知識型計画支援システム向業務論理記述言語用プリコンパイラ†

川嶋一宏†† 薦田憲久††
原田俊一††† 三森定道††

生産計画立案等の計画業務において，中間在庫の削減，製品短寿命化等のため，日々の状況にあわせてリアルタイムに，計画問題を処理する計画ソフトを変更できる計画システムが必要となってきた．そこで，筆者らは計画ソフトを業務の専門家が直接取り扱え，多量データの処理も行える新しいタイプの知識型計画支援システムを開発している．このシステムでは，記述内容の責任部署別に，計画ソフトを業務部門が責任を持つべき業務特有の論理（業務論理）と，システム部門が責任を持つべき業務の内容に独立した半固定的な解法の論理（解法論理）に分け，ソフト開発保守を行わせる．特に，業務論理プログラムは業務の専門家が理解できるように，計算順序に依存しない記述順序と業務専門用語による記述を可能とした．業務論理プログラムは業務論理記述言語で記述されるが，処理の効率化のため，プリコンパイラにより手続き型のプログラムに変換する．このプリコンパイラは，限られたメモリ領域内でデータ量に応じたメモリの割り付けを行い，計算に無駄のない計算処理の順序を作成する．いくつかの事例について，このプリコンパイラのオブジェクト性能，コンパイル性能等を評価した．この知識型計画支援システムは，実際の計画業務に適用され，計画システムの開発，保守の容易化，従来方式のソフト開発と同等の実行性能の実現などが確認されている．

1. はじめに

生産計画立案等の計画業務においては，生産技術，材料の急速な進歩や日々のデータ，設備等の状況にあわせて，頻繁に計画ソフトを変更する必要がある．このような要求を満足する計画システムは，従来の要求分析，機能仕様決定，モジュール設計というソフトウェアのライフサイクルモデル[1]を前提としては実現が困難であると思われる．ライフサイクルモデルの考え方は，計算機処理の知識だけでソフトウェアの専門家が開発，保守できるシステムソフトに適していると考えられる．しかし，計画ソフトのように多量なデータを処理し，しかも業務の知識と計算機処理の知識を必要とするアプリケーションソフトの開発，保守は仕様を考える業務の専門家と仕様からプログラムを作成するソフトの専門家では専門が異なるため，ライフサイクルモデルの考え方を適用すると，仕様書作成等に膨大な時間と手間がかかる．また，頻繁に起こる仕様の変更に際しても，変更に時間がかかる．さらに，仕様書がメンテナンスされないことが多く，特にアプリケーションソフトの利用者である業務の専門家がプログラムの内容を理解できなくなる等の問題が生じる．

そこで，業務の専門家がソフトを直接理解でき，しかも，複雑なアルゴリズムをもつ最適化計算等の計算処理も実現でき，多量なデータを処理しても，従来と性能的に等価な新しいタイプのソフト開発方式が必要と考えられる．筆者らは，このような観点から，計画問題を処理する論理（計画論理）を，業務部門（業務の専門家）が作成，保守責任を持つべき業務特有の論理（以下，業務論理と呼ぶ）と，システム部門（計算機処理の専門家）が責任を持つべき業務の内容に独立した半固定的な最適化計算等の解法の論理（以下，解法論理と呼ぶ）に分け，ソフト開発保守を記述内容の責任部署別に行える知識型計画支援システムを開発している[2)-4)]．

知識型計画支援システムにおける業務論理の記述においては，業務の専門家が読みやすい順序，用語が使用できる．このような特徴を持つ業務論理記述言語で記述された業務論理プログラムを効率的に実行させるため，手続き型のプログラムに変換するプリコンパイラを設けている．

本論文では，メモリ割り付けと計算順序作成によるプリコンパイラの計算処理の最適化方式について提案する．まず，この業務論理記述言語について説明し，次に，プリコンパイラの機能，方式を述べ，いくつかの事例についてのオブジェクト性能，コンパイル性能

† Precompiler for Process Logic Description Language in Intelligent Planning Support System by KAZUHIRO KAWASHIMA, NORIHISA KOMODA (Systems Development Laboratory, Hitachi, Ltd.), SHUN-ICHI HARADA (Omori Software Works, Hitachi, Ltd.) and SADAMICHI MITSUMORI (Systems Development Laboratory, Hitachi, Ltd.).

