrow 2$ //

$p_2 = 0 \rightarrow t_2 \leftarrow x \ ; \ p_2 \leftarrow 1$ //

$p_2 = 1 \rightarrow x \leftarrow t_2 + 1 \ ; \ p_2 \leftarrow 2$ //

$p_2 = 2 \wedge p_2 = 2 \rightarrow$ exit

per

と書き換えると元のcobegin ～ coend のプログラムと等価なnon-deterministic program が得られる。

rep構文はこのように並行並列プログラムを記述できる。いまrepの一般形を

rep $B_1 \rightarrow S_1$ // ····· // $B_n \rightarrow S_n$ per

とする。

並行プログラミングに於いて重要な概念として

1. Invariance ····プログラムの実行によって影響を受けない性質。
2. Potentiality ····プログラムの実行によって成立する可能性がある性質。
3. Inevitability ····プログラムの実行によってかならず成立する性質。
4. Bloking - free ····計算が途中で止まったり，ループに入ったりしない。
5. jに関し，Deadlock -free ····計算は必ず出口に到達するか乃至はゲート B_j が開いて続行する。
6. j に関しStarvation - free ····計算はかならずゲート B_j を開ける。

等がある。

これらをrep構文に対して適用したものを公理系にまとめると

1. Invariance

$$\forall_{k\in\{1,\cdots,n\}} \ (I \wedge B \Rightarrow [S_k] I) \wedge (I \Rightarrow P)$$
$$\vdash I \Rightarrow \mathrm{Invariant}(\mathrm{Rep}, P)$$

2. Potentiality

$$\neg Q(0)\, Q(m+1) \wedge \neg P \Rightarrow \exists_{k\in\{1,\cdots,n\}} (B_k \wedge [S_k] Q(m))$$
$$\vdash Q(m) \Rightarrow \mathrm{Potentially}(\mathrm{Rep}, P)$$

3. Inevitability

$$\neg Q(0) \wedge Q(m+1) \wedge \neg P = \exists i \in \{1,\cdots,n\}\, Bi$$
$$\wedge \forall k \Leftarrow \{1,\cdots,n\} \ (B_k \Rightarrow [S_k] Q(m))$$
$$\vdash Q(m) \Rightarrow \mathrm{Inevitably}(\mathrm{Rep})$$

4. Blocking-free

$$\forall k \in \{1,\cdots,n\} \ (I \wedge B_k \Rightarrow [S_k] I) \wedge (I \Rightarrow \exists k \ B_k)$$
$$\vdash I \Rightarrow \mathrm{Blocking\text{-}free}(\mathrm{Rep})$$

5. Deadlock-free

$$\forall k \in \{1,\cdots,n\} \ (I \wedge B_k \Rightarrow [S_k] I)$$
$$\wedge (I \Rightarrow \mathrm{Potentially}(\mathrm{Rep}, Bj))$$
$$\vdash I \Rightarrow \mathrm{Deadlock\text{-}free}(\mathrm{Rep})$$

6. Starvation-free

$$\forall k \in \{1,\cdots,n\} \ (I \wedge B_k \Rightarrow [S_k] I)$$
$$\wedge (I \Rightarrow \mathrm{Inevitably}(\mathrm{Rep}, Bj))$$
$$\vdash \mathrm{Starvation\text{-}free}(\mathrm{Rep})$$

となる。

但しここで，〔 α 〕Pは，αの計算がとまってPが成立することを意味する。また2と3は前提の対偶をとるとわかりやすい。

いまstrvation - freeを例をあげて証明してみよう。

```
cobegin
    with x , y when x > 0∧y < n do x←x − 1 ; y ← y + 1 end //
    with y , z when y > 0∧z < n do y←y − 1 ; z ←z + 1 end //
    with z , x when z > 0∧x < n do z←z − 1 ; x ← x + 1 end
coend
```

なるプログラムに於いてQ(m)を

$$Q(m) = 0 \leq x,\ y,\ z \leq n \ \wedge\ 0 < x+y+z < 3n \wedge m = x + 2z + 3y$$

と定義する。

$$Q(0) \equiv \mathrm{False}$$

$$Q(m+1) \wedge (x \leq 0 \vee y \geq n) \Rightarrow (y > 0 \wedge z < n \vee z > 0 \wedge x < n)$$

$$\wedge (y > 0 \wedge z < n \Rightarrow [\ y \leftarrow y - 1\ ;\ z \leftarrow z + 1\ ]\ Q(m))$$

$$\wedge (z > 0 \wedge x < n \Rightarrow [\ z \leftarrow z - 1\ ;\ x \leftarrow x + 1\ ]\ Q(m))$$

であるから公理3より

$$Q(m) \Rightarrow \mathrm{Inevitably}\ (\ \mathrm{Rep}\ ,\ x > 0 \wedge y < n\ )$$

$\exists m\, Q(m)$はinvariantだから公理6よりstarvation - freeがいえる。

このSuzuki-Flow Approachに見るように，非決定性並列プログラムを取扱うことにより，並行並列プログラムの多くの問題を論じることができる。そこでプログラムの形式をよりpureに整え，論証の形式を公理系にまとめたものにPrattのDynamic logicがある。

Dynamic logicでは公理系の完全性決定可能性についても明解な議論があるので紹介しておこう。

Regular First order Dynamic logicのwell formed formulaは次のように定義される。

可算個の関数記号 f_0，f_1 ‥‥述語記号 p^0，p^1，‥‥‥

及び変数記号 x_0，x_1 ‥‥が与えられているとき

項とは

ⅰ) 変数 x_i は項

ⅱ) fがn変数の関数記号で α_0，‥‥ α_n が項のとき f (α_0，‥‥ α_n) は項

well formed formulaの全体をWFF，Regularプログラムの全体をRGとすると

ⅰ) 変数xと項eに対しx← e ∈ RG

ⅱ) 原始式P∈WFF

iii) $\alpha, \beta \in RG \Rightarrow \alpha ; \beta, \alpha \cup \beta, \alpha* \in RG$

iv) $P, Q \in WFF, \alpha \in RG$のとき

$P, P \vee Q, \exists x P, <\alpha> P \in WFF$

v) Program修飾即ち$<\alpha>$をもたない任意のWFF Pに対し

$P? \in RG$

このWFF及びRGに対しmodel解釈を与えることができる。

いま領域Dが与えられているとしIはk-ary function symbol f及びPredicate symbol Pに対し

$$f^I : D^k \to D, \quad P^I : D^k \to \{true, false\}$$

なるf_I, p_Iをassignするmapとする。このようなmap Iはstateと呼ばれる。

Uをこのようなstateの集まりで更に

$\forall f$：k変数の関数記号　$\forall F : D^k \to D$

$\forall I \in U \quad \exists J = [F/f] I$

を満すものとする。

即ち$I \in U$ならIのfに於ける値をFに変更することによって得られるstate $J = [F/f]I$もUに属す。

このときRGのプログラム及びWFFの式の意味を

(1) $m(x \leftarrow e) = \{(I, J) \mid J = [e_I / x]\}$

但し$e_I = f_I (e_{1I}, \cdots, e_{nI})$

(2) $m(P?) = \{(I, I) \mid I \models P\}$

(3) $m(\alpha ; \beta) = m(\alpha) \circ m(\beta)$

$m(\alpha \cup \beta) = m(\alpha) \cup m(\beta)$

$m(\alpha*) = m(\alpha)*$

(4) $I \models P(e_1, \cdots, e_k) \equiv P_I (e_{1I}, \cdots, e_{kI})$がtrue

(5) $I \models \neg P \equiv \neg I \models P$

$I \models P \vee Q \equiv I \models P \vee I \models Q$

$I \models \exists x P \equiv \exists d \in D \quad [d/x] I \models P$

$I \models <\alpha> P \equiv \exists J (I, J) \in m(\alpha) \wedge J \models P$

によって定義する。

論理学的観点に立つときDynamic logicについては公理系として多くのことが証明されている。

1) regular first order dynamic logicはその妥当な式全体が帰納的に可算でないようにmodelを作ることができるので有限公理化可能ではない。

2) regular first order dynamic logicのうちWFFを〔α〕P, <α>Q, Rの形でP, QRがプログラム修飾を受けないものに制限したものはdomainをarithmatic universeにとるとき完全。

3) regular propositional dynamic logicは決定問題が可解。

4) context-free propositional dynamic logic は unsolvable。

1)はHarel, Meyer, Prattの結果であり，全域で帰納的な関数の停止問題を妥当な式とするmodelを作り，全域で帰納的な関数の全体が帰納的に可算でないことを利用する。

2)はプログラム修飾を受けた論理式を通常の一階の論理式に分解するというCookの手法を適用し，一階の論理式とそこでの公理系の完全性問題に話をすりかえる。

3)はConstableによる。〔α〕P型sentenceがαに或る制約を課したオートマトンがacceptするstringをもつかどうかという問題の妥当性と等価になる。

4)は3)と同じ手法で証明する。

ここでは2)の意味で完全性の証明されているfirst order Dynamic logieの公理系について触れておこう。

公　理

（P）　命題論理の公理

（←R）　〔x←e〕P≡P^{e}_{x}

（?R）　〔Q?〕P≡Q⇒P

（;R）　〔α;β〕P≡〔α〕〔β〕P

（∪R）　〔α∪β〕P≡（〔α〕P∧〔β〕P）

推論規則

（MP）　P, P⇒Q⊢Q

（G）　P⇒Q ⊢ 〔α〕P⇒〔α〕Q

　　P⇒Q ⊢$^{\exists}$ x P ⇒ $^{\exists}$x Q

（I*）　P⇒〔α〕P⊢〔α^{*}〕P

（C*）　P（n＋1）⇒<α>P（n）

　　⊢ P（n）⇒<α^{*}>P（0）

5. Milner model

Milnerは入出力ポートをもちそれにより直接data交換を行なうプロセスのネットワークにより並列実行を実現するモデルを考え，プロセスに直接semanticalな記述を与えることを試みた。

Actor modelではcontinuationが並列プロセス記述のkey pointであった。Soquential なmodelではcontinuation Cは

$$C\ ;\ Val \rightarrow \Sigma^*$$

但しValは入力値Σは出力値の全体

である。

しかし communicating processを考える場合は出力系列は入力portから入る他のprocessからの値によっても影響を受けるので仮に

$$C : Val^* \rightarrow \Sigma^*$$

但し$c(xy) = a(x)c'_x(y)$とできる

としてよかろう。

$$P(x) = \lambda y\, c(xy)$$

と定義すると

$$P(x) = a(x)\lambda y\, C'x(y)$$

いま$\lambda y\, C'x(y) = \lambda z \lambda y\ C'x(zy)$と仮定すると

$$P(x) = a(x)\lambda z \lambda y\, C'x(zy)$$
$$= a(x)\lambda z\, P'x(z)$$

P(x)をプロセスと呼ぶとプロセスとは出力してから他のプロセスに変身するものであること，またsequential programの場合のcontinuationと，密接な関連があるものであると見ることができる。

プロセス全体をP，入力領域をV，出力領域をUとするとき上記のmodel では

$$P = U \times (V \rightarrow P)$$

が成立している。

いまLをportラベルの全体とし，プロセスとはport $\mu \in L$に対するP_μのいくつかの和としよう。すると

$$P_L = \sum_{\mu \in L} U_\mu \times (V_\mu \rightarrow P_\mu)$$

である。

さて最後にnon-deterministicな実行を許すために$\sum_{\mu \in L} U_\mu \times (V_\mu \rightarrow P_\mu)$のどれかを指定するものと考え

$$p \subset \sum_{\mu \in L} U_\mu \times (V_\mu \rightarrow P_\mu)$$

とする。

即ち

$$P_L = P\left(\sum_{\mu \in L} U_\mu \times (V_\mu \to P_\mu)\right)$$

が成立するものとする。

ここで新たにP_Lの元をprocessと定義する。

Registerの定義を例としてあげよう。2つのportをそれぞれ$\bar{\alpha}$, $\bar{\gamma}$とし

$$U_{\bar{\alpha}} = V_{\bar{\gamma}} = \{0\},\quad V_{\bar{\alpha}} = U_{\bar{\gamma}} = Z = \{\perp, 0, \pm 1, \pm 2, \cdots\cdots\}$$

とするとき

$$\mathrm{Reg}\ z = \{<0, \lambda z'\ \mathrm{Reg}\ z' >_{\bar{\alpha}}, <z, K(\mathrm{Reg}\ z)>_{\bar{\gamma}}\}$$

但し$K = \lambda x \lambda y\, x$

とおく。

$$\mathrm{Reg} : Z \to P\{\bar{\alpha}, \bar{\gamma}\}$$

でReg zはport $\bar{\alpha}$からz'が入力すると，Reg z'へ（即ちzをz'へ書き換える。）またport $\bar{\gamma}$からはtriggerがかかり次第Registerの内容zを出力するわけであるからReg zはRegisterの内容がzである状態を表わしている。

プロセス間に2つの演算を導入する。

1) $p \mid p' \equiv \{<u, \lambda\nu(f\nu \mid p'>_\mu \mid <u, f>_\mu \in p\}$
$\cup\{<u', \lambda\nu(p \mid f'\nu')>_\mu \mid <u', f'>_\mu \in p'\}$
$\cup\cup\{(f\overset{1}{u} \mid \overset{1}{f}u) \mid <u, f>_\mu \in p, <u', f>_{\bar{\mu}} \in p'\}$

2) $p \setminus \alpha \equiv \{<u, \lambda\nu(f\nu \setminus \alpha)>_\mu \mid <u, f>_\mu \in p, \mu \in \{\alpha, \bar{\alpha}\}\}$

いま簡単のために

$p = \{<u_1\ f_1>_\alpha, <u_2\ f_2>_\beta\}$

$p' = \{<\nu_1\ q_1>_\gamma, <\nu_2\ q_2>_{\beta'}\}$

としよう。

$p \mid p'$は

と図示される。

port βと$\bar{\beta}$は接続されており常に情報が交換されるものとする。またそのことを別とすると全ては非決定的に実行されるものとする。

αからinput⇒出力u_1, next process $\lambda\nu\ f_1\nu \mid \overset{1}{p}$

γ 〃 ⇒ 〃 ν_1, 〃 $p \mid \lambda\nu\ g_1\nu$

$\beta - \bar{\beta}$で情報交換⇒βからの出力uがβ'への出力，又β'からの出力ν_2がβへの入力でnext processは$f_2\ \nu_2 \mid g_2\ u_2$であるから

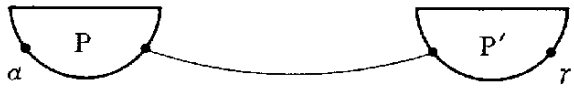
$$p \mid p' = \{<u_1, \lambda\nu\ f_1\nu \mid p'>_\alpha, <\nu_1, p \mid \lambda\nu\ g_1\nu>_\gamma\}$$

$$U\{(f_2\ \nu_2 \mid g_2\ u_2)\}$$

となって1)が得られる。

2)は内部結合されたport名を削除する演算。

例えば　$p \mid p' \setminus \beta$ は



と図示されることになる。

pがラベルset L，p'がラベルset Mをもつとする。

$$N=\{\alpha \mid \alpha\in L \wedge \alpha\in M \vee \bar{\alpha}\in L \wedge \bar{\alpha}\in M\}$$

とおき，

$$p \mid\mid p' \equiv p \mid p' \setminus N$$

により $p \mid\mid p'$ を定義する。

ラベルの付けかえをS；L→Mとするとき

p_L に対し

$$p[s]=\{<u,\ f>_{s(\alpha)} \mid \alpha\in L\}$$

によりラベルを付けかえたprocess p[s]を定義する。

更にprocessの記法を若干変更する。

$$p=\{<u_1,\ f_1>\alpha,\ <u_2,\ f_2>\beta\}$$

に於いて f_1，f_2；Val → Pであるから

$$f_1=\lambda\ \nu_1\ f'_1(\nu_1),\quad f_2=\lambda\ \nu_2\ f'_2(\nu_2)$$

と書ける。そこで

$$p=\alpha\ \nu_1\ \bar{\alpha}\ u_1\ f'_1(\nu_1)+\beta\ \nu_2\ \bar{\beta}\ u_2\ f'_2(\nu_2)$$

と書くことにしよう。そしてαをinput label，$\bar{\alpha}$をoutput ラベルと呼ぶ。

この書き方に従ってQueueを定義すると

$$queue\ (Nil)=\alpha x\ queue(x)$$

$$queue\ (x.s)=\alpha y\ queue(x.s.y)+\bar{\beta}\ x\ queue(s)$$

となる。

いま次のようなprimitive processを用意しよう。

1) Do f = i x $\bar{0}$ f(x) Do f

2) Ask p = i x　if　p(x) then $\bar{0}_1$ x Ask p
else $\bar{0}_2$ x Ask p

3) Gate = isc $\bar{o}$x $\bar{\tau}$ Gate

γ に triggerがかかるごとにGateがあく。

4) Trig = $i x \bar{r} \bar{o} x$ Trig

i に data x が到着するごとに trigger r に従って x を流すかどうか決める。

5) Sink = i x Sink

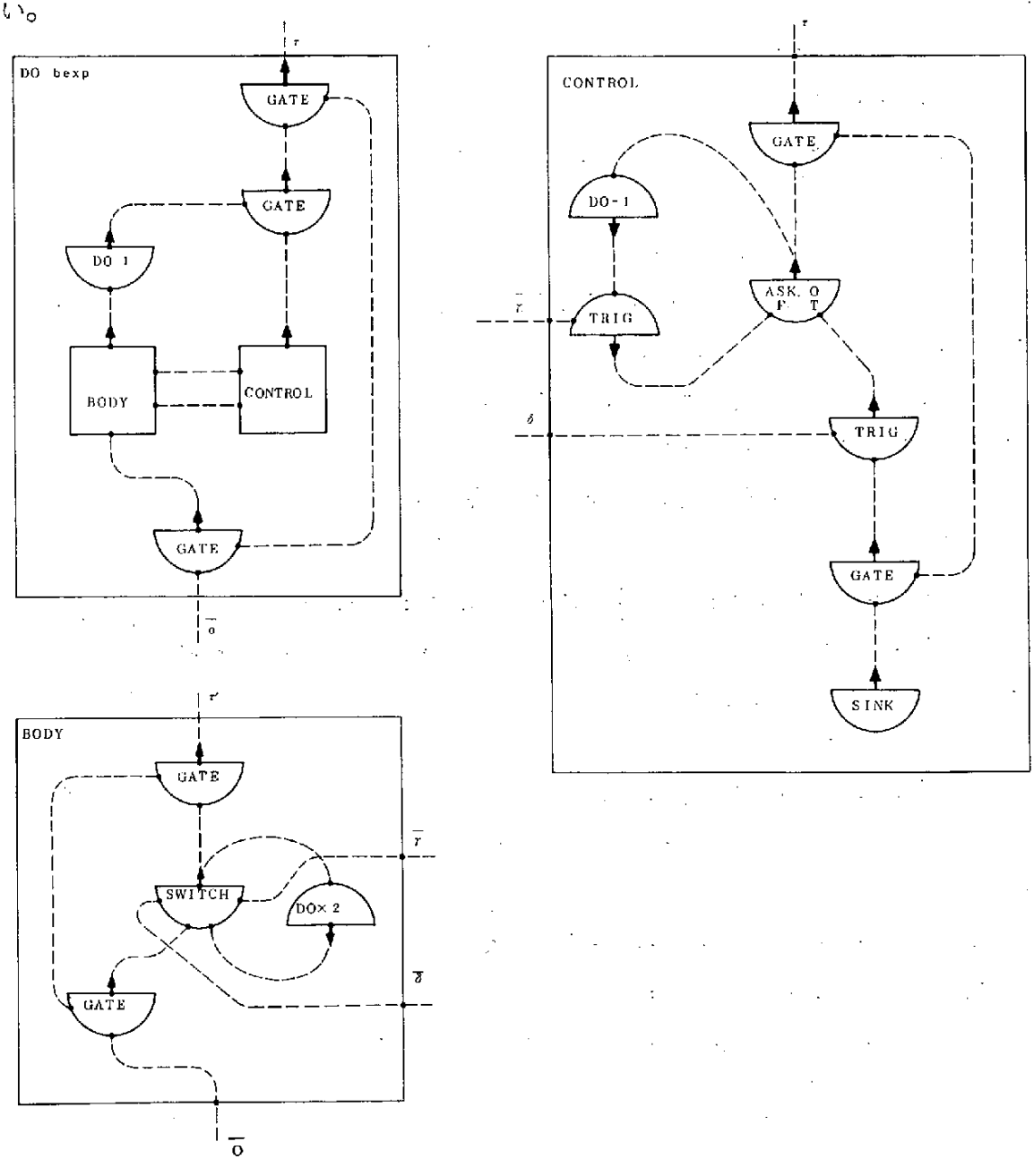
6) Switch = $i x(r_1 \bar{o}_1 x$ Switch + $r_2 \bar{o}_2 x$ Switch)