†† (株)日立製作所システム開発研究所
††† (株)日立製作所大森ソフトウェア工場

等の評価を示し，開発したプリコンパイル方式の有効性を示す．

2. 業務論理記述言語の仕様

2.1 対象となる計画問題の取り扱い

生産計画等の計画問題では，注文や材料の特性（寸法や成分），製造方法の制約条件およびコスト等の優先度から，最適な組み合わせやスケジュールを求めることが行われる．その対象となる変数の数は大規模な問題では数千となる．このような計画問題を対象とする場合，プログラムをエンドユーザが取り扱えるようにすることと計算処理の効率化を両立させることが必要である．本論文で扱う知識型計画支援システムでは，業務特有の制約条件や優先度を算定する部分と，それらを前提あるいは評価項目として最適化計算する部分の二つに分ける．一方，現実の計画問題には各種の例外処理があり，上述のように単純で汎用的なアルゴリズム（最適化計算）だけで解ける問題は少ない．そこで，計画問題の取り扱いを**図1**のようにする．

（1） エンドユーザが開発，保守するための業務論理とシステム部門が開発，保守する解法論理とに分離し，作成，保守責任を分担する．

（2） 業務論理と解法論理はソフト的には，対象ごとに専門用語を定義できる変数名和英対応表（業務論理の入出力データのデータ構造と変数名の定義），データ的には条件マトリックス（評価値）をインタフェースとする．

（3） 業務論理は，計算機処理の専門家でないエンドユーザが，変更頻度の高い論理を保守するため，業務の専門用語による記述（理解容易な日本語記述）が可能とする．しかも，思考パターン（例えば，トップダウン）に則した記述ができる業務論理記述言語を提供する．業務論理は対象データ入力，条件マトリックス出力を前提とする．条件マトリックスの行および列は計画の対象となる計画変数（例えば材料，注文）に対応する．各計画変数は数十～数万の実現値（対象データ）をもち，材料 NO, 注文 NO 等の識別子で区別される．条件マトリックスの行 NO や列 NO を変化させ，条件マトリックスの全要素を計算するための繰り返し（ループ），その条件マトリックスの行や列に対応する対象データの入力等の計算制御の手続は，言語の機能としてサポートする．

（4） 解法論理は，比較的変更頻度が低く，むしろ処理効率が重要である．しかも，計算機処理の専門家であるシステム部門が保守するため，従来どおりの汎用言語で記述する．

これにより，変更の多い問題を対象（材料，注文）の特性値を用いて計画時の評価項目の算定方法の記述（業務論理）をエンドユーザが理解，変更できる．さらに，プログラムの記述内容をその内容に責任をもつ部署別に開発保守することができる．

2.2 業務論理記述言語

業務論理の記述内容としては，データ構造と変数名の定義（変数名和英対応表）と，その変数名を用いた業務論理記述言語で業務部門が記述する条件マトリックスの計算式定義（業務論理ソースプログラム）の2種類がある．業務論理ソースプログラムと変数名和英対応表の記述例を**図2**に示す．

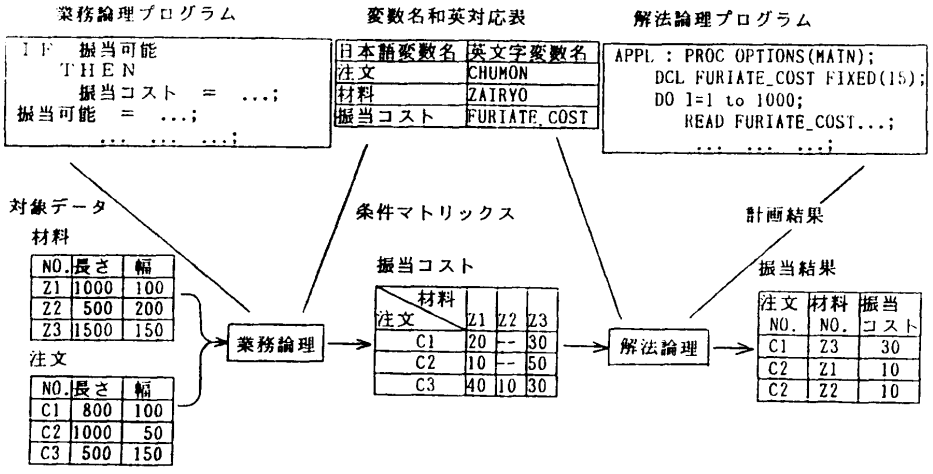


図1 計画支援システムのソフト構成
Fig. 1 Software structure of planning support system.