これらのプロセスを以下に示す図のように接続することによって

$$P = \alpha x \bar{\beta}\ 2^x\ P$$

なる process を実現できる。

またこのプログラムが primitive な Data-Flow プログラムになっていることに注意されたい。



このようにMilner modelはactor modelをよりpureな形で展開していると同時にData Flow modelをも包含していると考えることができる。Actor modelとの対応で考えるとき

$$p = \alpha\, x\, \bar{\beta}\, f(x)\, p$$

なるプロセスはいわゆるpure actorで関数f(x)を実現している。

$$p = \alpha\, x\, \bar{\beta}\, f(x)\, q + \gamma\, x\, \bar{\delta}\, g(x)\, p$$

$$(p \neq q)$$

なるプロセスはimpure actorでside effectを表現している。

さてMilner modelについて一応理解していただけたと思うが，Milner modelに於いてプログラムの正当性を証明するといった議論をするにはどうしたらよいであろうか。Milnerはこれに対してprocess間の等価性を定義し，それをinductionによって証明するという方法を提示している。

いま

$$p = \alpha\, x\, \beta\, y\, \bar{\gamma}\, (x + y)\, p$$

というプロセスがあったとしよう。pに対し入口2から3が，次にβから4が入力し，$\bar{\gamma}$から7が出力する系列を

$$p \overset{\alpha 3}{\rightarrow} \beta\, y\, \bar{\gamma}\, (3 + y)\, p \overset{\beta 4}{\rightarrow} \bar{\gamma}\, (3 + 4)\, p \overset{\bar{\gamma} 7}{\rightarrow} p$$

と書くものとしよう。

すると一般に $\overset{\mu\nu}{\rightarrow}$ という関係が考えられる。これはλ-Calculusや Combinatory logicでは書き換規則そのものが式の意味を規定するのだという考え方に立っている。それに対するScottの反論としてdenotational semanticsが提出されてきたわけである。Milnerのプロセスは

$$P_L = P\left(\sum_{\mu \in L} (U \times (V \rightarrow P_\mu))\right)$$

なる方式を満たす領域P_Lによって決められるが，この式はPを含んでいるためScott modelのconstructionでは作れない。そこでPlotkin等によってPower domain constructionが開発されたわけである。

しかるにPlotkin modelではdeadlockをもつprocessともたないprocessが論理的に等価となってしまう可能性がある。従って書き換え規則型のsemanticsの方が適当であろう。

いま

$$P = (\bar{\alpha}\, a_1 \,.\, b_1 + \bar{\beta}\, a_2 \,.\, b_2) \mid \beta x \,.\, b_3(x)$$

$$= \bar{\alpha}\, a_1 \,.\, (b_1 \mid \beta x \,.\, b_3(x)) + \bar{\beta}\, a_2 \,.\, (b_2 \mid \beta x \,.\, b_3(x))$$

$$+\beta\,d\,.\,(\,\bar{\alpha}\,a_1\,.\,b_1+\bar{\beta}\,a_2\,.\,b_2\,)$$
$$+\,b_2\mid b_2\,(\,a_2\,)$$

であるから

$$P\xrightarrow{\bar{\alpha}a_1}\ b_1\mid\beta\,x\,.\,b_3\,(\,x\,)$$
$$P\xrightarrow{\bar{\beta}a_2}\ b_2\mid\beta\,x\,.\,b_3\,(\,x\,)$$
$$P\xrightarrow{\beta d}\ (\,\bar{\alpha}\,a_1\,.\,b_1+\bar{\beta}\,a_2\,.\,b_2\,)\mid b_3\,(\,d\,)$$
$$P\xrightarrow{\varepsilon}\ b_2\mid b_3\,(\,a_2\,)$$

である。

ところで上記4例のうち上の3例はPの外部とのcommunicationであるが，最下段はPの内部でのcommunicationの結果であって観測不可能であり，null derivationで移れる。

そこでnull derivationだけの違いは等価性に影響を与えないものと考える。

そこで

$$B\xrightarrow{\varepsilon^{n_1}}\cdot\xrightarrow{\mu_1\nu_1}\cdots\cdots\xrightarrow{\varepsilon^{n_r}}\cdot\xrightarrow{\mu_r\nu_r}\cdot\xrightarrow{\varepsilon^{n_r+1}}B'$$

を

$$B\overset{\mu_1\nu_1\cdots\mu_r\nu_r}{\Rightarrow}B'$$

と書くことにする。

そして⇒に基づいて同値関係≈を次のように定義する。

1) $\forall B,\ C\qquad B\approx_0 C$

2) $B\approx_{k+1}C$

$\equiv_{fd_D}\forall\, s\in(\Lambda\times V)^*$

i) $B\overset{s}{\Rightarrow}B'\Rightarrow^{\exists}C'\quad C\overset{s}{\Rightarrow}C'\wedge B'\approx_k C'$

ii) $C\overset{s}{\Rightarrow}C'\Rightarrow^{\exists}B'\quad B\overset{s}{\Rightarrow}B'\wedge B'\approx_k C'$

このとき

$$B\approx C\equiv_{f_D}\forall k\quad B\approx_k C$$

電総研　制御部　論理システム研究室

大谷木．重夫

【参考文献】

(1) D. B. MacQueen "Models for distributed computing" Rapport de Recherche No. 351. April 1979

(2) S. K. Basu, R. T. Yeh "Strong verification of programs" IE^3 Trans. Software Engineering Vol. SE-1 No. 3 September 1975

(3) S. A. Cook "Soundness and Completeness of an Axion System for Program Verification" SIAM J. Compt. Vol. 7 No. 1 February 1978

(4) R. Milner "LCF: A way of Doing Proofs with a Machine" Proc. 8th MFCS Symposium, Olomouc, Czechoslovakia (1979)

(5) L. Aiello, M. Aiello, R. W. Weyhrauch "Pascal in LCF: Semantics and Examples of Proof" Theoretical Computer Science 5 pp 135-177 1977

(6) G. D. Plotkin "LCF Considered as a Programming Language" Theoretical Computer Science 5 p 223-225 1977

(7) J. A. Goguen "Initial Algebra Semantics and Continuous Algebras" JACM. Vol. 24 No. 1 January 1977 pp 68-95

(8) Arvind, J. B. Dennis "Tutorial Seminar on New Computer Architechture" IFIP Congress 80.

(9) C. Hewitt, P. Bishop, R. Steiger, I. Greif, B. Smith, T. Matson, R. Hale "Behavioral Semantics of Nonrecursive Control Structures" LNCS 19 pp 385-407 April 9-11, 1974. Springer

(10) C. Hewitt, "Protection and Synchronization in Actor Systems "MIT AILab. working paper 83 November 1974

(11) C. A. R. Hoare "Communicating Sequential Process" CACM Vol. 21 No. 8 August 1978 pp 666~

(12) J. H. Howard "Proving Monitors" CACM Vol. 19 No. 5 May 1976 pp 273~

(13) C. A. R. Moare "Monitors: An Operating Systems Structuring Concept" CACM Vol. 17 No. 10 October 1974 pp 549~

(14) L. Flon, N. Suzuki "Consistent and Complete Proof Rules for the Total Correctness of Parallel Programs" Proc. 19th IE^3 Symp. on Foundations on Computer Science October 1978

(15) G. Milne, R.Milner "Concurrent Process and their Syatax" J. ACM Vol. 26 No. 2 April 1979 pp 302-321

(16) D. Harel, A. Pnueli, J. Stavi "A Complete Axiomatic System for Proving Deductions about Recursive Programs" Proce. 9th Ann. ACM Symp. on Theory of Computing, Boulder, Col., May 1977.

(17) G. D. Plotkin "A Powerdomain Construction" SIAM J. Computer Vol. 5 No. 3 pp 452~ September 1976

(18) M. B. Smith "Powerdomains" LNCS 45 pp 537-543 Springer 1976

(19) A. Church "The Calculi of Lambda-Conversion" Annals of Mathematics Studies 6 1941.

(20) D. Scott "Lambda Calculus and Recursion Theory "Proc. 3rd Scandinavian Logic Symposium p 154-193 (North Holland)

(21) J. E. Donahue "Complementary Definitions of Programming Language Semantics" LNCS Vol. 42

(22) D. Harrel "First order Dynamic Logic" LNCS Vol. 68

(23) R. Milner "A Calculus of Communicating Systems" LNCS 92

(24) V.R. Pratt "A Practical Decision Method for Propositional Dynamic Logic" Proc. 10th Ann. ACM Symp. on Theory of Computing, May 1978

(25) R. L. Constable "The Role of Finite Automata in the Development of Modern Computing Theory" Proceedings of the Kleene Symposium June 18-24 1978 at Madison 1980 NH. pp 61~

〔２〕 PROLOGの拡張に関する若干の考察

1. 証明と並列探索

ここでは論理プログラミングの祖先である定理証明の性格を調べて，論理型計算言語，PROLOG〔Kowalski 79〕への並列機能の導入について少しコメントしてみたいと思う。

定理証明とは言うまでもなく，公理の集合Γから結論Pを許された推論規則を使うことにより導びき出すことであるが，公理のとり方，推論規則の組み合わせに応じて色々なタイプの定理証明が存在する。今迄の定理証明の多くは一階述語論理の範囲内のものが多く，又ΓからPを直接導びく代りに，$\Gamma \cup \{\sim P\}$を節（clause）形式に変換した後導出法（resolution method）により矛盾を導びき出す反証（refutation）タイプのものが大部分である。

Gentzen流又はHilbert流の直接的証明法によるにしろ，導出法を使った反証法によるにしろ，証明を構成するためには可能な推論の鎖を１つずつつなぎ合わせ証明木（反証木）を作り出さなければならない。

これを探索の立場から眺めると，可能な推論（証明木）の全体から成る証明空間を探索して必要とする証明を見付け出すわけであるが，このような探索の手間は通常莫大なものとなる。これは例えば命題論理の充足可能性問題がＮＰ－完全であることからもその一端が知れる。

定理証明を証明空間の探索として捉えるとその特徴は第１にこの証明空間の広大さと，第２は副作用のないことである。ここで言う副作用がないというのは，探索順序が証明空間に影響しないということである。この事は当り前でもあるが，例えば積木の世界を考え，その中のsingle－hand のロボットが目標の形に積木を積むというplanningの問題などと比べた場合には大きい特徴である。副作用がないことにより証明空間を並列に相互干渉なしに探索することができる。（PROLOGはこの証明空間を逐次探索し，行き詰るとバックトラックを始めて，他の方向を探索し直す。並列探索可能な証明空間をわざわざ逐次探索しているわけである。この点に着目し，多数のプロセッサー，巨大なメモリの存在を前提にPROLOGを並列に動かそうという提案も既に幾つか出ている。〔中島８０〕〔田村８０〕）。しかし，やみくもな並列探索はすぐに数の爆発を起すことは明らかである。結局定理証明の問題には２つの要因があるように見える。

(A) 計算機パワーを活かして，並列探索を行い，解を早く見付け出したい。

(B) むやみに並列探索を行うと数の爆発が起きて，計算機パワーで処理しきれなくなる。

PROLOGを証明言語としてみると並列探索は当然であり，(B)のようなことがあるにしても並列実行が望ましい。しかしPROLOGは現在既に計算言語である（例えばoccur checkの無視）。このことを認めた上で計算言語としてPROLOGを考えてみると必ずしも並列実行が望ましいとは言えない面がある。というのはプログラムの作製，或いは虫取りの思考活動

は（目的指向型の）逐次的活動であるので，PROLOGの（top－down）逐次実行は人間の思考形態にふさわしいとも言えるからである。例えばPROLOGを並列に実行するとして，虫取りに困難が生じないだろうか。といって並列実行を全く捨ててしまうのが得策とも言えない。

関数引数の並列評価や，guarded command の条件部の並列評価等有用な並列実行があるからである。これらはいずれも非常に制限された形を持ち，何が起るか予想できる並列実行である。（決定性の並列実行）翻がえって(B)を眺めると，これは何が起るか予想できない非決定的な問題に無制限の並列実行を導入した報いと言える。（非決定性の並列実行）

プログラムの作製，虫取り等の論理的活動は（恐らく）逐次実行が適合する。PROLOGに並列実行を導入するにしても，このような論理的に追い易い逐次実行に導びかれた，しかも非決定性の少ない場面に導入することが望ましいと思われるがどうだろうか。

PROLOGに制限された形の並列実行を取り入れたものとして local horn の提案がある。〔渕７７〕 local horn は bottom up，並列に実行される horn 文の集合であり，PROLOGのプログラムに埋め込まれた形で局所的に述語を定義する。top－down，逐実実行が bottom-up 並列実行を制御しているわけである。しかし local horn も無制限に大きいものであれば数の爆発が起きることは充分に考えられる。

いずれにしてもPROLOG を並列実行したり，並列実行を局所的に取り入れる際は，数の爆発を押える効果的な制御構造を取り入れることが不可避である。

2. 一般的な節の取り扱い

現在PROLOGは horn 節しか扱うことができない。これを任意の節まで取り扱えるように拡張したい。ここでの提案は，場合分けの原理（ＣＡ… Case Analysis）を取り入れることにより，一般節に対する証明を horn 節用に工夫されたＳＮＬ，ＳＰＵなどの戦略〔Kuehner 70〕（以下 horn 戦略と呼ぶ）の繰り返しに帰着させることである。

人間の基本的推論形式には，３段論法や数学的帰納法があるが，その他大切なものとして場合分けがある。場合分けによる証明を形式的に書くと，

$$A \vdash C,\ B \vdash C \iff A \vee B \vdash C$$

と書くことができる。場合分けによる証明は Gentzen 流，或いは Hilbert 流の推論体系には既に組み込まれているが，計算機に適した導出法の分野ではまだハッキリと導入されたことがない（ようである）。場合分けによる推論・証明は極めて適用性が広く，色々な形で導出法の分野に取り入れることができるが〔佐藤 8.0〕，ここでは余り一般的にせず，horn 戦略との関連で述べることにする。

場合分けによる一般節の取り扱いを例で示そう。次に示す節集合Sは充足不能である。

$$S \triangleq \{ A_1 \vee A_2 \ , \quad \sim A_1 \vee A_2 \ , \ A_1 \vee \sim A_2 \ , \ \sim A_1 \vee \sim A_2 \ \}$$

Sにはnon－horn節$A_1 \vee A_2$があるので，horn戦略を直接適用することができない。そこで，$A_1 \vee A_2$をhorn節A_1，A_2に分解して，次のようなhorn set S_1，S_2を作る。

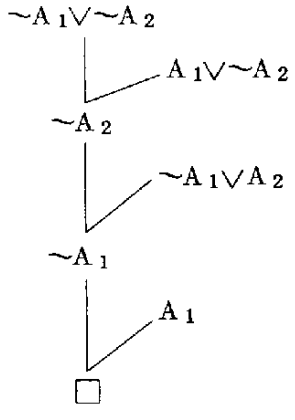
$$S_1 \triangleq \{A_1\} \cup \{\sim A_1 \vee A_2, A_1 \vee \sim A_2, \sim A_1 \vee \sim A_2\}$$
$$S_2 \triangleq \{A_2\} \cup \{\sim A_1 \vee A_2, A_1 \vee \sim A_2, \sim A_1 \vee \sim A_2\}$$

場合分けの原理より

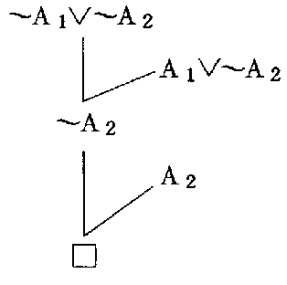
$$S_1 \vdash \square \ , \quad S_2 \vdash \square \quad \Leftrightarrow \quad S \vdash \square$$

であるから，S_1とS_2を夫々horn戦略を使って反証（refutation）すればSの反証ができたことになる。

実際これをＳＮＬにより実行すると，例えば



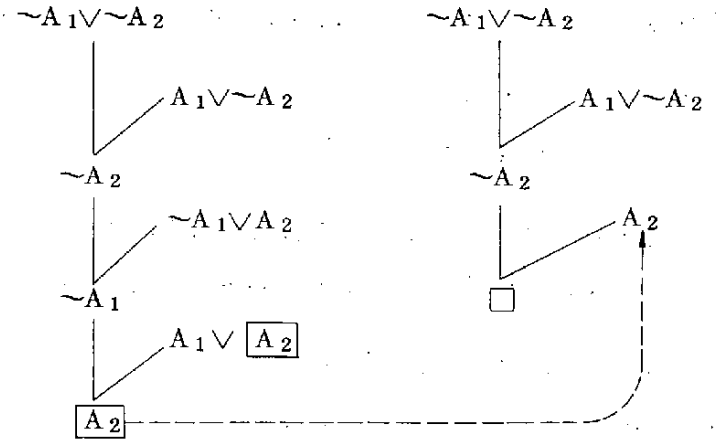
（S_1の反証図）



（S_2の反証図）

となる。今の場合は問題も簡単で，しかも命題論理の問題であったから，単にメノコでnon－horn節をhorn節に分解し，ＳＮＬを適用したが，もっと一般的な複雑な問題の場合には変数も入ってくるし，場合分けの数も増すので，もっと組織的な方法により場合分けの原理を適用しなければならない。そのため次のように考える。

S_1の反証図のunitリテラルA_1にA_2を置き戻す。するとS_1の反証図はA_2の導出図になるが，S_2の反証図のunitリテラルA_2はこの導出されたA_2であると見ることができる。S_1，S_2の反証図をこのように関連させてSの反証図とした時の図を次に示す。



（S_1の反証明を変形したA2の導出図）　　（S_2の反証図）

この図で四角枠で囲った部分（リテラル）は結論として導びき出されるまでは全く証明に関わっていないが，場合分けの情報を保持する役目を果している。

さて以上の例題を検討することにより，一般的な反証を場合分けの導入によりＳＮＬ反証の繰り返しに帰着させるためには次のようにすればよいことが分かる。

(a)　non－horn節を含む充足不能な節集合Ｓが与えられている。簡単のためＳ中の負節は唯一つＣとする。

(b)　Ｓ中のnon－horn節の余計な正リテラルを四角枠で囲う。四角枠で回われたリテラルを＜閉じたリテラル＞と呼び，残りの枠のないリテラルを＜開いたリテラル＞と呼ぶ。

(c)　閉じたリテラルに通し番号（場合分けの順番）を添る。ここまでの処理でできた節集合をS′とおく。S′は閉じたリテラルをすべて除去した時，horn setであり且つ充足不能である。

(d)　S_1≙S′としてｉ＝１から始める。S_1の唯一の負節Ｃをtop－clauseとするＳＮＬ反証を試みる。但し閉じたリテラルは導出に参加しない。しかしunifierは作用させる。

(d1)　Siの唯一の負節Ｃをtop－clauseとするＳＮＬ反証により，□（空節）が導びかれば反証は成功である。そうでない時は閉じたリテラルのみから成る節が導出される。そこでこの節の中で最小の添字を持つ閉リテラルを適当に選び，四角枠をはずす。（例えば，導出されたものが $\boxed{A}\boxed{B}\boxed{C}\boxed{D}$（2 2 4 3）であったとする。ここで $\boxed{A}$（2）を選ぶと結果は $A\boxed{B}\boxed{C}\boxed{A}$（2 4 3）となる。）

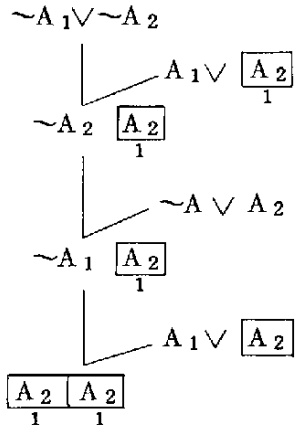
次にこの枠がはずされたリテラルと同じものが残りの閉じたリテラルにある場合はそれを削除（merge）する。（$A\boxed{B}\boxed{C}\boxed{A}$（2 4 3）は$A\boxed{B}\boxed{C}$（2 4）になる）導出された節に変数が含まれている時はfactoringを施こしてからこの種の削除を行う。

(d2)　(d1)で得られた新しい節をSiに加えてSi+1とする。

バックトラックを適当な所まで行って(d1)に戻る。

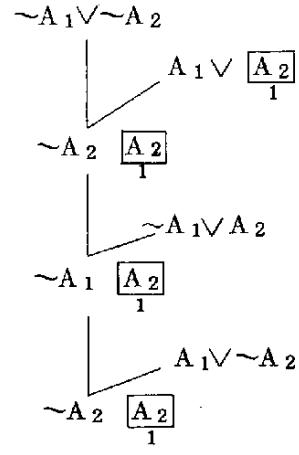
このアルゴリズムの完全性はS'の持つ閉じたリテアルの総数に関する数学的帰納法により容易に証明できる。このアルゴリズムをＳＮＬ＋ＣＡと呼ぶ。

ここで問題になるのは(d2)のバックトラックする地点である。(d1)で今Ⓐが開いてＡになったとする。バックトラックはこのⒶが証明に加わっていない地点，即ち side clause にⒶが初めて導入された直前の位置まで戻ることが望ましい。そうでないと無限ループに入る恐れがあるからである。例で説明する。Ｓ≒{～A_1∨～A_2, A_1∨～A_2, ～A_1∨A_2, A_1∨A_2}として，枠付けによりＳをhorn set S_1≒{～A_1∨～A_2, A_1∨～A_2, ～A_1∨A_2, A_1∨ $\boxed{A_2}_1$}にする。最初のＳＮＬが次のようなものであるとしよう。



（S_1に対するＳＮＬ，（ｄ１）の所まで）

次に(d2)に移り $\boxed{A_2}_1\ \boxed{A_2}_1$ のA_2を選んで枠を開き，A_2についてmergeすると結局A_2が得られる。A_2をS_1に加えた節集合をS_2とする。次にもし最も近い選択点にバックトラックすると，２回目のＳＮＬは，１回目のＳＮＬと２段目まで同じで，そこでA_1∨$\boxed{A_2}$の代りにA_1∨～A_2を選ぶことになる。



（S_2に対するＳＮＬ，直前の選択肢までバックトラックした場合）

すると3段目の導出は$\sim A_2$ $\boxed{A_2}_1$ とA_2又は$\sim A_1 \vee A_2$との間で行なわれるが，いずれも無限ループに入ってしまう。

次にＳＰＵの繰り返しで一般節に対する反証を行う方法を考えよう。やり方は前に述べた(a)～(c)まで同じで，(d)以下が異なる。

(d′) $S_1 = S'$とし，ｉ＝１から始める。Ｓｉに対するＳＰＵ反証を試みる。但し閉じたリテラルは導出に参加させない。しかしunifierは作用させる。

(d1′) Ｓｉに対するＳＰＵ反証を行う□（空節）が得られれば反証成功。もし閉じたリテラルのみから成る節が導出されたら，(d1)と同じく，最小添字を持つリテラルの枠を取り，factoringを行ってから，このリテラルに関するmergeを行う。

(d2′) (d1′)で得られた新しい節をＳｉに加えてSi+1とし(d1′)に戻る。

このアルゴリズムの完全性もS′中の閉じたリテラルの総数に関する数学的帰納法により容易に証明される。このアルゴリズムをＳＰＵ＋ＣＡと呼ぶことにする。

ＳＰＵ＋ＣＡは実はpositive hyper resolutionに非常に近い。ＳＰＵ＋ＣＡに於ける１回のＳＰＵ反証が１つのＰＩ－clashに対応する。違いはＳＰＵ＋ＣＡに於ける導出すべき正リテラルの選択が予め添えられた番号によるのに対し，ＰＩ－Clashに於ける正リテラルの選択が予め定められた述語記号間の順序によるという点である。

通常の無制限の導出法に場合分けを徹底して取り入れることを考える。充足不能な節集合Ｓのすべてのリテラルを枠付けして通し番号を添える。導出は，節中の最も小さい番号を持つリテラルを開いて行う。

この反証法は完全であってlock resolutionと全く一致する。逆に言うとlock resolutionは場合分けのみによる推論であると言える。

以上見て来たように場合分けの原理は導出法の分野で重要な役割りを果し，ある導出法に場合分けを取り入れてもその導出法の完全性を損なうことがないと考えられる。しかし機械的に場合分けを行う限り場合分けの効果は薄いと見るべきである。ＳＮＬ＋ＣＡに於いても慎重に場合分けの番号を添らないと無限ループに入る可能性がある。

（電総研　パターン情報部
推論機構研究室
佐　藤　泰　介）

【参考文献】

〔中島 80〕 中島秀之："Parallel PROLOG，第２１回プログラミングシンポジウム，1980

〔田村 80〕 田村浩一郎他："述語論理型プログラムの分散処理アーキテクチュアによる実現"，情報処理学会，記号処理研究会１１－１，１９８０

〔Kowlski 79〕 Kowalski，R．A．："Algorithm＝Logic＋Control"，CACM Vol．22 №7，１９７９．

〔渕 77〕 渕 一博："述語論理的プログラミング－EPILOGの提案－"，情報処理学会，記号処理研究会１－２，１９７７．

〔Kuehner 70〕 Kuehner，D："Some special purpose resolution system，" Machine Intelligence Vol．7，１９７０．

〔佐藤 80〕 佐藤泰介，松本裕次：Horn set と証明木の記述，情報処理学会第２１回全国大会，１９８０．

〔 3 〕 並列定理証明機の一構成法

1. はじめに

一階述語論理の証明系は，その応用上も重要である。代表的な証明系の１つである分解証明法が〔 Robinson 65 〕によって提案されて以来，様々な戦略（strategy)が報告されているが〔 Chang 73 〕〔 Kowalski 79 〕，未だ満足のいくものは得られていない。問題解決や定理証明に一階述語論理を用いる場合の最大の問題点はその効率である。効率改善には主に２通りの方向があると考えられる。１つは新たなる戦略の開発である。しかし，対象が一般的な論理式の集合である以上，一般的な戦略による効率の限界は余り望むべくもなく，発見的（ heuristic ）な手法，しかも問題に即した制御が必要となるであろう。第２番目は，分散処理等の並列計算機構によって計算時間を吸収する方法である。本稿ではこのような分散処理系による定理証明法（定理証明機）の構造について考察する。

著者らは定理証明の制御に用いる為に証明グラフ及び証明経路式を定義した〔松本 80 〕が，ここでは証明グラフの各節点を１つの処理装置（ processor)で置き換えた形の証明機について考える。ただし一階述語論理の節形式のうち，扱うのはHorn節だけであるが，この立場からはPROLOGの並列実行マシンと見なす事もできる。勿論，ここで提案する証明機には節や句の並び方による計算の制御やカットオペレータなどは存在しない。

次節で証明グラフとそれに付随する定義や定理について述べ，次々節で定理証明機の構成法について概観する。ここでわかるように本証明機はハードウエアによって実現するには未だ不充分である。最後の節では実現のために触れる。

2. 証明グラフ

2.1 諸 定 義

用語等は主に〔 Chang 73 〕〔 Kowalski 79 〕に従う。本節では最小限の定義を述べるに留める。

〔定義１〕 項（ term)とは変数，定数もしくは f（ t_1，… ， t_n) なる式である。ここに f は n 項関数， t_1，…， t_n は項である。

〔定義２〕 節 (clause)とは次のような論理式である。

B_1 ， B_2 ，…… ， $B_m \leftarrow A_1$ ， …… ， A_n ($m \geq 0$, $n \geq 0$)，ここに A_1，… ， A_n，B_1，… ， B_m は原子式*(atomic formula)である。特に，$m = 0$ または１の時，上の節を（ n 次の ）Horn節と呼ぶ。

以下，論理式は節形式で与えられ，定理証明とは，与えられた節の集合から矛盾を導く過

* Pをn項述語，t_1，…，t_nを項とすると，P（t_1，… ，t_n）が原子式である。

程とする。節の形によって次のように呼び方を変える事がある。

1) $m \geq 1$, $n = 0$ の節を表明文 (assertion) と呼ぶ。

2) $m = 0$, $n \geq 1$ の節を目的文 (goal statement) と呼ぶ。

3) $m \geq 1$, $n \geq 1$ の節を手続文 (procedure statement) と呼ぶ。

〔定義3〕 証明グラフは次のような五つ組 (N , E , OL , IL, □) によって定義される。

N : 節点 (node) の集合。節点には原子式に対応するものと，2次以上のHorn節1つ1つについてＡＮＤ節点と呼ばれるものが存在する。

E : 枝 (edge)の集合。枝は節形式の implication edgeに対応する。但し，2次以上のHorn節はＡＮＤ節点を介する。すなわちn次のHorn節 $B \leftarrow A_1, \cdots, A_n$ には $(A_1, AND_i) \cdots\cdots (A_n, AND_i)$, (AND_i, B) なるn＋1本の枝が対応する。ここに AND_i はこのHorn節に対応するＡＮＤ節点である。iは識別子。

ＯＬ : 枝の出力部*に与えられるラベルを表わす。形式的にはＥから原子式の引数を表わす tupleへの写像である。ただし (AND_i, A), (ϕ, A) なる枝には値が与えられない。一般に，(A, B) なる枝に対しては述語Ａの引数を与える tupleを値とする。

ＩＬ : 枝の入力部*に与えられるラベルを表わす。形式的にはＥから原子式の引数を表わす tupleへの写像である。ただし (A, AND_i), (A, □)なる枝には値が与えられない。一般に，(Ａ , Ｂ) なる枝に対しては述語Ｂの引数を与える tupleを値とする。

□ : 空節を表かす節点。グラフ上では目的文の出力部の節点として用いられる。

〔例1〕 次のようなHorn節の集合を考える。

Ｃ 1　Human (Turing)←

Ｃ 2　Human (Socrates)←

Ｃ 3　Greek (Socrates)←

Ｃ 4　Fallible (x)←Human (x)

Ｃ 5　←Fallible(x) , Greek(x)

図1にこれらのHorn節が表わす証明グラフを示す。Ｃ1，Ｃ，Ｃ2は並列枝 (parallel edge) を表わすので図ではひとまとめにした。枝に付けられたアルファベットａ，ｂ，ｃ，ｄ，ｅ，ｆは枝の名前である。例えば枝ｂについて見ると，ＯＬ(b)＝(x)，ＩＬ(b)＝(x)である。ただし，図では異なる節が同一変数を共有しないように，例えば，Ｃ４，Ｃ５についてＣ５の変数ｘをｙに名称変更 (renaming) した。∧はＡＮＤ節点を表わす。

---

* 枝には一般に2つのラベルが与えられる。有向枝の始節点側を出力部，終節点側を入力部と呼ぶ。

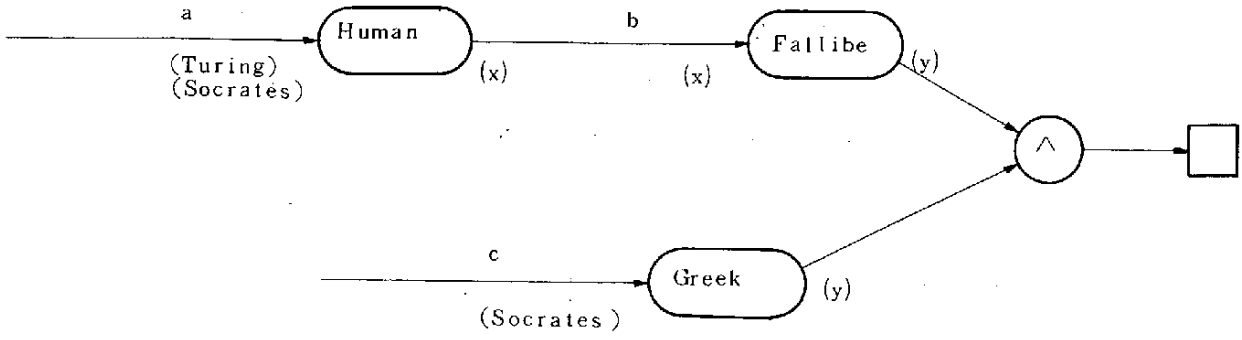


図1. 例1の証明グラフ

〔例2〕 次のHorn節集合を考える。

C 1 On(A , B) ←

C 2 On(B , C) ←

C 3 Green (A) ←

C 4 Blue (C) ←

C 5 ← Green (x), Blue(x)

C 6 Green (y) ← On(z , y), Green(z)

これらのHorn節が表わす証明グラフを図2に示す。小文字のアルファベットは枝の名前である。

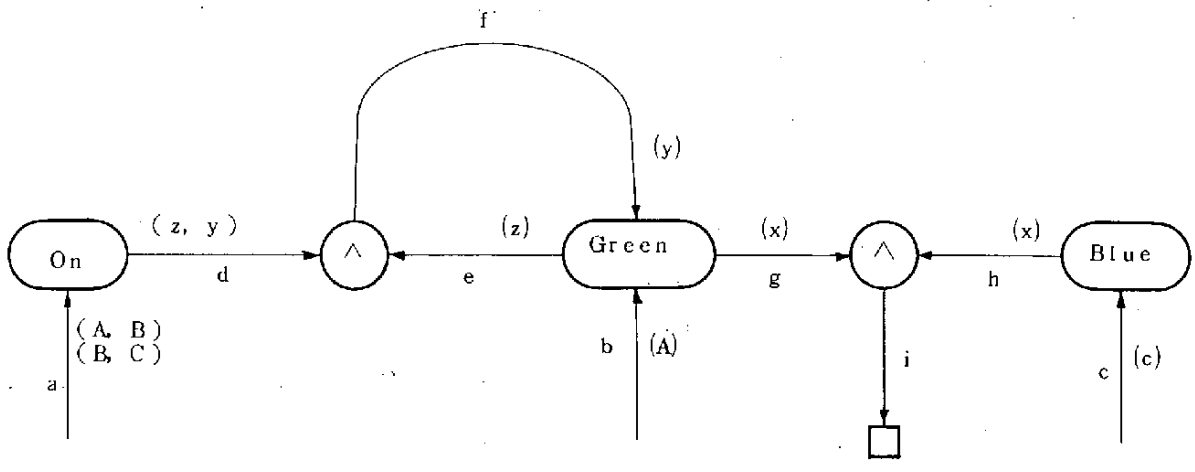


図2. 例2の証明グラフ

2.2 証明グラフによる定理証明とその完全性

著者らは証明グラフを用いた定理証明の制御の為に証明経路式を定義した〔松本 80〕。ここでは証明経路式については詳しく述べないが，証明グラフ上での証明過程は表明文（出

力部が空の枝）から目的文（入力部が空節点□の枝）への経路として表わされ，経路式はそれを記述するものである。統一化（unification）は節点の入，出力部のラベルの統一化に対応する。証明は証明経路式を用いて行うが空節点□から辿るTop－down法，表明文から出発するBottom－up 法など自由である。ただし，証明が完了した時の経路式の行程はpossitive hyper resolutionに対応し，次のように完全性が成立する。

〔定理1〕 充足不能なHorn節集合から得られる証明グラフの証明経路式には，空節を導出する導出木に対応する経路が必ず含まれている。

（略証） possitive hyper resolutionによって得られる導出木に相当する経路は証明経路式に含まれ，possitive hyper resolutionは完全（complete）である。したがって，充足不能なHorn節集合からは必ず空節を導出する導出木が得られる。

例えば，例2の証明グラフに対応するHorn節集合は充足不能であり，導出木としては図3に掲げるものが存在するが，空節点□からTop－downに辿った経路式にもこれに対応するものが存在する。

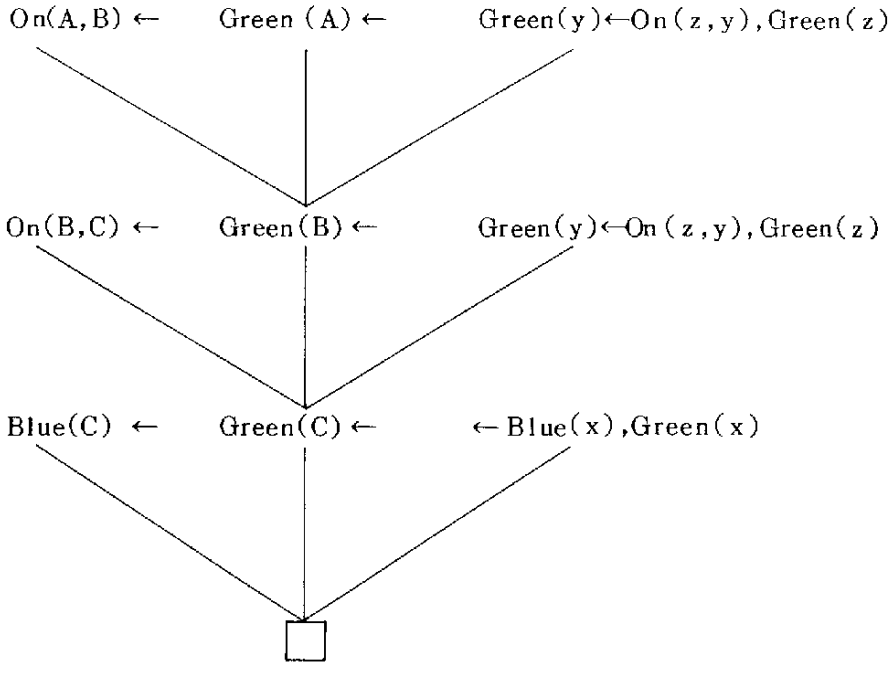


図3 例2の導出木

3. 並列定理証明機

本節では，前節で定義した証明グラフをハードウエア実現するための試みについて述べる。詳細については検討の余地が多く残されており，それらについては次節で言及する。

3.1 基本構成

基本的には，証明グラフをそのままの形で分散処理する。各節点が1つの処理装置（processor）で置き換えられ，汎用性を持たせるため各枝は通信網の中に仮想的に作られる事になる。ただし，ここでは通信網の形態は具体的には考えず，全体としては大まかに図4のようなハードウエア構成を考える。ここにＰＥは最小処理単位（processing element）である。

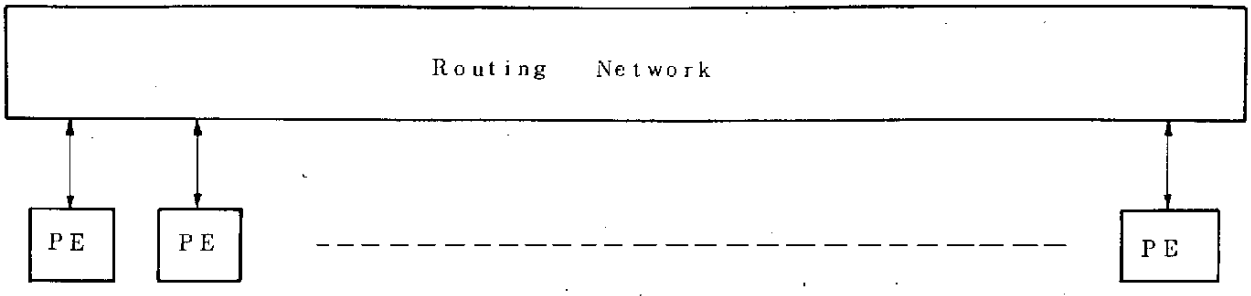


図4. 基本構成

処理装置は主に3種に大別して考える。手続型機（procedural processing element PPE），述語機（processing element for predicates，ＰＥＰ），ＡＮＤ機（processing element for ＡＮＤ－nodes，ＰＥＡ）である。手続型機は和，積の計算など，述語論理上では1つまたはそれ以上の述語で定義されるが，一般に無限個の真集合を持ち，処理装置上で手続的に定義するのが好ましい述語の働きをする。述語機及びＡＮＤ機はそれぞれ手続型以外の述語及びＡＮＤ節点に対応するが，統一化や要求（request）* に対する応答（response）* などの通信をやり取りする。

全体としての動作はここではTop－down形式について考察する。したがって，目的文の出力部である空節点が要求（request）を発する事によって動作を開始する。次に各処理装置の動作について説明し，最後に例によってその動作の概要を見る。

3.2 手続型機（Procedural processing element PPE）

和や積など一般に無限個の真集合を持つ述語（例えばTimes（ｘ，ｙ，ｚ）などｘ＋ｙ＝ｚの時のみ真）に要求が発せられた場合，2つ以上の変数が例示化（instantiation）されていない時には，それを満す変数への割当てが無限個存在し，応答の候補となるtupleが決定できない。したがって，このような処理装置の応答は，常に，例示化されていない変数が1つ以下の要求に対するものに制限する方がよい。これはPROLOGのプログラミングの際に

---

* 1つの枝について見ると，枝の始点側の節点を下位節点，終点側の方の節点を上位節点と呼ぶ。要求は上位から下位へ，応答は下位から上位の節点へ発せられる。

も問題になる事であり，２つ以上の変数が例示化されないままこのような述語を起動するようなプログラムは効率的には失格である。本稿では，手続型機の内部構造には触れず，一台の処理装置とみなし，例示化されていない変数の数が１以下の要求に対してのみ応答を行うと仮定する。

3.3 述語機(processing element for predicates,PEP)