2.2.1 変数名和英対応表

計画業務において，入出力データのデータ構造と変数名はあまり変更されない．このため，入出力データのデータ構造と変数名の定義は，業務論理ソースプログラムと独立した表形式の変数名和英対応表で行う．

変数名和英対応表は計算機処理のための業務部門とシステム部門のインタフェースとなるものである．業務論理ソースプログラムで使用する入出力データ名を日本語変数名（業務部門が用いるデータ名称）と英文字変数名（システム部門が用いるデータ名称）で定義する．さらに，各入出力データのデータ件数，各データ項目の領域長，属性等も定義する．

2.2.2 業務論理ソースプログラム

業務論理ソースプログラムは条件マトリックスの計算式を業務論理記述言語で定義するものである．また，業務論理が対象データを入力，条件マトリックスを出力することを前提としているので，計算式の定義では，データの入出力や計算の繰り返し等を記述する必要がない．業務論理記述言語では条件マトリックスの計算式をいくつかのブロックにまとめて記述する．ブロック記述の文法は可読性の向上，コンパイルの効率化のため，左再帰性のないように決定する．各ブロックは，ブロック名を定義する BLOCK 文，ブロック内の計画変数を定義する FOR 文，ブロックの入出力変数を定義する ENVIRONMENT 文，計算式を定義する LOGIC 文から記述される．また，LOGIC 文において，計算式の定義は下記に示す式によって行う．

（1） 算術演算式
（2） 論理演算式
（3） 関数式
（4） 業務テーブル検索式
（5） 条件選択式

なお，業務論理記述言語の文法は付録に示す．

変数名和英対応表

REC－ID	日本語変数名	英文字変数名	属性	領域
対象総称(計画変数)	注文	CHUMON	FILE	100
対象項目	注文幅	CHUMON_HABA	数値	3
(入力変数)	注文長さ	CHUMON_NAGASA	数値	4
	種別	SHUBETU	文字	2
対象総称(計画変数)	材料	ZAIRYO	FILE	1000
対象項目	材料幅	ZAIRYO_HABA	数値	3
(入力変数)	材料長さ	ZAIRYO_NAGASA	数値	4
	材料費	ZAIRYO_HI	数値	3
条件マトリックス	振当コスト	FURIATE_COST	数値	4

業務論理ソースプログラム

```
block(振当コスト);
 for (x,y) where
      x is one of 注文,
      y is one of 材料;
 environment;
  input 注文幅(x),注文長さ(x),種別(x),
        材料幅(y),材料長さ(y),材料費(y);
  output振当コスト(x,y);
 end environment;
 logic;
  振当コスト(x,y) = 製造コスト(x,y) + 材料コスト(y);
  製造コスト(x,y) = 注文面積(x) * 単位製造コスト(x,y);
  材料コスト(y)   = 単位材料コスト(y) * 12;
  注文面積(x)     = 注文幅(x) * 注文長さ(x);
  単位製造コスト(x,y) = 製造テーブル(種別(x),材料幅(y));
  単位材料コスト(y)   = 材料長さ(y) * 材料費(y);
 end logic;
end block;
```

図 2 業務論理の記述例
Fig. 2 Example of process logic description.

3. プリコンパイラの機能

生産計画等においては，対象データや条件マトリックスのデータ量が多量となり，膨大な計算量を必要とする．そこで，業務論理ソースプログラムの実行速度を低下させないため，業務論理ソースプログラムと変数名和英対応表を入力して，実行用の手続き型プログラムを自動生成するプリコンパイラを用いる（図3参照）．以下，プリコンパイラの主な機能を説明する．

3.1 計算順序作成機能

業務論理ソースプログラムには，エンドユーザが理解もしくは記述しやすいように，条件マトリックスの計算式が思考過程に沿って記述されているだけで，計算機で計算する順には記述されていない．また，条件マトリックスを計算するための繰り返しの指定（ループ文等）もない．このため，プリコンパイラは，業務論理ソースプログラムに記述された計算式の半順序関係を用い，計算式を計算可能な順序に編集する．