証明グラフ上で述語を表わす節点に対応する処理装置の動作について考察する。この処理装置はその述語を満足させる項の組を格納する真値表[*1](On－set table)と，上位節点から受けた要求を記録するための要求表(request table)を持つ。この処理装置に必要な動作は表の検索と項の組の統一化等である。次に，上位及び下位節点からの通信によってＰＥＰがどのような動作をするか，その概要を示そう。要求及び応答はその述語のもつ引数の組の形（ｎ項述語の場合はｎ組）で与えられ，要求については要求元の処理装置名と共に送られるとする。[*2]

１） 上位節点からの要求を受付けると，

STEP1： 同一の要求もしくはその要求が例示化となるような要求が既に同一節点から要求されている場合には,その要求を棄却，他の節点から上のような要求が要求表に既に記入されている場合には，その要求を節点　となる。
新しく要求表に記入するが，このときは下のSTEP3を省略する。
それ以外の場合には，その要求を節点名と共に要求表に記入する。

STEP2： 真値表内の項の組の中で，要求と統一化可能なものとの統一結果を全て要求先の節点に応答する。[*3]　この時，統一結果の中に要求の内容と全く同一のものが存在すればSTEP3を省略。

STEP3： 全ての下位節点に同じ要求を発する。[*4]

２） 下位節点からの応答を受付けると，

STEP1： 同一の応答もしくはその応答が例示化となるような項の組が既に真値表に記入されていれば応答を棄却し，そうでない場合はその応答を真値表に記入する。

STEP2： 要求表の中でその応答と統一化可能なものが存在すれば統一結果を要求先の上位節点に応答する。

＊１　以下では，当該記述に対する表明文の内容は処理装置の動作開始前に，真値表に記入されていると仮定する。
＊２　述語によっては，手段型機と同じように例示化されていない変数の数が１以下の要求に対してのみ応答を行うと仮定してもよい。
＊３　応答先の節点がＰＥＰＥＰの場合には，その枝のＯＬの値と得られた統一結果との間に必要な置換(substitution)をその枝のＩＬの値に施したものを応答の値とする。
＊４　要求を送る先の節点がＰＥＰの場合には，その枝のＩＬの値と要求との統一化に必要な置換をその枝のOLの値に施したものを要求の値とする。

3.4 ＡＮＤ機(processing element for AND－node , PEA)

証明グラフ上でＡＮＤ節点に対応する処理装置の動作について考察する。これは２次以上のHorn節に対応し，要求表のみを持つが，ｎ次のHorn節に対応するＡＮＤ機はｎ＋１列の要求表を持ち，それぞれが下位の節点及び元の要求に対応する。すなわち要求表は図５のように行列形をなし，上位（図では節点Ｂ）からの要求により原則的に１つの行が作られる。行の各項目には各下位節点への要求が記入され，下位節点からの応答により統一化された項

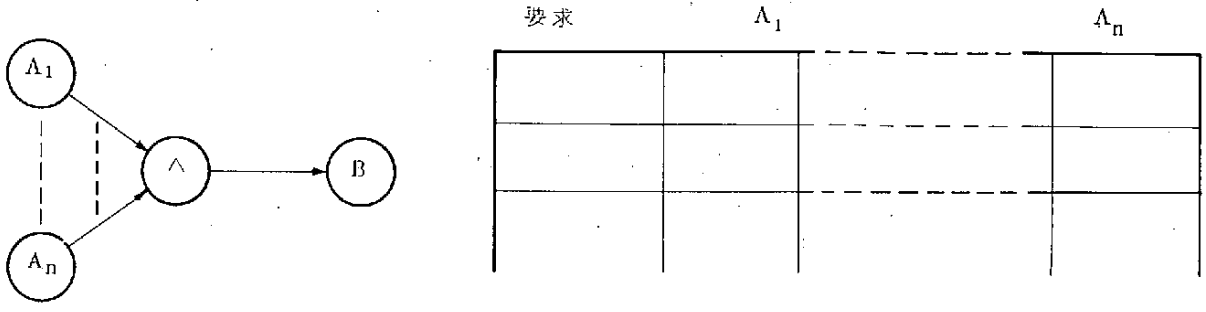


図5. ＡＮＤ節点とその要求表

目の記入や，それに伴う他の節点への新たな要求を作成する。行の各項目には要求が統一化済かどうか記入する欄が設けられており，一行の全ての項目が統一化されると，上位節点への応答が作られる。動作の概要を次に示す。

１） 上位節点からの要求を受付けると，

STEP1： 同一の要求が既に要求欄に記入されていれば，その要求を棄却する。

STEP2： 各下位節点へそれぞれの要求を行う。[*1]

２） 下位節点からの応答を受付けると，

SPEP1： 要求表のその節点に対応する列の中に応答と統一化のものが存在すれば，[*2]

ａ）応答と要求が同一の時はその項目に統一化済である事を記入する。

ｂ）応答が要求の例である時は新たに行を設けその応答に対応する項目に統一化済を記入する。他の項目については統一化に要した置換をもとの行の各項目に施したものとする。この時例示化された項目に対してはその項目を新たな要求として対応する節点に送る。

ｃ）応答の例になっている項目が存在し，統一化済でなければ統一化済を記入する。

＊１ 各下位節点への要求は，上位からの要求とその板のＩＬの値との統一化に必要な置換を各下位節点への板のＯＬの値に施したものである。

＊２ ＡＮＤ機はTop－downに動作するので，応答と統一化な要求は必ず存在する。

STEP2： 一行の全ての項目が統一化済になれば，各項目の統一化に要した置換をその行の要求に施して上位節点へ応答として送り，その行を要求表より消去する。

3.5 例

階乗の計算を例にとって，並列定理証明機の動作を眺めてみよう。階乗Fact（ x， y ）（ x ！＝ y の時に真）は次のように定義される。

C 1　Fact（ 0， 1 ）←

C 2　Fact（ n， N ）←Fact(m， M)， Times（ n， M， N ）， Sum(m, 1, n)

ここにTimes（ x, y, z)は x y ＝ z の時に真となり， Sum(x, y, z)は x + y ＝ z の時に真となる。

例えば２！を求めようとすると，目的文として，

C 3　←Fact（ 2, x ）

を上の２つの節に加えればよい。証明グラフは図６のようになる。ここにTimes，Sumは手続型機，Factは述語機であるが，ここではFactは手続型機と同様に例示化されていない変数の数が１以下の要求だけを受け付けると仮定する。FactとＡＮＤ節点の初期状態を図７に，操作開始後の処理装置の動作の様子を図８に示す。①…⑬は解へ到るまでの通信の順序である。勿論，これ以外の動作も行なわれるが，余分な部分は省略した。

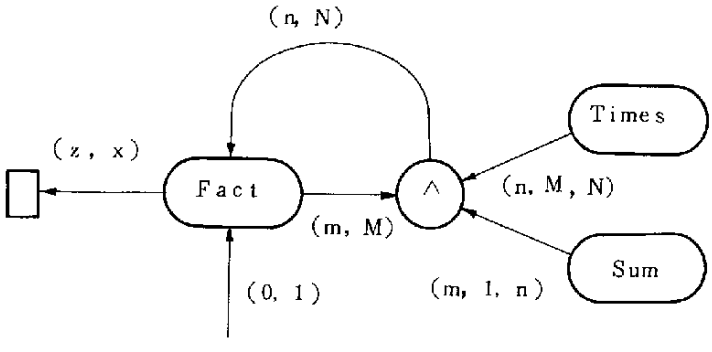


図６.　階乗計算の証明グラフ

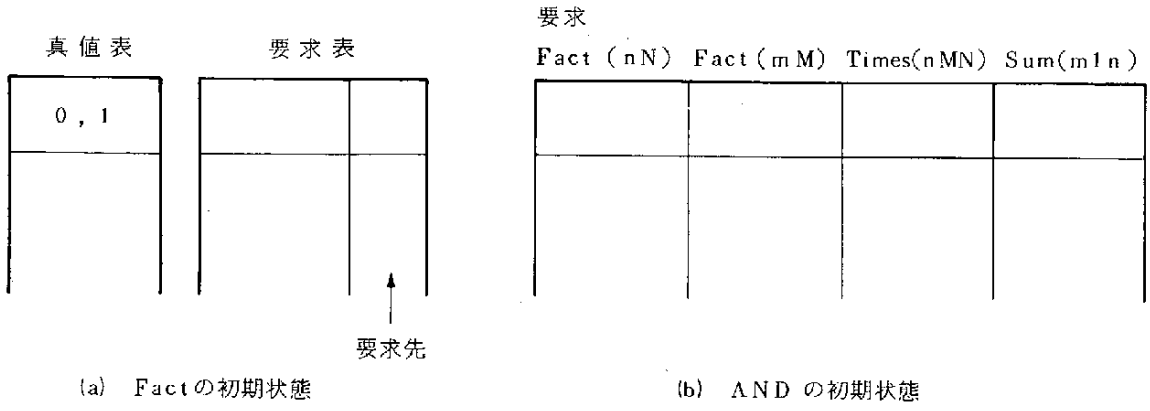


図 7　処理装置の初期状態

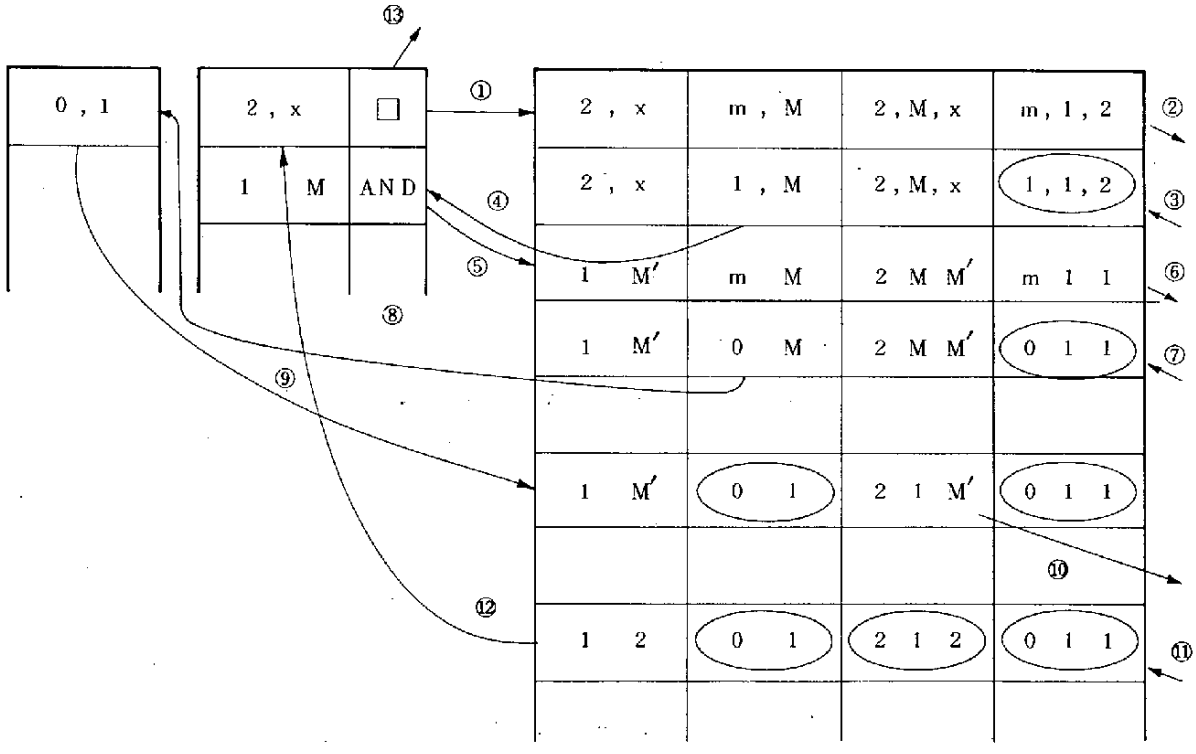


図 8　処理装置の動作

4. あとがき

並列定理証明機について概説した。本節では残された問題や，ハードウエア実現をするに当って解決しなければならない点について述べる。第一の危惧は真値表や要求表の容量増大である。特に真値表は時間と共に増大し，処理時間に直接的な影響を与える。但し，これは他の分野証明法にも共通の問題で，分解子（resolvent）の数を減らすための様々な努力がなされている。これらの戦略の考え方を並列定理証明機に取り入れる事は興味深い問題である。また直値表の増大や，通信の量を軽減するため要求や応答を制限する事も考えられるが，完全性との兼ね合いを常に注意する必要がある。

ハードウエアとして実現するための問題としては，まず routing network の決定がある。また各処理装置の入力バッファや通信の制御など，実現までに解決すべき問題は多い。

また扱う論理式の集合を非Horn節へ拡張する事も重要であるが，証明グラフを分解してHorn節に帰着させる方法やOR節点を設ける方法など現在考察中であるが，PROLOGなどからもわかるように，Horn節だけでも充分であるとも考えられる。これらについて詳細は別の機会に譲る。

電総研　パターン情報部
推論機構研究室
松　本　裕　治

〔参考文献〕

〔Robinson 65〕 Robinson，J. A.，"A Machine Oriented Logic Based on Resolntion Principle," J. ACM. 12，PP. 23 − 41. Jan. 1965.

〔Chang 73〕 Chang，C. L. and Lee，C. T.："Symbolic Logic and Mechanical Theorem Proving", Academic Press，1973.

〔Kowalski 74〕 Kowalski，R.，"Predicate Logic as Programming Language，" IFIP 74, PP. 569 − 574, 1974

〔Van Emden 77〕 Van Emden，M. H.，"Programming with Resolution Logic，" Machine Intelligence 8，Edinbnrgh Univ. Press，pp. 266 − 299. 1977.

〔Kowalski 79〕 Kowalski，R.：Logic for Problem Solving，North Holland，1979.

〔松本 80〕松本裕治，佐藤泰介，"証明経路式による定理証明の制御" 情報処理学会人工知能と対話技法研究会資料 17 − 3. Sept. 1980.

〔松本 81〕松本裕治，"定理証明機の一構成法について，"電子通信学会昭和56年度総合全国大会，1288. Apr. 1981.

〔４〕 代数的仕様に基づく段階的詳細化

－ HISPを例として －

1. まえがき

代数的仕様のプログラミングへの応用に関する基本的考え方などについては〔１〕を参照されたい。

抽象データ型の形式的仕様記述法の研究から出発し，一般的な仕様記述法に発展しつつある代数的方法の特徴は次の２点に要約出来よう。

(1) 比較的簡単な規約に基づいて，任意の抽象レベルで厳密な記述が出来る。このことが抽象データ型の仕様記述法としての最大の利点でもあり，仕様／プログラムの段階的詳細化のために有効な特質となる。

(2) 代数的方法における意味記述の担い手である等式は書換規則とみなすことが出来る。この等式の“意味の二面性”が，プログラミングの全過程（仕様記述／設計からコーディング／実行まで）を等式に基づく意味記述で行なう可能性を開く。

代数的方法は上記のような利点を有し，仕様記述を始め，設計，（狭義の）プログラミング，検証など多くの局面において有効な手法を提供しそうである。

本稿では，仕様／プログラムの構造化の基本的手法である段階的詳細化を代数的方法に基づいて試みた結果について述べる。記述には，現在筆者らが研究開発中のＨＩＳＰ言語を用いている。ＨＩＳＰについては参考文献〔１〕，〔２〕などを参照されたい。

2. 仕様／プログラムの詳細化[2]

この節では簡単なプログラミングの例題に対し，そのプログラム仕様をＨＩＳＰを用いて段階的に詳細化していく様子を示す。ＨＩＳＰにおいては，仕様とプログラムは同質のものとして捉えられるという考え方をしている。従って仕様からプログラムへの移行は連続的に進行し，その正しさも比較的容易に検証可能であろうというのが基本的なねらいである。

考える問題は次の“電報解析問題”である。

問題： 入力ファイルにいくつかの電文が入っている。各電文はいくつかの語からなり最後には区切り語“ＺＺＺＺ”がある。入力ファイルの終りは，区切り語“ＺＺＺＺ”だけから成る電報で示される。処理の目的は電文の解析である。各電文に対し，その電文が含む語数とその中で何語が12文字以上の長語であるかを示す報告書を作りたい。報告書の書式は次の通りとする。

```
TELEGRAMS ANALYSIS
TELEGRAM 1
  16 WORDS OF WHICH 3 OVERSIZE
TELEGRAM 2
  106 WORDS OF WHICH 14 OVERSIZE
TELEGRAM 3
  42 WORDS OF WHICH 2 OVERSIZE
     ...
     ...
END ANALYSIS
```

トップレベルの仕様はモジュールＴＡで示される。

```
TA ::  /* Telegram Analysis */
 create
  sort In, Out
  op   teleanal-of _ : In -> Out
 end
```

電文単位での仕様はモジュールＴＡＴで与えられる。ＴＡＴはモジュールＴＲとＲＧをサブ・モジュールとして参照し，ＴＡを詳細化（refine）して得られる。

```
TR ::  /* Telegram Reader */
 constr
  sub  BOOL, INT /* INTeger (built-in) */
  sort TelSeq /* TELegram SEQuence */
  op   is-1st-tel-empty? : TelSeq -> Bool
       word#of-1st-tel-of _ : TelSeq -> Int
       o-sized-word#of-1st-tel-of _
                            : TelSeq -> Int
       rem-of _ : TelSeq -> TelSeq
         /* remainder of */
 end

RG ::  /* Report Generator */
 constr
  sub  INT
  sort Report
  op   telegram-analysis-eol : -> Report
       _ telegram _ eol
         _ word-of-which _ oversized-eol
           : Report, Int, Int, Int -> Report
       _ end-analysis-eof : Report -> Report
    end
```

```
TAT ::  /* TeleAnal of level Telegram */
 refine  TA
  sub TR, RG, INT, BOOL
  op  _ : TelSeq -> In
      _ : Out -> Report
      _ _ #append-reports-of _ :
       Report, Int, TelSeq -> Report (hidden)
  eq  var ?rpt?:Report; ?ts?:TelSeq;
          ?n?:Int
    ( ?rpt? ?n? #append-reports-of ?ts?
      = if is-1st-tel-empty?( ?ts? )
        then ?rpt?
        else (
          ( ?rpt? telegram ?n? eol
            word#of-1st-tel-of ?ts?
            word-of-which
            o-sized-word#of-1st-tel-of ?ts?
            oversized-eol )   ?n? + 1
          #append-reports-of   rem-of ?ts? )
        fi )
    ( teleanal-of ?ts?
                = ( telegram-analysis-eol 1
                    #append-reports-of ?ts? )
                    end-analysis-eof )
      end
```

語レベルの仕様はモジュールＴＡＷで与えられる。ＴＡＷはＴＡＴのサブ・モジュールＴＲをモジュールＷＴＲですげ換えて得られる。ＷＴＲはモジュールＷＲをサブ・モジュールとして参照することによりＴＲを詳細化して得られる。

```
WR ::  /* Word Reader */
 constr
  sub  BOOL
  sort WordSeq
  op is-1st-word-eot? : WordSeq -> Bool
     is-1st-word-oversized? : WordSeq -> Bool
     rem-of _ : WordSeq -> WordSeq
 end

WTR :: /* Word Telegram Reader */
 refine  TR
  sub WR / INT, BOOL
  op  _ : WordSeq -> TelSeq
      word# : WordSeq -> Int (hidden)
      o-sized-word# : WordSeq -> Int (hidden)
      next-telseq-of
                : WordSeq -> WordSeq (hidden)
  eq  var  ?ws?:WordSeq
  ( is-1st-tel-empty?( ?ws? )
                = is-1st-word-eot?( ?ws? ) )
  ( word#( ?ws? )
          = if is-1st-word-eot?( ?ws? )
            then 0
            else 1 + word#( WR.rem-of ?ws? )
            fi )
```

```
  ( word#of-1st-tel-of ?ws? = word#( ?ws? ) )
  ( o-sized-word#( ?ws? )
    = if is-1st-word-eot?( ?ws? )
      then 0
      else if is-1st-word-oversized?( ?ws? )
           then 1 +
            o-sized-word#( WR.rem-of ?ws? )
           else
            o-sized-word#( WR.rem-of ?ws? )
           fi
      fi )
  ( o-sized-word#of-1st-tel-of ?ws?
                  = o-sized-word#( ?ws? ) )
  ( next-telseq-of ?ws?
    = if is-1st-word-eot?( ?ws? )
      then WR.rem-of ?ws?
      else next-telseq-of WR.rem-of ?ws? fi )
  ( rem-of ?ws? = next-telseq-of ?ws? )
end

TAW ::  /* TeleAnal of level Word */
   TAT (* TR <- WTR *)
```

文字レベルの仕様を得るためには，ＷＲ及びＲＧを文字入出モジュールＣＲ（character reader），ＣＷ（character writer）を用いて詳細化すればよい。詳細は略す。

3. 設計の詳細化[3][4]

この節ではＨＩＳＰで２½次元ディスプレイの設計仕様〔３〕を記述してみた結果を示す。例題の詳細については文献〔３〕を参照されたい。ＴＥＸＴ，ＦＩＧＵＲＥ以下の記述は省略している。

まず appearance，in の２つの関数の値を各 Displayable Object に対して定義することによりＤＩＳＰＬＡＹの仕様が与えられることが示される。

```
DISPLAYnull::
 constr
  sub   BOOL
  sort  DisplayableObject, Coordinate,
        Illumination
  op    appearance: DisplayableObject,
                    Coordinate -> Illumination
        in: DisplayableObject, Coordinate
                                 -> Bool
 end
```

つぎに Picture に対して appeurance，inの値が定められる。

```
DISPLAYpicture::
 refine DISPLAYnull
  sort   Picture, Contents, Bound, Trans
  op applyBound: Bound, Coordinate -> Bool
     applyTrans: Trans, Coordinate
                              -> Coordinate
     makePicture: Contents, Bound, Trans
                              -> Picture
     _: Picture -> DisplayableObject
     _: Contents -> DisplayableObject
  eq var ?cont? : Contents ;
         ?bound? : Bound ;
         ?trans? : Trans ;
         ?coord? : Coordinate
  ( appearance(makePicture(
      ?cont?, ?bound?, ?trans?), ?coord?)
    = appearance(?cont?,
                 applyTrans(?trans?, ?coord?)) )
  ( in(makePicture(?cont?, ?bound?, ?trans?),
       ?coord?)
    = applyBound(?bound?, ?coord?) )
 end
```

さらに contents に対して appearance , in の値が定められる。

```
DISPLAYpContents::
 refine DISPLAYpicture
  sort  Component
  op    empty$c: -> Contents
        addComponent: Contents, Component,
                      Coordinate -> Contents
        _: Component -> DisplayableObject
        minus: Coordinate, Coordinate
                              -> Coordinate
        combine: Illumination, Illumination
                              -> Illumination
  eq var ?cont? : Contents ;
         ?comp? : Component ;
         ?coord?, ?coord1? : Coordinate
  ( appearance(addComponent(
          ?cont?, ?comp?, ?coord1?), ?coord?)
    = if in(?comp?, minus(?coord?, ?coord1?))
      then if in(?cont?, ?coord?)
           then combine(appearance(?comp?,
                    minus(?coord?, ?coord1?)),
                  appearance(?cont?, ?coord?))
           else appearance(?comp?,
                  minus(?coord?, ?coord1?)) fi
      else appearance(?cont?, ?coord?) fi )
  ( in(empty$c, ?coord?) = false )
  ( in(addComponent(?cont?, ?comp?,
                    ?coord1?), ?coord?)
    = in(?comp?, minus(?coord?, ?coord1?))
      or in(?cont?, ?coord?)  )
 end
```

そしてcomponentとしてView, Text, Figureの3つがあることが示される。

```
DISPLAYpcComponent::
 refine DISPLAYpContents
  sub  TEXT:: create sort Text end,
       FIGURE:: create sort Figure end
  sort View
  op   _: View -> Component
       _: Text -> Component
       _: Figure -> Component
 end
```

最後にViewに対してappearance, inの値が定められる。

```
IDlist::
 constr  sub BOOL
  sort IdList, PictureId
  op   empty: -> IdList
       insert: PictureId, IdList -> IdList
       first: IdList -> PictureId
       null: IdList -> Bool
       equal: PictureId, PictureId -> Bool
  eq var ?p? : PictureId ;
         ?il? : IdList
     ( first(insert(?p?, ?il?)) = ?p? )
     ( null(empty) = true )
     ( null(insert(?p?, ?il?)) = false )
 end
```

```
DISPLAYpccView::
 refine DISPLAYpcComponent
  sub IDlist, BOOL
  op empty$v: -> View
     addPicture: View, Coordinate, PictureId,
                 Picture -> View
     findPicture: View, Coordinate -> IdList
     deletePicture: View, PictureId -> View
  eq var ?v? : View ;
         ?coord?, ?coord1? : Coordinate ;
         ?id?, ?id1? : PictureId ;
         ?p? : Picture
  ( appearance(addPicture(?v?, ?coord1?,
                          ?id?, ?p?), ?coord?)
    = if in(?p?, minus(?coord?, ?coord1?))
      then appearance(?p?, minus(?coord?,
                                 ?coord1?))
      else appearance(?v?, ?coord?) fi )
  ( in(empty$v, ?coord?) = false )
  ( in(addPicture(?v?, ?coord1?, ?id?, ?p?),
       ?coord?)
    = in(?p?, minus(?coord?, ?coord1?))
      or in(?v?, ?coord?) )
  ( findPicture(empty$v, ?coord?)
    = IDlist.empty )
  ( findPicture(addPicture(?v?, ?coord1?,
                           ?id?, ?p?), ?coord?)
    = if in(?p?, minus(?coord?, ?coord1?))
      then insert(?id?,
                  findPicture(?v?, ?coord?))
      else findPicture(?v?, ?coord?) fi )
  ( deletePicture(empty$v, ?id?) = empty$v )
  ( deletePicture(addPicture(?v?, ?coord?,
                             ?id1?, ?p?), ?id?)
    = if equal(?id?, ?id1?)
      then ?v?
      else addPicture(deletePicture(?v?,
             ?id?), ?coord?, ?id1?, ?p?) fi )
 end

DISPLAY::DISPLAYpccView
```

4. あとがき

ＨＩＳＰによる段階的詳細化の例を示した。むしろＨＩＳＰ記述法は固定化されているわけではなく（たとえば２節の問題に対し参考文献〔５〕では少し異った記述を試みている），今後言語の改訂も行ないつつ整備されるべきものである。

しかし現在までのＨＩＳＰの研究から得られた次のような見通しは有益と思われる。

(1) 代数的方法の基礎を成す多ソート代数は，プログラミングの構造化における有効なモデルを提供する。このモデルは特にプログラムを関数と考える関数的プログラミングになじみ易い。

(2) 意味記述の基礎として等式を採用し，これを書換規則と見なすことにより，仕様とプログラムの意味に対する簡明な基礎が与えられ，検証などの透明度向上に役立つ。この効果は将来書き換えを効率よく実現出来る書換機械が実現すればより顕著になろう。

（電総研　ソフトウェア部
言語処理研究室
二木厚吉）

【参考文献】

〔１〕 二木，代数的プログラミング。第５世代計算機調査報告書－資料４，５３－６８，１９８０－３，ＪＩＰＤＥＣ。

〔２〕 K. Futatsugi and K. OKada, Specification Writing as Constraction of Hierarchically Structured Clusters of Operators. Information Processing 80, 287−292, Oct. 1980.

〔３〕 J. Guttag and J. J. Horning, Formal Specification As a Design Tool. Proc. of 7th Annual ACM Sympo. on POPL, 251−261, Jan. 1980.

〔４〕 二木，設計ツールとしての階層的仕様。昭５６年度信学会全大，１９８１－４

〔５〕 K. OKada, K. Futatsugi, and K. Torii, Reliable Program Derivation in Functional Languages by Applying Jackson's Design Method. Proc. of FTCS−10, 91−96, Oct. 1980.

〔５〕 高速リスト処理アルゴリズム

1. はじめに

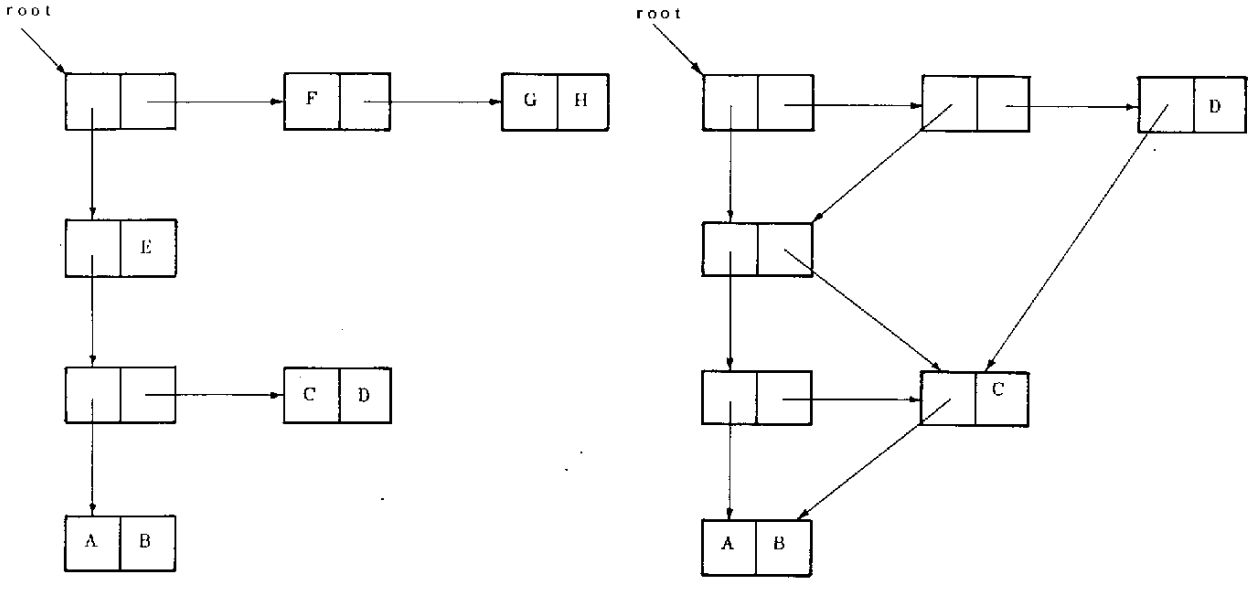
構造を持つデータを扱うリスト処理（例えばＬＩＳＰなど）では，処理は次の様に行なわれる。まず，処理系の外より構造データを読み込んで内部表現に変換し，次いで，その内部表現に対して，構造データ中の各要素の選択と結合の組合せを繰返すことによって構造データの変換を行ない，目的とするデータ構造を作成する。最後に，内部表現を外部表現に変換して出力することが行なわれる。

本稿では，内部表現に対する処理及びそれを構成する処理系における処理アルゴリズムの問題点と，それを打破する最近の研究成果について述べる。

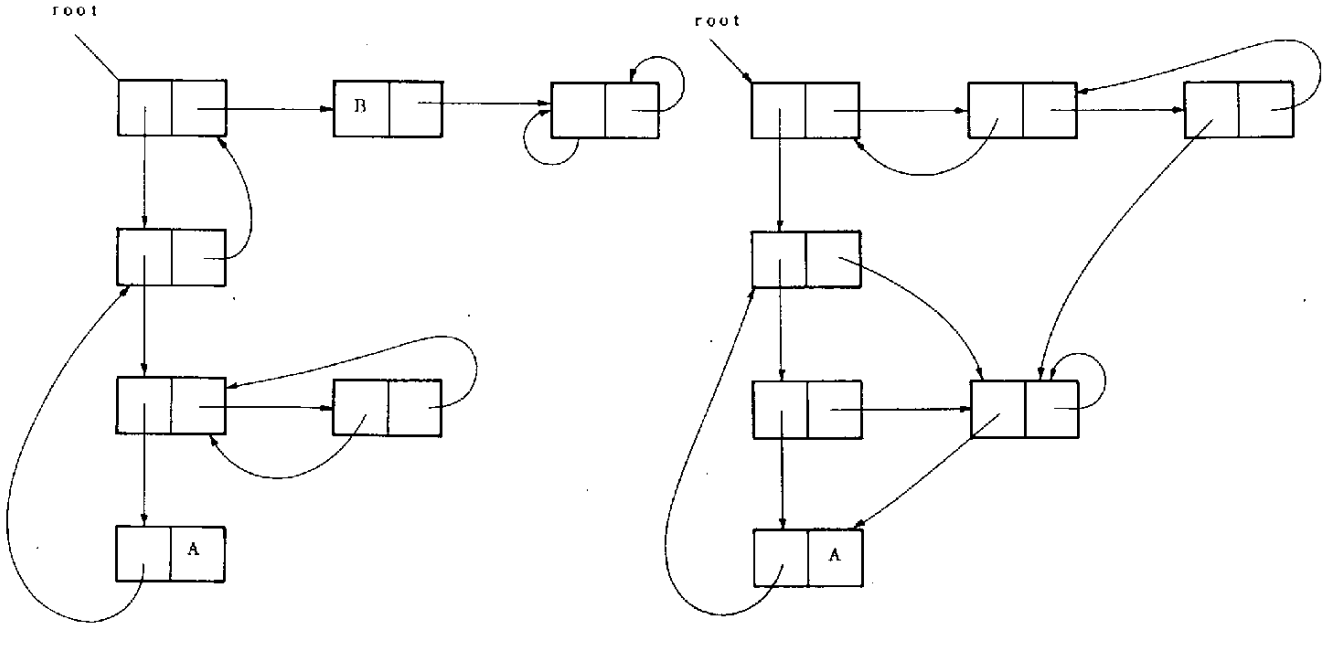
リスト処理においては，その処理対象とする内部表現はリスト構造又はリストと呼ばれ，木（ tree ），共有部分木を持つ木（ tree with shared subtree ），再入リスト（ reentrant list ），リスト（ list ）に分類される。例えば，ＬＩＳＰでは，それぞれは，図１に示す様に表現される。共有部分木を持つ木では，ある部分木は，２ケ所以上から指される共有ポインタと呼ぶポインタによって共有（ share ）される。再入リストでは，あるセルに含まれるポインタが，そのセルを含めて，ルート（ root ）からそのセルまでの道筋にある任意のセルを指す再入ポインタであることを許すが，部分木の共有は許さない構造である。リストは，共有ポインタ及び再入ポインタを持つことが可能な構造であり，それ故，任意の構造を表現できる最も自由な構造であると言える。

リスト処理，即ち，構造を持つ対象中からその構成要素の選択をして取り出し，又それらを結合する操作を行なうためには，構造の中を自由にたどる（ traverse ）ことが基本となる。図１(1)の木を見ればわかる様に，ルートから任意の葉（ leaf ）へ向っては，ポインタをたどることによって進むことは可能だが，それは逆に葉からルートへ向って進むことは不可能である。そのため，一般には，補助スタックを用意して，ルートから葉へ向って進む時には，その道筋のセルの番地を補助スタックにプッシュ（ push ）し，反対に，葉からルートへ向って戻る時には，補助スタックをポップ（pop)することにより，木の要素を自由にたどることを可能にしている。補助スタックを用いて木をたどる方法は，高速なたどりを実現できるという利点がある反面，補助スタック領域として，たどる対象とは別の領域が必要になることが最大の欠点となっている。例えば，ｎセルから成る任意の木を補助スタックを用いてたどる場合には，単純なアルゴリズムならば，深さｎの補助スタックを用意する必要がある。すなわち，処理する対象が占める記憶領域と等しい大きさの領域を補助スタックのために用意しなくてはならないことになる。

処理する対象が小さい場合には，補助スタックによる処理の利点である高速性を利用することが可能であるが，ＬＩＳＰシステムなどで不要になったセルを回収する回収再生システ



(1) (2)



(3) (4)

(1) 木， (2) 共有部分木を持つ木， (3) 再入リスト， (4) リスト

図 1. リスト構造

ム（ garbage collection， 以後G．Cと略す）では，このことは致命的な欠陥となって現われる。

ほとんどのG. C.で用いられるマーク・スキャン（ mark－scan ）法では，まず第1段階として，使用中の全てのセルをたどり，各セルのマークビット（ mark bit ）を1にするマーキング（ marking ）を行なう。次いで第2段階として，マークビットが0であるセル，すなわち現在使用中でないセルを全て回収し，自由リストを作成する。従って，G．Cの第1段階であるマーキングでは，充分な深さの補助スタックを用意しなくてはならない。ところが，G．Cは使用可能なセルがなくなり自由リストが空になった時に行われるのであるから，あらかじめ莫大な補助スタック領域を用意することにより，使用可能なセルの総量を減らすことは好ましくない。また一方で，もし補助スタックの深さが充分でない場合には，マーキングにおいて，補助スタックのあふれ（ overflow ）を生じ，システムは停止を余儀なくされることになる。

上記の様に，補助スタックは処理が高速に行なえる反面，致命的な欠陥を生じる欠点を有するため，これを克服する手法が開発されてきている。以下では，まず第2節で，補助スタックを用いずに木をたどる手法と種々のマーキング・アルゴリズムについて触れ，第3節では，補助スタックを使用せず，かつまたそれを使用する場合よりも高速な種々のコピー・アルゴリズムについて述べる。第4節では，木の高速汎用コピー・アルゴリズムを具体的に述べる。最後に第5節で，高速リスト処理の今後の方向を示す。

2. 固定作業領域アルゴリズム

プログラム変数を除いては，処理対象とは別に取られる補助スタックなどの補助作業領域を一切使用しないリスト処理アルゴリズムは，固定作業領域アルゴリズム（ constant 又は bounded workspace algorithm ）と総称される。

Schorr &Waite〔1〕は，任意のリストを補助スタックを用いずにマーキングする有名なアルゴリズムを示しているが，そこでは，各セルにマークビットの他に1ビットのタグビット（tag bit ）を仮定している。簡単のため，図1(a)の木をマーキングしている状態を図2に示す。図2では，root，car，cdr と前順序（ preorder ）で木をたどりつつ，1度アクセスしたセルのマークビットは初期状態の0から1に，また，carの指す先をマーキング中のセルのタグビットは1に，cdrの指す先をマーキング中のセルのタグビットは0になっている。この場合には，処理対象の中に rootへ戻る道筋を埋め込むことにより，固定作業領域アルゴリズムを実現している。図2に示したSchorr &Waiteのアルゴリズムで示した手法は，ポインタ反転法（ pointer reversal method ）と呼ばれる。これに対して，Lindstrom〔2〕は，ポインタ循環法（ link permutation method ）を発見している。図3にポインタ反転法とポインタ循環法におけるセル中のポインタの変化を示す。

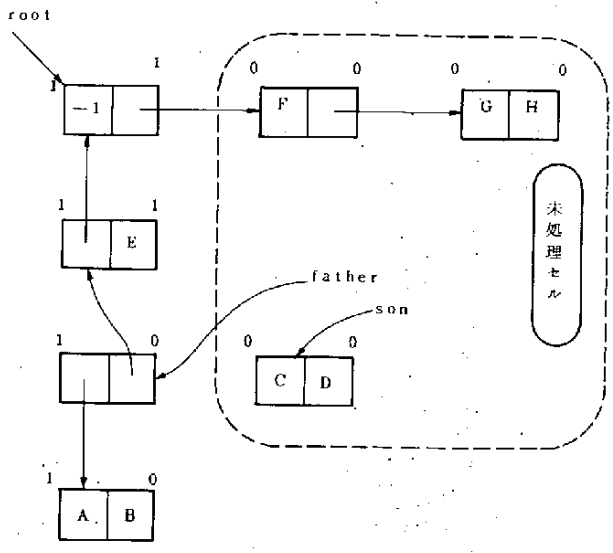


図 2.　Schorr & Waite のリスト・
マーキング・アルゴリズム

（各セルの左肩にマーク・ビット，右肩にタグビットを付す）

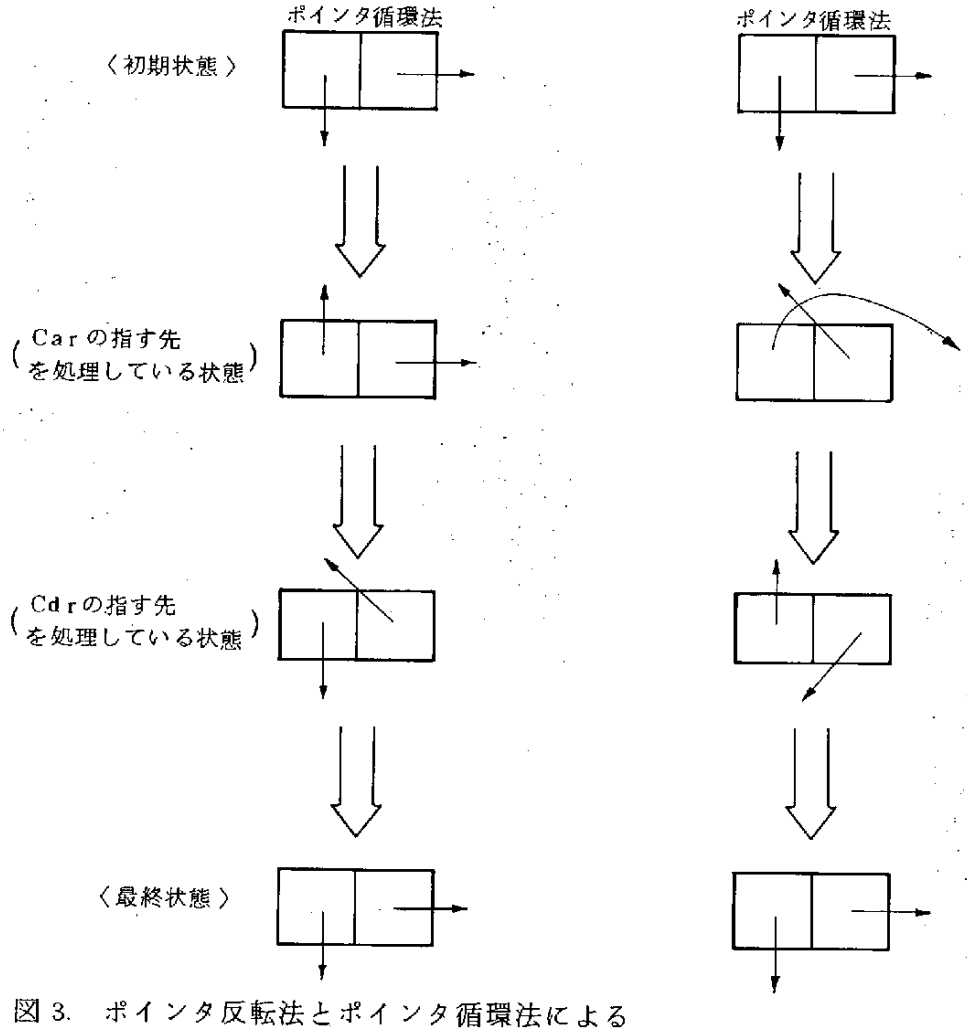


図 3.　ポインタ反転法とポインタ循環法による
セルの状態遷移

ポインタ循環法の特色は，図３に示す各状態の遷移が，全く同一の操作によって行なわれるという点にある。このため，Schorr & Waiteのアルゴリズムでは，木のマーキングの場合に，各セルにマークビットの他に１ビットのタグビットを必要としたが，Lindstromのポインタ循環法によれば，タグビットを必要とすることなくマークビットのみによって木をマーキングすることが可能となる。（図４を参照）

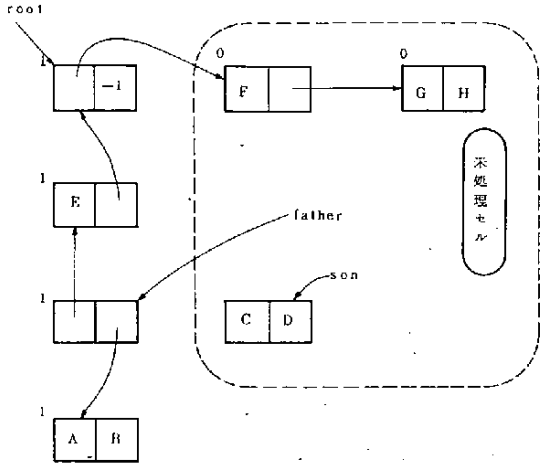


図４.　Lindstromの木のマーキング・アルゴリズム
（各セルの左肩にマークビットを付す）

次に，これを発展させてマーキングする対象を任意のリストとした場合には，非常に難かしいだけでなく，数学的にも興味のある問題となる。Lindstrom〔３〕は，ｎセルから成るリストに対してO(n log n)でマーキングを行なうアルゴリズムを示している。これに対して，長谷川〔４〕は，より高速なマークビットのみを用いるリスト・マーキング・アルゴリズムを求めている。アルゴリズムの詳細については，文献を参照されたい。

木やリストのマーキングに関しては，固定作業領域アルゴリズムは，補助スタックを用いるアルゴリズムよりも処理速度は遅い。そのため，従来は，補助スタックを用いるマーキングとSchorr & Waiteのリスト・マーキング・アルゴリズムを併用する方式を用いていた。そこでは，まず，補助スタックを用いてマーキングを行ない，スタックがあふれたならば，その先の葉までの部分をSchorr & Waiteのポインタ反転法を用いてマーキングするというやり方である。これに対しても，より効率の良いアルゴリズムが出現してきているが，詳細は参考文献〔４〕を参照されたい。

3. 高速コピーアルゴリズム

リスト処理を並列処理によって実現しようとする場合には，処理する対象のコピーを作成し，それらに関して同時に処理を実行することが欠かせない。

前節で述べたマーキング・アルゴリズムでは，補助スタックを用いる場合の方が，固定作業領域アルゴリズムよりも処理速度の点で優れていた。これに対して，本節で述べるコピー

アルゴリズムに関しては，補助スタックを用いる場合よりも，補助スタックを用いない固定作業領域アルゴリズムの方が，すべての場合により高速な処理を実現している。図5に，以下に述べる分類による高速コピーアルゴリズムの現状を示す。

コピーアルゴリズムは，コピーが作成される領域により3種類に分類される。まず，最も制限のない場合として，タグなし自由リストへのコピーがある。この場合には，コピーを構成するコピーセルは，タグなし自由リストから1セルづつ取り出され，また，コピーを行なう対象を構成する原セル（ original cell ）にもタグは付いていない。したがって，タグなし自由リストへのコピーアルゴリズムは，汎用アルゴリズムであると言える。次いで，より高

コピー領域 / コピー対象	連続領域	タグ付き自由リスト	タグなし自由リスト
木	clark〔5〕 長谷川〔7〕		Lindstrom〔2〕 Lee〔6〕 長谷川〔7〕
再入リスト	長谷川〔8〕	長谷川〔8〕	
リスト	Fisher〔9〕 Clark〔10〕 長谷川〔11〕	Lindstrom〔3〕 長谷川〔11〕	Robson〔12〕

図5. 高速コピーアルゴリズムの分類

速化を計るために，コピーする対象を構成するセルと，コピーセルを取り出す自由リストの各セルにそれぞれ1ビットのタグビットを付ける場合があり，タグ付き自由リストへのコピーと呼ばれる。この場合には，タグなし自由リストへのコピーの場合よりも，原セルとコピーセルの各々のタグビットの情報を利用できるため，より高速な処理が可能となる。最後に，更にコピーが作成される領域に制限を加えた連続領域へのコピーがある。そこでは，ある番地より始まる連続領域の記憶領域にコピーを作成する。この場合には，コピーする対象と作成中のコピーとが，連続領域にあるか否かにより容易に区別がつくためと，コピーする対象をたどった順序が連続領域に保存されるために，上記2つの場合よりも，より高速なコピーアルゴリズムを実現できる。

次に，コピーする対象から，木，再入りスト，リストの順にコピーアルゴリズムは分類され，この順に処理速度は遅くなる。

また，木のコピーの場合には，処理中に原木（ original tree ）を全く変更することなく1

パスでコピー木の作成が可能であるが，他の場合には，処理中に原構造の変更を伴なう1パスまたは2パスのアルゴリズムとなっている。

次節では，これらのコピーアルゴリズムの中から，木の汎用コピーアルゴリズム〔7〕について述べる。

4. 木の汎用コピーアルゴリズム

ここでは簡単のために2進木のコピーを考えることにする。コピーする2進木は，図6に示すように，car，cdrの2つのポインタを持つセルから構成され，各ポインタは，他のセルまたはアトムを指し，また，共有ポインタや再入ポインタは存在しない。ここで，アトムを指すポインタをAポインタ，セルを指すポインタをFポインタと呼ぶと，car, cdr にそれぞれのポインタを持つ各セルは，図6に示すように，ＡＡ，ＡＦ，ＦＡ，ＦＦの4種類のセルタイプに分類される。更にコピー木を構成するセルは，タグなし自由リストavailより関数New－Cell（avail）により，1セルづつ取り出される。

図8に，1パスの固定作業領域アルゴリズムTree Copy 1 を示す。そこでは，原2進木は，変数rootの指すスタートセルから出発して，root，car，cdrと前順序でたどられながら，それに対応するコピー木が作成されてゆく。また，この過程で，原2進木に対しては，その構造を一切変更せず，単に原2進木を構成するポインタがたどられるのみである。

補助スタックを使用せずに，効率のよい進木のたどりを実現するため，Tree Copy 1では，自由リストからコピーセルの他に内部スタックを構成するセルを取り出して使用し，スタックとしての使用を終了した時点で，そのセルをコピーセルとして再使用する。つまり，スタックを作成するために，後で使用するコピーセルを自由リストから先取りして使用する。

内部スタックによって原2進木をたどる時，内部スタックには全てのタイプの原セルの番地を格納すなわちプッシュする必要はなく，単にFポインタを2つ持つ原ＦＦセルに関して，そのcarの下ポインタの値oldcdrが内部スタックに保存されればよい。したがって，図6に示す原2進木を前順序でたどってゆくと，図7に示す様に，たどる

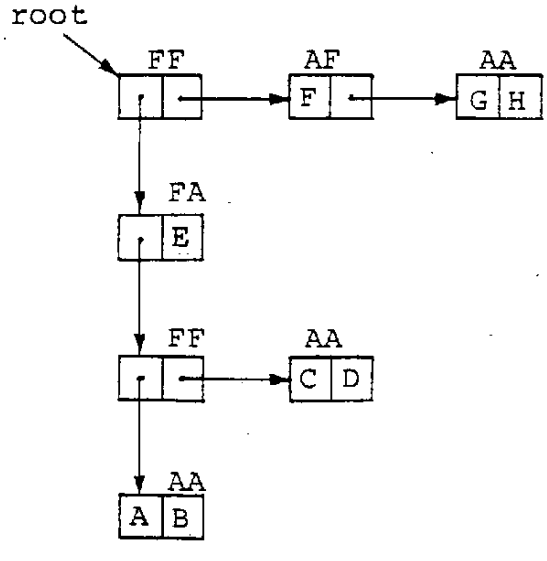


図6. セルタイプの分類

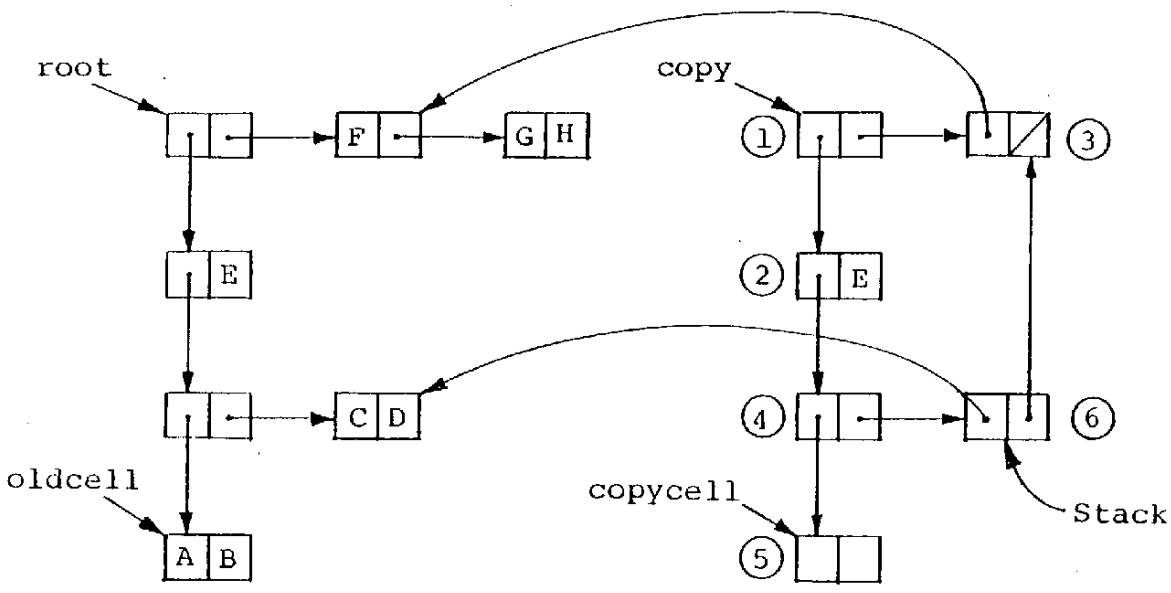


図7. 内部スタックの実現

原2進木に対応して，コピーセル①，②，④，⑤が自由リストavailから取り出され，同時に，大域変数（global variable）Stack によって指されるコピーセル③，⑥から成る内部スタックが作成される（変数Stackの初期値はＮＩＬである）。この時，コピーのＦＦセルのcarポインタは，内部スタックの各セルを指す。

原2進木をたどり，葉を構成する原ＡＡセルに達した時には，自由リストavailから取り出したコピーセルのcar，cdrに原Ａポインタが格納され，次いで，内部スタックはポップされて次のコピーの作成先，即ち，ポップされた内部スタックの先頭のスタック要素に保存されていた原ＦＦセルのoldcdrの指す先へ制御が移る。この時，解放されたスタック要素は，原ＦＦセルのoldcdrの指す原セルに対応するコピーセルとして再使用されることになるため，自由リストから新たなコピーセルが取り出されることはない。

原セルのタイプがＦＡまたはＡＦの場合には，コピーセルに格納されるＦポインタは，自由リストから新たに取り出すコピーセルを指し，また，Ａポインタの値は原Ａポインタ自身であるので，容易に原セルに対応するコピーセルを作成することができる。

ここで示した2進木の汎用コピーアルゴリズムTree Copy 1 は，解析の結果，補助スタックを使用する場合に比較して，最悪の場合でも同速度，最高の場合には，1.25倍の処理速度を達成している。

```
procedure TreeCopy1(root) :
  begin pointer oldcell, copycell, copy, Stack ;
    oldcell <- root      ;  copycell <- New_Cell(avail) ;
    copy    <- copycell  ;  Stack    <- NIL             ;
    until oldcell = NIL do
      begin pointer oldcar, oldcdr, copycar, copycdr ;
        oldcar <- car(oldcell) ;
        oldcdr <- cdr(oldcell) ;
        case CellType(oldcar, oldcdr) of
          AA : car(copycell) <- oldcar   ;  cdr(copycell) <- oldcdr  ;
               copycell      <- Pop_Stack()                          ;
               oldcell       <- Get_Next_Cell(copycell)              ;
          AF : copycdr       <- New_Cell(avail)                      ;
               car(copycell) <- oldcar   ;  cdr(copycell) <- copycdr ;
               copycell      <- copycdr  ;  oldcell       <- oldcdr  ;
          FA : copycar       <- New_Cell(avail)                      ;
               car(copycell) <- copycar  ;  cdr(copycell) <- oldcdr  ;
               copycell      <- copycar  ;  oldcell       <- oldcar  ;
          FF : copycar       <- New_Cell(avail)                      ;
               copycdr       <- New_Cell(avail)                      ;
               car(copycell) <- copycar  ;  cdr(copycell) <- copycdr ;
               car(copycdr)  <- oldcdr   ;  Push_Stack(copycdr)      ;
               copycell      <- copycar  ;  oldcell       <- oldcar  ;
        caseend
      end ;
    return copy
  end
```

```
procedure Push_Stack(cell) :
  begin
    cdr(cell) <- Stack ;
    Stack     <- cell
  end
```

```
procedure Pop_Stack() :
  begin
    if Stack = NIL
      then return NIL
      else
        begin pointer t ;
          t     <- Stack      ;
          Stack <- cdr(Stack) ;
          return t
        end
  end
```

```
procedure Get_Next_Cell(cell) :
begin
  if cell = NIL then return NIL
    else
      begin pointer nextcell ;
         nextcell <- car(cell) ;
         return nextcell
       end
end
```

```
procedure CellType(oldcar, oldcdr):
  begin
    return
      if atom(oldcar)
        then if atom(oldcdr)
                then AA else AF
        else if atom(oldcdr)
                then FA else FF
  end
```

図8. 木の高速汎用コピーアルゴリズム Tree Copy 1

5. あとがき

従来のソフトウエアシステムは，進歩は見られるものの，本質的には，高価な故に小さなメモリ空間を前提として作成されてきており，メモリ空間を有効に使用する技法が伝統的に影響を与えている。しかし，今後は，超ＬＳＩの発達の結果得られる，安価で巨大なメモリ空間を有効を生かすために，従来とは，量・質の双方の面で異ったソフトウエアシステムの構成法が必要となろう。

例えば，現在のリスト処理システムでは，本質的に共有（ share ）する必要のある構造だけでなく，単にメモリの節約という点から，非常に多くの構造が共有される様にシステムが作成されている。また一方では，従来のシステムは，並列処理を前提としては作成されていないだけでなく，狭いメモリ空間を前提としたシステム作成法が，並列処理化を妨げているとも言える。例えば，ＬＩＳＰでＧ.Ｃ.を並列に行なおうとした場合には，非常に複雑なアルゴリズムとなってしまっている。

したがって，今後は，ハードウエアの急激な発達の成果を，如何にソフトウエアシステムが吸収し反映させるかという段階に来ているという認識下に，処理対象の持つ本質的な問題以外の問題に影響を受けることがない様なシステムの作成法が取られることが望まれる。

（ 電総研　ソフトウエア部
情報システム研究室
長谷川　　洋 ）

【参考文献】

〔１〕 Schorr，H，and Waite，W．"An efficieut machine-independent procedure for garbage collection in various list structures". Comm．ACM 10，8 (Ang 1967)，501—506

〔２〕 Lindstrom，G．Scanning"list structures without stacks or tog bits". Inf．Process Letters 2,2（1973），47—51

〔3〕 Lindstrom, G. "Copying list structures using bounded workspace". Comm ACM 17, 4 (April 1974), 198−202.

〔4〕 長谷川 洋："Notes on list marking algorithms using constant workspace". 京都大学数理解析研究所講究録396 (Sept 1980), 310−318.

〔5〕 Clark, D. W "A fast algorithm for copying binary trees". Inf Process. Letters 4, 3 (Dec. 1975), 62−63.

〔6〕 Lee, K. P. "A linear algorithm for copying binary trees using bounded workspace." Comm ACM 23, 3 (march 1980), 159−162.

〔7〕 長谷川 洋："木構造をコピーする高速汎用アルゴリズム", 情報処理学会記号処理研究会資料12−5 (June 1980)

〔8〕 長谷川 洋："固定作業領域による再入リストの高速コピー・アルゴリズム"。信学技報 AL79−30(July 1979)。

〔9〕 Fisher, D. A. "Copying cyclic list structuses in linear time using bounded workspace." Comm ACM 18, 5(May 1975), 251−252

〔10〕 Clark, D. W. "A fast algorithm for copying list structures." Comm ACM 21, 5 (May 1978), 351−357

〔11〕 長谷川 洋："高速リスト・コピー・アルゴリズム" 信学技報 AL 78−81 (Jan. 1979)。

〔12〕 Robson, J. M. "A bounded storage algorithm for copying cyclic structures." Comm ACM 20.6 (June 1977), 431−433.

〔 6 〕 記号処理データフローマシン

1. はじめに

記号処理を行う言語として現在Lispが代表的であり，また他の言語をLispが代表的であり，また他の言語をLispで記述することも多く試みられており，Lispはいわば記号処理の機械語的存在となっている。

ところが多くのLispは既存の計算機の上に実装されており，それを高速に走らせるために様々な工夫がなされている。そのうち特にLispはデータ型が自由であるために，実行時にデータ型のチェックを行なわなければならず，実装時に苦労するところである。また高速化のために「関数引数の課題」を犠牲にしていることが多く，特に関数を値として返す場合に正常に働くシステムは少ない。このような理由から，Lisp専用の計算機を作ろうとする試みがなされ，実際に米国では商用機も出始めている。以上のLispはいずれもプログラムが逐次的に処理されるという点では広い意味でvon－Neumann型であり，主記憶とＣＰＵ間でのデータ交信の道が一本であるため，高速化の際の一つの障害となっている。

近年，データフロー計算機なるものが注目され始め，数値計算の場合には様々な解析がなされている。ところが，それを記号処理に適用する場合にはまだ確固とした定式化は行なわれていない。

ここで提案するのはLisp用のデータフロー計算機であり，データ自身も一つのプロセッサとして考え，機能の分散化をはかったものである。

2. 基本データ

Lispの点対を拡張して，リストを基本データとする（図１）。これは<arg 1> … < arg n > にはデータ（数値あるいは，他のリストへのポインタ）が入っており，ここに「添字」 i と「送り先」 cont が対で送られてくると，< arg i > を cont に送るという動作をするプロセッサと考えられる。以後の説明では図１のような箱をプロセッサと考え，その一番上の欄に演算の種類を書いておくことにする。また<と>でかこまれた部分はデータが入るスロットの名前を表わし，そうでない所は実際のデータが入っているものとする。

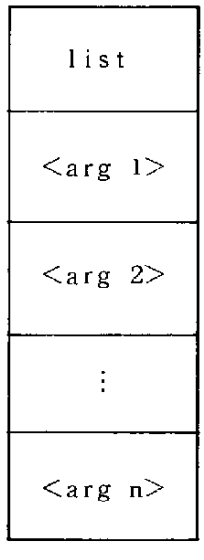


図1. 基本データ list

listに添字や送り先を送りだすプロセッサが必要であるがこれをapplyと呼ぶ（図２）。applyは一般に関数を呼出すのに使用され，全部のスロットが満たさ

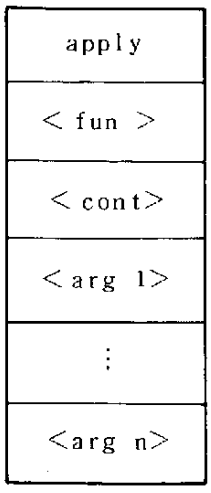


図 2. 関数呼出し apply

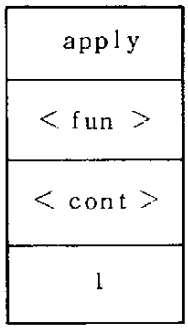
れると送り先 cont と引数 grg 1 … arg n を関数 fun に送り出す。

これを使うと list の最初の要素を取り出す関数 first は図3の左のように書くことができる。これをデータフローグラフで書くとその右の図のようになる。

この他，基本的な演算を行なうプロセッサがあるがそれは引数と送り先のスロットを持っており，その全てがそろうと引数に対して特定の演算を行ないその結果を送り先に送る。また特殊な演算として，list を作る mklist というプロセッサがあり，これはその引数を要素とするような list を作りだす働きをする。ここが通常のデータフロー計算機と異なるところであり，プロセッサが動的に生成されている。

3. 閉関数

ここでは閉関数というのはその値を計算する本体と，そこで参照する引数の結合状態を示す環境の二つがそろったものをいうことにする。すなわち，(LAMBDA (FN)(FN U V)) というのは関数の本体であり，これに U，V の値を表わす環境を付けて始めて閉関数と呼ぶ。



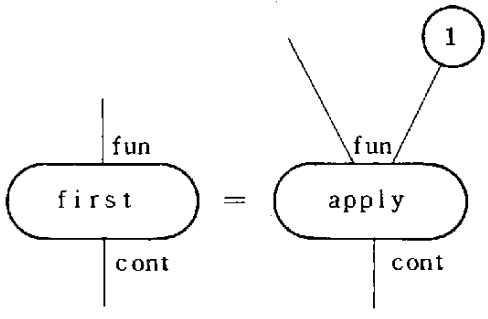


図 3. 関数 first

Lisp では FUNCTION という関数が閉関数を作り出すが，多くの Lisp ではここが不完全なために関数引数がうまく働かないことがある。

閉関数を処理するために closure というプロセッサを用意する（図 4 ）。ここの< env > には環境が入っており，apply によって cont , arg 1 , …… arg n が送られてくるとそれらを<cont>,<arg 1>, ……<arg n>, に中継してさらに<env>を<args>に送り出す。（なおＮＩＬという記号は送り先がないことを表わす。

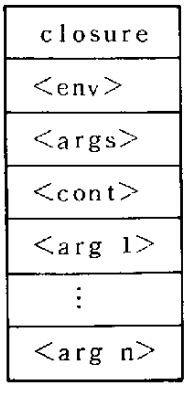


図 4. closure

U，V の値がそれぞれ 3 , 10のとき

(FUNCTION (LAMBDA (FN) (FN U V)))

は図 5 のようになる。そこで使用している spread の第一番目の要素を第二番目以降で示されているところへ振り分ける働きをするプロセッ

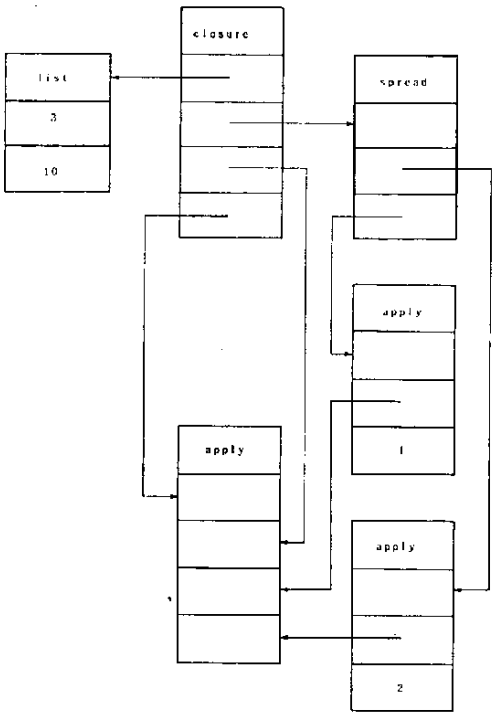


図5. 閉関数(FUNCION(LAMBDA (FN)(FN U V))) U=3, V= 10

サである。この図で，プロセッサの一番目を指している矢印はそのプロセッサ自体へのポインタであり，二番目以降を指しているのはそのプロセッサのそのスロットを表わすものとする。また，この関数では，UやVの値を変えていないので，それらを必要とする場所に直接埋め込んでもよい(図6)が，一般には後で変えることがあるので図5のようにしておく。

このようにclosureというプロセッサを用意することによって，環境まで含めた閉関数を定義するときに，関数本体の複製を作る必要がなくなり，新たに作るのはclosureだけでよくなる。

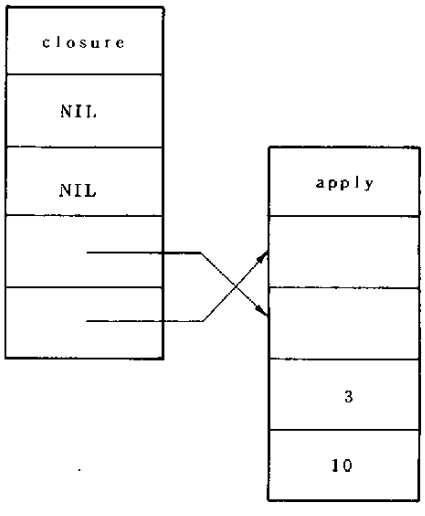


図6. 図5の簡略版

4. 関数呼出し

図5の閉関数が引数として図3のfirstを取ったときの動作を解析してみる。なお図3のfirstは閉関数の形では図7のように書くことができる。

結果をcontという所に送るものとすると，まずclosureにfirstとcontが送られてくるのでそれをそれぞれの場所に送り出し，環境であるlistをspreadに送る(図8. a)。ここで

spreadの要素が全部そろったので最初の要素を二番目，三番目の要素で示している送り先に送る（図8. b ）。次に右の二つの apply の要素がそろったので，listに送り先と1あるいは2を送る。listは送られてきたものを順番に処理して，3と10をそれぞれの送り先に送り出す（図8. c ）。ここで，applyは cont 3，10を closureに送り，closureは contと3，10を closureに送り，closureは contと3をspreadに中継する（図8. d ）。さらに spreadは3を contに対して送り出し，contでは求める結果が得られたことになる。

今までのことをLispのS式で追跡してみる。

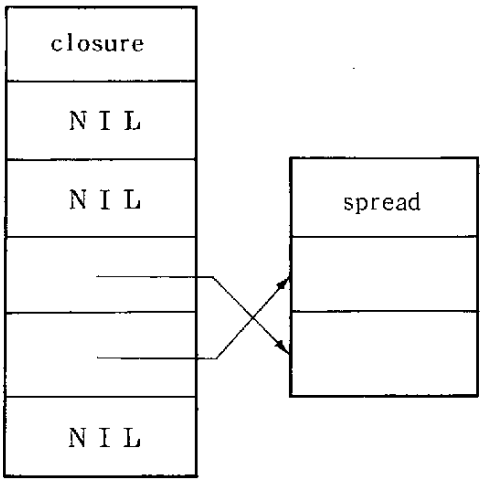


図7. 閉関数 first(FUNCTION (LAMBDA(X Y) X))

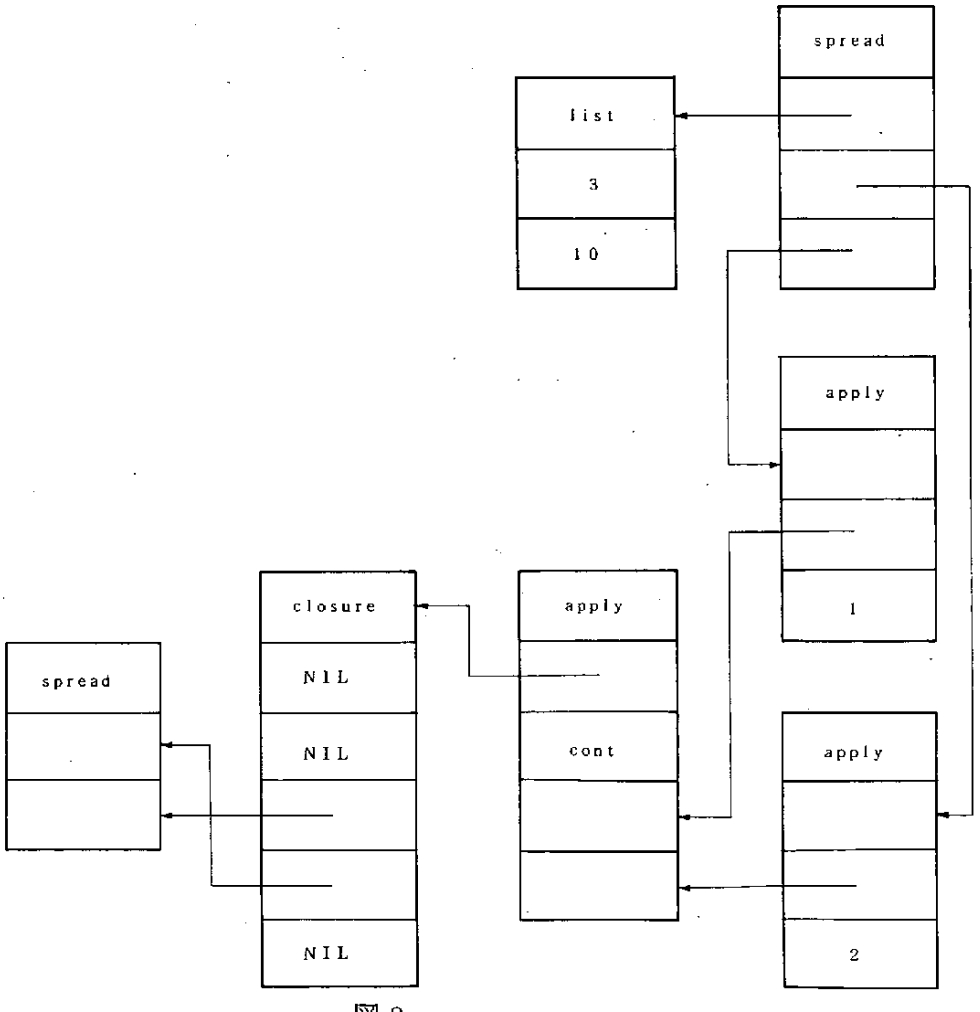


図8. a

図8, 図7を引数として図5を呼出し

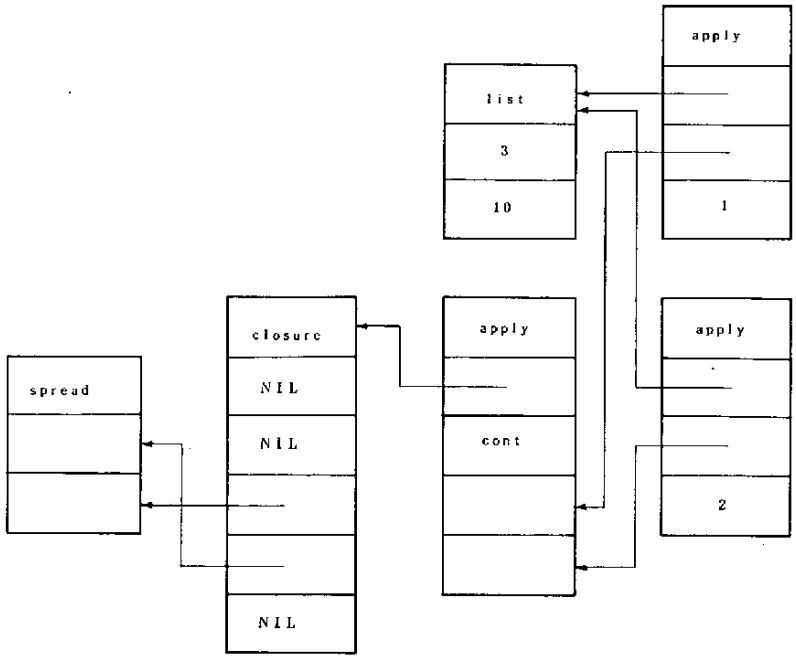


図 8. b

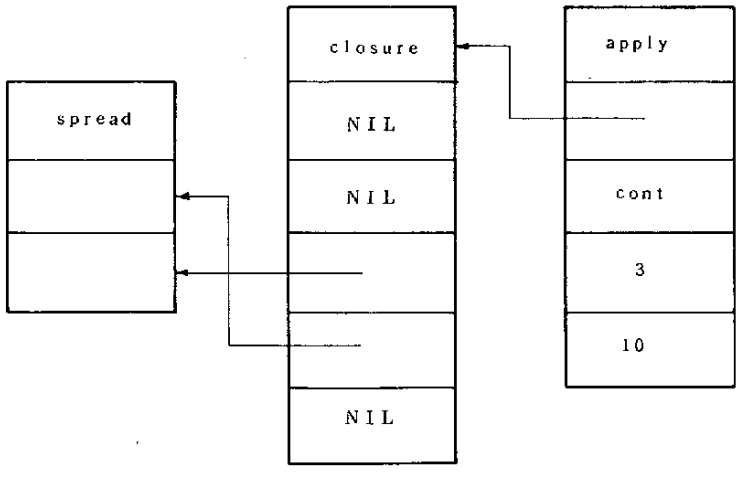


図 8. c

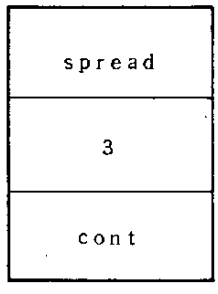


図 8. d

(FUNCTION (LAMBDA(FN)(FN U V)))

をU＝3，V＝10のもとで評価すると

(FUNARG ((U. 3)(V. 10))

(LAMBDA (FN)(FN U V))) …… ＊1

となり，これが図5に相当する。

また，firstは

(FUNCTION (LAMBDA (X Y) X))

と書け，これを評価すると

(FUNARG NIL (LAMBDA (X Y) X)) …… ＊2

となり，図７に相当する。

ここで＊２を引数として＊１を呼出すには

((FUNARG ((U. 3)(V. 10))
(LAMBDA (FN)(FN U V))
(FUNARG NIL (LAMBDA(X Y) X)))

を評価することになる。この式を変形するとＦＮのところにfirstの定義が代入されて

((FUNARG NIL (LAMBDA (X Y) X))
3 10)

となる。（図8.cに相当）。

これを評価するとＸに３がＹに10が結合され，Ｘの値である３が結果となる。

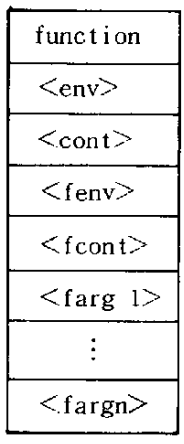


図９．　function

4. 閉関数定義

今までは閉関数はすでに出来上がっているものとしていたが，ここではそれを作り出すプロセッサを考える。

functionというプロセッサがこれを行なうが（図９），これは新らしいプロセッサclosureを作り出し，<fenv> …… <fargn> までをclosureの対応する場所に<env>を<args>に書き込んでそのclosure自体をcontに送る。

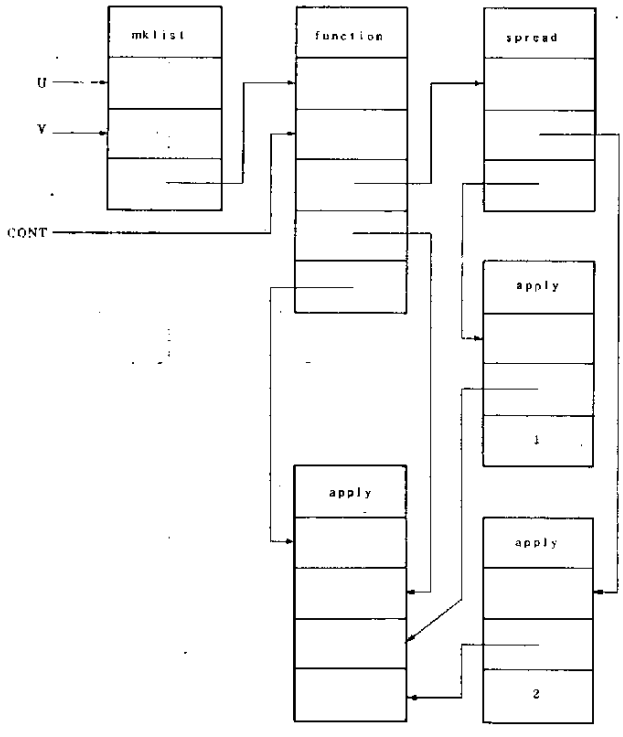


図10　図５の式を作る関数

例として前節で考察したclosureを作るfunctionを図10に示す。この例では，環境は単なるリストであるが，Algolのようなブロック構造の中で閉関数を作ると，環境も階層化して，変数に対するアクセスはAlgolのディスプレイ流の考え方が必要となる。

5. 例　題

例として list を二つの要素だけからなるものにして，これを Lisp の点対と見なして，Lisp のAPPEND を定義してみる。

ＣＡＲをとるのは first を使用すればよいが，ここでは閉関数を使う必要はなく，図３のもので用が足りる。ＣＤＲは同様にして２番目の要素を取り出せばよい。また，CONSには，mklist を使用する。条件文は if というプロセッサを使う（図11）。これは<pred> が真ならば<data>を<con>に送り，そうでなければ<alt>に送る働きをする。

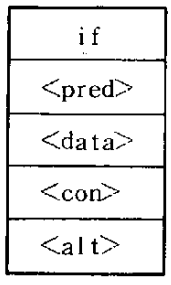


図11　if プロセッサ

これらを使うと，APPENDの定義である

```
(LAMBDA (U V) (COND
    ((NULL U) V)
    (T (CONS (CAR U) (APPEND (CDR U) V)))))
```

は図12のようになる。なお，この図で空いている要素のところには“＜”，“＞”でくくった便宜上の名前を入れてある。

6. Lisp の変更

以上述べたような基本的機構で Lisp を実現しようとしたときに問題となるのは，まず，引数の評価が同時に行なわれるので従来のような左から評価することを利用して副作用を使用するプログラムの方法が作えなくなることである。しかし，これは関数型本来の使用法ではなく，評価に順序付けの必要なところでは，あらわにそのことを指定した方がプログラムが理解し易くなると思われる。

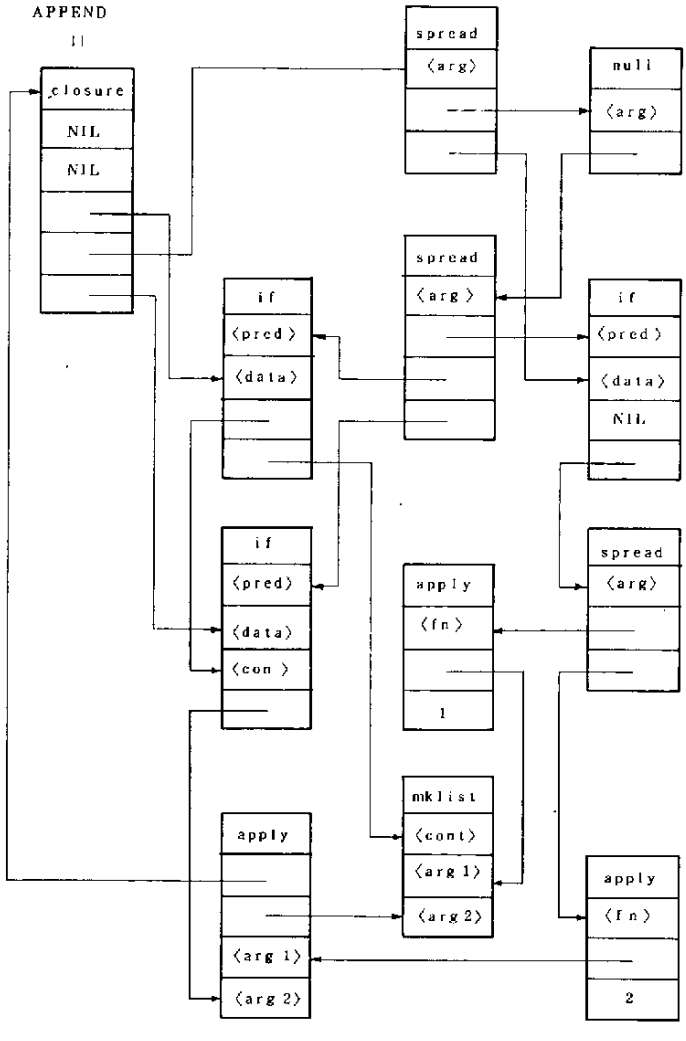


図12　閉関数 Append

次に，より重要な変更点は今までの動的 scope ruleをやめて静的な scope ruleを採用したことである。これは関数引数の場合に効率を落とさずに実行するためと，tail recursionの場合に環境を深くしないですむからである。また実際のプログラムでも，動的 scopeでなければならない場合は少なく，Algol流 block構造を許せば十分に補えるはずである。

6. まとめ

以上では，今までのデータフローマシンにプロセッサを作り出すプロセッサ（mklistとfunction）をつけ加えるとLispのような記号処理を行なうことができることを示した。ただこのように自由にプロセッサを作り出すことができるようになると，全体の資源の管理の問題が起こってくるが，これはLispの場合のガーベージコレクション等の手法が使えるものと思う。また，プロセッサ間の交信の頻度がどの程度になるか，を測定して交信をどの程度まで局所化できるか等の分析を，シミュレータを作り進めている。

電総研 パターン情報部
推論機構研究室
元 吉 文 男

【参考文献】

Bobrow：“A Model and Stack Implementation of Multiple Environments,” CACM 1973 (591−603)

Sussman, Steel Jr, “An Interpreter for Extended LAMBDA Calculus”, MIT AI Memo 349.

Keller, etc：“A Loosely − coupled Applicative Multi − processing System”, NCC 1979 (861−870)

Dennis：“The Varieties of Data Flow Computers”, 1st Intl' Cont on Distributed Computing System, (430−439)。

元吉，横井：“記号処理データフローマシンLispを題材として”情報処理学会21回全国大会

横井：“記号処理データフローマシン 基本計算機構の紹介” 情報処理学会21回全国大会

〔７〕 並行処理の記述と管理

1. まえがき

ハードウエアとソフトウエアのコストの極度なアンバランスが動機となって，新たな計算機システムの研究開発の必要性が主張されている。一般にハードウエアコストは激減したが，ソフトウエアコストはさほど低下していないと言われている。ソフトウエアが大規模になればなるほど，目的のシステムはいかなるものか表現すること（仕様），その仕様を忠実にプログラム化すること（実現），さらに完成されたシステムをその後の要求の変化に応じ矛盾なく拡張変更すること（保守）のいずれもが急激に困難になることがその理由である。

システムの巨大化にともなう複雑化を緩和するために，システムを独立性の高いサブシステム（モジュール）群に分解して開発し，そののちにそれらを組み上げるという方法は広く認められている。大規模システムでは，モジュールに分割しての多人数による開発，既成モジュールの利用，完成後の拡張改良は避けられず，モジュールの統合は，モジュールへの分割と同様に主要な課題である。

現時点で有望と思われる統合の方法は，既成システムと結合して既成モジュールを利用するいわゆる機能分散のアプローチであろう。ところが，各モジュールが単独に機能と性能に関し妥当な動作をすることがわかっていても，それらを機能的に正しくしかも所期の性能が発揮できるよう結合するのはむづかしい筆者らは，既成モジュール群を組み合わせて目的のシステムに仕立て上げる作業を自動化する『分散システムの自動合成』を目指している。モジュール同志の関係，各モジュールの特殊性を『コンフィギュレーション記述言語』を用いてプログラミングし，書かれたコンフィギュレーションプログラムを解析して，そのシステムに適したプロセッサの結合形態とモジュールの割りつけの決定を自動化したい。題材として扱う分野は，時間という定量化可能なメジャが存在し，この研究のインパクトが大きい実時間制御用システムの分野である。本稿では，多数台のロボットを結合し協調動作をさせることを目的として開発された実時間制御用分散計算機システムPPSCを，分割と統合の観点から整理する。

2. モジュール化

プログラムの複雑さを緩和し管理を容易にするために，またプログラムをプロセッサ群上で走らせるためには，そのプログラムを分割し，再び有機的に統合することが必要である。モジュールへの分割の基準としては，機能の違いやデータの種類，インタフェイスとインタラクションの最小化，実行頻度や時間的制約など数多くが考えられる。その大半は，定式化や定量化が不可能であり，抽象化とも密接な関連があるので，モジュール化を機械的に行なうのは困難である。設計とコーディングの段階で人手に頼らざるを得ない。しかし幸なこと

に，実時間処理システムでは，機能の分化がかなりはっきりしており，インタフェイスとインタラクション，実行頻度や時間的制約など定量化可能な測度も多く，モジュール化は比較的容易である。以下にGuarding Processと名づけた並行プログラミング法について述べる。

我々の場合，プログラムの基本要素を次のようなモジュールに限る。モジュールはその内部でのみ参照変更可能な共有データと，処理の流れを記述する複数のパスから成る。

```
module
    .....
    var .....
    path .....                                    (1)
    path .....
    ..... endmodule
```

パスはその処理の途中でモジュールの外部とデータと制御の授受をするために，複数のポートを持つことができる。

```
path id (ins # outs) : n ;
  .....                                           (2)
  begin ..... end
```

各パスのサーバをプロセスと呼ぶ，各パスに対しその数が指定できる。以上の状況を図示する図1のようになる。ここでPATH−1, PORT−1−a, PORT−1−b, が4層になっているのは，PATH−1のサーバは4プロセスであり，各プロセス毎にポートが確保されていることを示している。

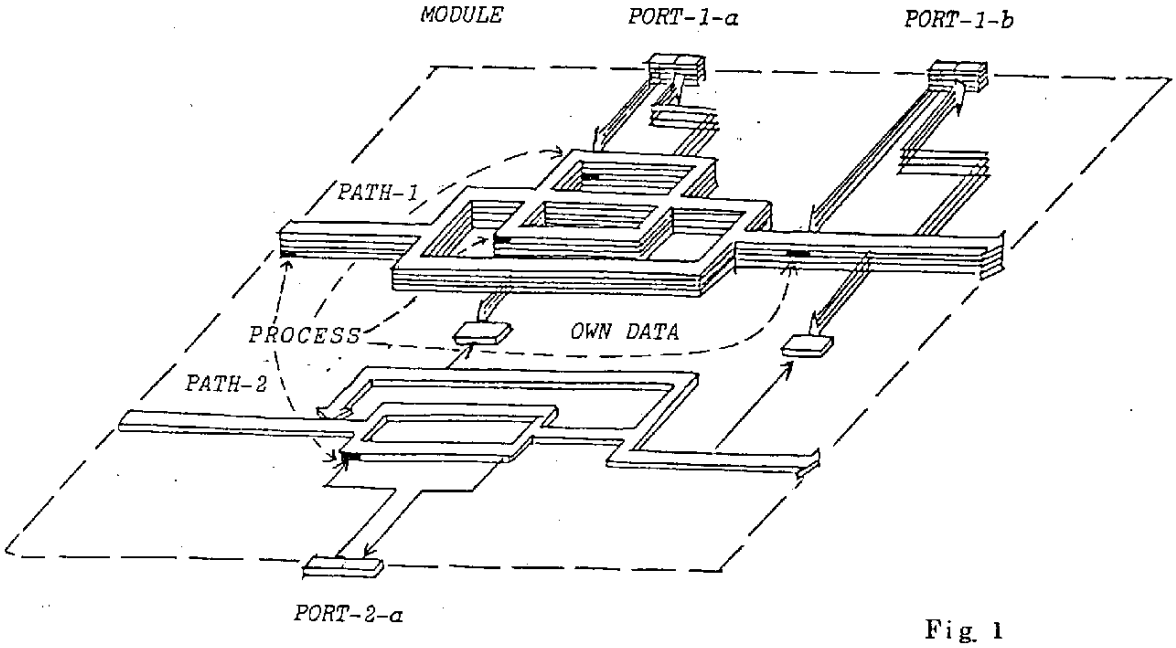


Fig. 1

3. 並行処理

システム内で複数のモジュールが並行動作すると逐次動作する単一モジュールのみからなるシステムにない困難な問題が生じる。その第1は，各モジュールの動作が時間に依存するため，システムの挙動に再現性がないこと，第2はシステムの状態を完全に把握するのが不可能なことである。ここでは論理的な再現生を保証するための並行処理について述べる。タイミング等の時間的再現性については6.に述べる。

モジュールMa パスPa上のプロセスRaiはモジュールMbのパスPbに

connect Mb Pb (exprs # vars) : statement (3)

によって結合要求を出す。Mb・Pb では他のパスに結合されていないひまなプロセスRbjがあれば，Raiの処理に専念させる。ひまなプロセスがなければ，Raiは待たされる。Mb・Pb・Rbjと結合されるとMa・Pa・Raiはstatement部で

call Mb・Pb・Qi(exprs # vars) (4)

によってMb・Pb・Rbj のポートQiを呼ぶ。一方，Mb・Pb・Rbjは

```
acept   C1 , Q1 (ins1 # outs1) : statement1
        C2 , Q2 (ins2 # outs2) : statement2          (5)
          .....                  endacept
```

によって，条件Ciを充し（省略可），かつポートQiに受信するまで待ち，要求を ins_i にとり込み，$statement_i$ の処理が終了したとき $outs_i$ を $Ma・Pa・Ra_i$ のvarsに戻す。

モジュール内のプロセス同志はモニタ同様に共有データを用いて，データと制御の授受を行なう。すなわち，プロセスは

```
when   C1 : statement1
       C2 : statement2                               (6)
       .....        endwhen
```

に出合うと，条件C_1，C_2…… のいずれかが成立するまで待ち対応する $statement_i$ を処理する。

4. 欠陥と障害の検出

3のような並行処理の技法は並行プログラミングと呼ばれ，通常同期と通信に関しオブジェクト指向型の機構に仕立て上げられており，アクセスするサブジェクト（モジュール）のための配慮はなされていない。その結果，モジュールが正しい順序で使用されない（既成モ

ジュールを利用したり，モジュールを利用したり，モジュールの改造などを行なうと，誤った順序での使用がしばしば起きる）ときには永久に待たされる事態も起きる。

3.で述べた並行記述のうち，モジュール内の通信と同期に関する部分はオブジェクト指向型であり，モジュール間の通信と同期に関する部分はサブジェクト指向型になっており，次のようにして，サブジェクト側（モジュールを使用する側）のオブジェクト使用に関するアルゴリズムの欠陥や，さまざまな原因で誘発される制御の流れの異常（制御障害）が検出される。

connect文によって，2つのモジュールのパス上のプロセスは1対1に対応づけられている。それゆえ，connectによって結合される側（スレイブ）が受信するcall要求は，connectを発した側（マスタ）からのものに限られている。それに違反するとcall側の誤りである。受信した要求は図2のようにスレイブ側でacceptによって，受理可能なポートの候補に入っているか否かがチェックされる。候補に入っていなければマスタの誤りである。交信が頻繁なほど細かくチェックされることになる。現実にはもう少し複雑である。チェックの手続を一切記述しないでも，完全にチェックが行なわれることは注目に値する。

制御障害検出時に，もしシステム開発者に，その時点で受理可能なポートを通知すれば，開発者は正しいアルゴリズムへ誘導されるであろう。

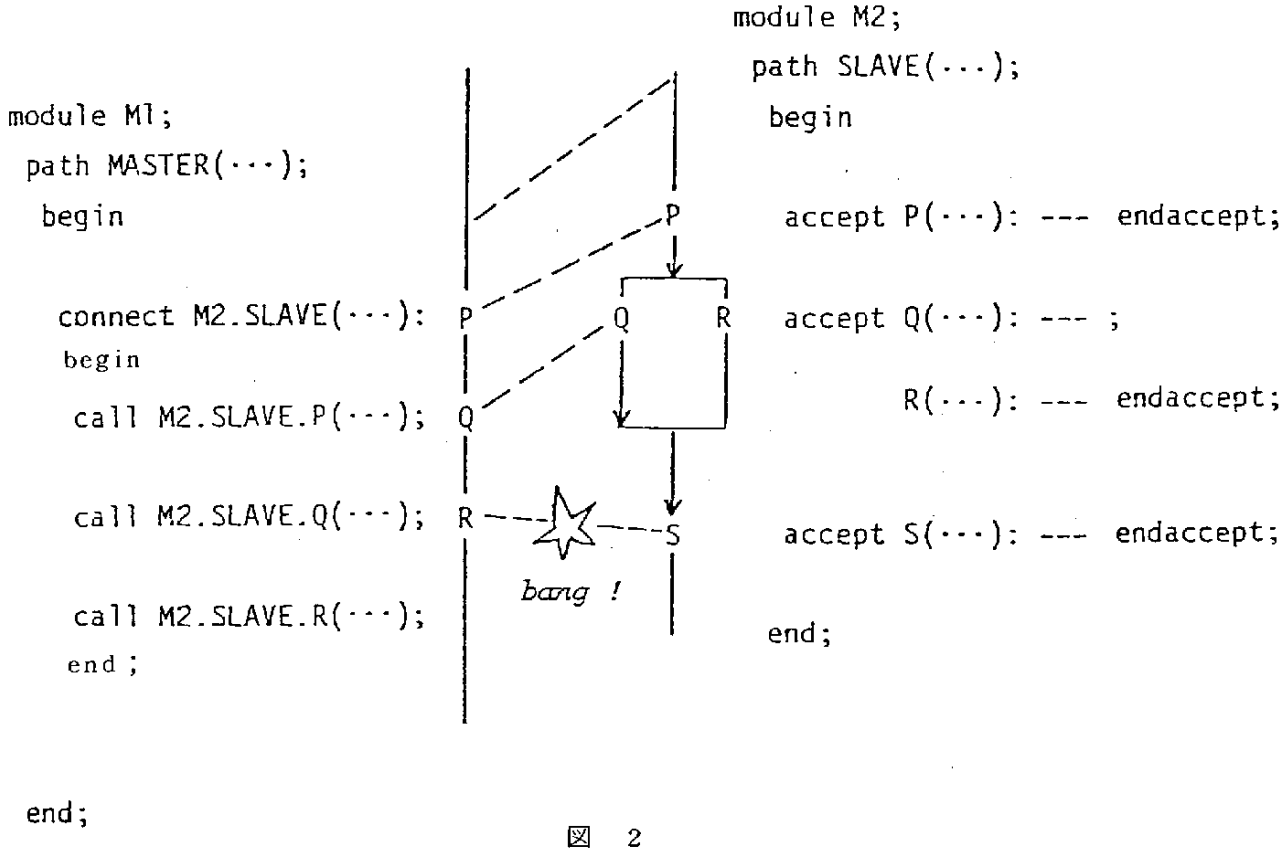


図 2

5. 構　造　化

複雑なシステムを扱い易くするために広く認められた方法のもう１つは，モジュール群に構造を持たせることである。前章までに述べたGuarding Processが非常に高度な構造化の能力を持つことを示す。

AがBを呼び，BはさらにCを呼ぶという状況は，前章までのわく組では次のようになる。

master　slave
A　→　B　→　C
master　slave

"スレイブはマスタからの要求の妥当性をチェックできる"ことは，スレイブはマスタより正しいことを仮定している。この事情は，システムの階層化や階層的な開発方法（図３）とうまく一致する。

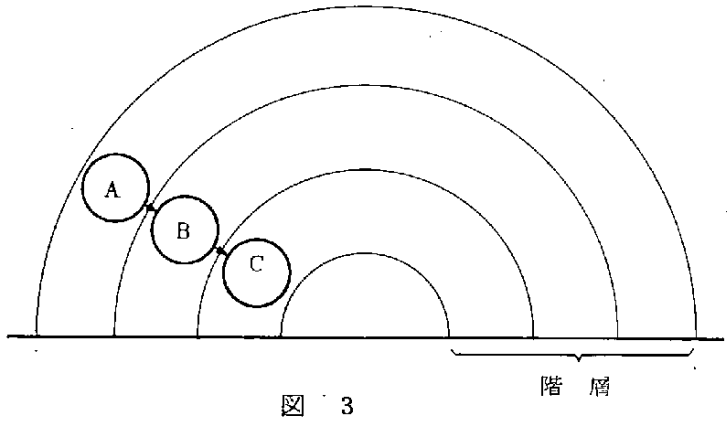


図　3

モジュール間のインタラクションは，マスタ側ではconnect文の内部に限られているので，モジュールの関係はconnectの関係に置きかえられる。connectはモジュールの半順序関係を表わしており，モジュールの統合に際しては，その関係を結合してグラフを作る。できたグラフを用いて色々な性質を議論することができる。以上の結果，Guarding Processがモジュールの論理的な統合化に有効であることは明らかであろう。

6. スケジューリングとモニタリング

統合のさいに追加したモジュールによる負荷の変動，確率的な入出力，通信路の混雑，再送などの要因で，統合することによって一般にプロセスの時間的振舞いは不確定となる。このような状況のもとでも，実時間処理システムでは，ある種のプロセスにタイミングやレスポンスを保証しなければならない。プロセスにタイミングとレスポンスを保証し，あわせてモジュールの走行状況をモニタして，システム全体のチューニング，モジュールあるいはプロセッサのコンフィギュレーションの改善に役立てるためのデッドラインモニタについて述べる。

タイミングやレスポンスを保証して欲しいパスでは次のように記述する。

every arrival − period Service service − period
within response − period at start − time :
statement* endevery　　(7)

この記述から，タイミングとレスポンスを保証するような各プロセッサへのモジュールの割りあて，あるいはプロセッサのコンフィギュレーションの決定が可能と思われるが，未だ定かでない。

プロセッサの中にこのようなモジュールがもし複数割りあてられたときには次の方法によってそれらのタイミングとレスポンスは保証できる。

(7)のように記述されたパスのサーバであるプロセスiは，初期値（APi°, SPi°, RPi°, STi°）とその現在値（APi, SPi, RPi, STi）で特徴づけられる。最も最近にスケジューリングの行なわれた時刻を t_j，t_j で選ばれたプロセスをk，kの走行時間をtとする。時刻 t_1 のスケジューリングは次のようになる。現実には多少の修正が必要であるが，ここではその本質のみを述べる。

$$AP_k-(t_1-t_j)\rightarrow AP_k,\quad SP_k-t\rightarrow SP_k,\quad RP_k-t\rightarrow RP_k$$
$$AP_m-(t_1-t_j)\rightarrow AP_m,\quad RP_m-(t_1-t_j)\rightarrow RP_m\quad \text{for}\quad m\neq k$$

時刻t で選ばれるプロセスは走行可能状態にあるプロセスの集合を Σ としたとき，

$$\min_{i\in\Sigma} RPi=RPn$$

なるnである。次のスケジューリング時刻は

$$t_1+\min\{\{APi\},\{RPi\}\}$$

である。

このデッドラインモニタとGuarding Processは次のようにして融合される。スケジューリング要求を出しているプロセスに，対応するマスタから何らかの要求が入るとstatement*の処理が終了した時点で，そのプロセスはスケジューラの管理から外されevery文の処理が終了する。タイミングやレスポンスに関する要求のないモジュールも，(7)を近似的に適用（現実には同じではない）することができる。プロセッサへのモジュール配分やプロセッサのコンフィギュレーションが悪いとき，あるいはプロセッサの絶対的な性能不足は，負荷のあふれとして検出される。

7. あとがき

以上のGuardingとデッドラインモニタを用いればモジュールの集積性が良くなり，応用システムの開発は容易になるはずである。　残念ながらPPSCは異機種システムであり，その有効性は現在までのところ完全には確認されていない。Guarding Processにもとづく言語も設計を終えているが，不満な点があり実働化には至っていない。コンフィギュレーション記述言語として昇華するには，アルゴリズムより仕様の記述に重点を移す必要があろう。プロ

セッサへの負荷割り当てに関しては，ＮＰ完全性が大きな壁として立ちはだかっている。評価可能な近似アルゴリズムの発明が必要である。

電総研　制御部
情報制御研究室
塚　本　享　治

【参考文献】

(1) 長田，塚本：　実時間制御用分散計算機システムの構築技術，システムと制御 Vol. 24, №12，pp 761 － 769(1980)

(2) M. Tsukamoto：　Structuring Concurrent Programs with Control Fault Detection，proc. 14th IBM Computer Science Symposium，pp. 275—309(1980)
（この文献は非公式文献）

(3) 塚本：分散システムにおける制御障害の検出と状態の回復に関する考察，情報処理学会第２２回大会（１９８１）

昭 和 56 年 3 月 発 行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園３－５－８
機 械 振 興 会 館 内
ＴＥＬ　(434) 8211(大代表)

印刷所　株式会社　正　　文　　社
東京都文京区本郷３－38－14
ＴＥＬ　(815) 7271

55—R015