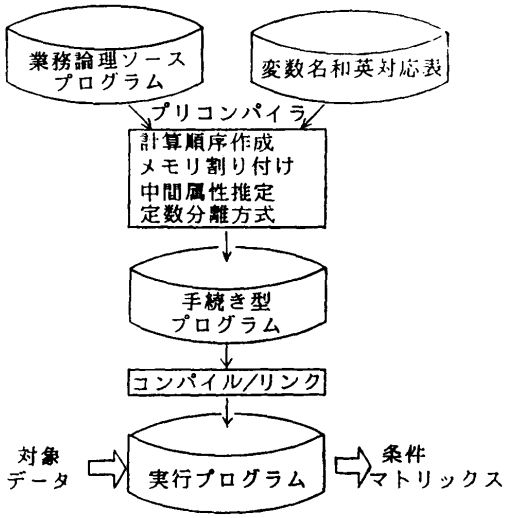


図 3 業務論理実行のプロセス
Fig. 3 Execution procedural of process logic.

特に，繰り返しの指定においては，データの入出力や計算が必要以上に繰り返されることのない計算の順序を制御する．また，条件分岐のある計算式においては，その計算式に必要な計算をすべて先に実行するのではなく，条件判定を優先し，分岐後の計算に必要な計算のみが実行される計算順序を作成する．

3.2 メモリ割り付けとデータ入出力文追加機能

プリコンパイラは使用可能な記憶容量（以下，最大メモリ量）の範囲で，実行可能な手続き型プログラムを自動生成しなければならない．このため，業務論理で処理される大量なデータ（入出力データ，中間データ）のすべてに，メモリを割り付けることはできない．そこで，有効なメモリ割り付けが可能な繰り返し順序を作成し，割り付けるメモリ量が最大メモリ量を越えないように，データの入力や計算等の中間結果にメモリを割り付ける．さらに，対象データや条件マトリックスの入出力文をプログラムに追加する．

（1） 入力データのメモリ割り付け

入力データにメモリを割り付けなければ，条件マトリックスの各要素を計算する度に，同じ入力データを何度もファイルから入力することとなり，実行時間が大きくなる．そこで，入力データを再入力することなく，繰り返し計算ができる入力データのメモリ割り付けを行う．また，割り付けられたメモリにデータを入力する入力文を作成する．

（2） 中間データのメモリ割り付け

条件マトリックスを計算するための繰り返し計算では，必要回数以上に実行される計算式が生じることがある．その計算式をその繰り返しの外に移動し不要な計算を削除するためには，移動した計算式の中間結果（中間データ）にメモリを割り付ける必要がある．プリコンパイラでは，各計算式の計算回数が最小となる中間データのメモリ割り付けを行う．

（3） 出力データのメモリ割り付け

条件マトリックスの要素の値が計算される度に，その結果をファイルに格納すれば，出力データに対しメモリを割り付ける必要はない．しかし，条件マトリックスの要素の計算順序と出力順序が異なる場合には，出力データのメモリ割り付けを行う．また，メモリからデータをファイルに出力する文を作成する．

3.3 その他の機能

プリコンパイラには，対象データ入力，条件マトリックス出力を前提としたオブジェクトの最適化機能の外に以下の機能がある．

（1） 定数分離機能

業務論理には定数が数多く含まれているが，業務論理の中で定数の変更が最も多い．このため，業務論理ソースプログラム内の定数（プログラム内に記述されている定数データ）をプログラムから分離し，業務論理実行時に入力データとして読み込むようにする．これにより，定数のみの変更に対するコンパイル等のオーバヘッドを削減する．

（2） 中間データ属性作成機能

中間データの属性は定義されない．各中間データを計算する計算式より，有効数値の考え方などに基づき中間データの属性（タイプ：数値，論理値，文字列；領域：桁数，長さ）を推定し，中間変数の宣言文を作成する．

4. 最適化手法

4.1 メモリ割り付けによる計算時間の最適化手法

コンパイラの最適化では，ループ中の式で繰り返しに対して，値の変化しない式をループ不変な式と呼ぶ．通常，ループ不変な計算をそのループの外へ移動することによって，計算回数を削減する最適化を行っている[5]．本論文ではこの考え方を発展させ，数次元の条件マトリックスを計算するための多重なループの中で，メモリを必要としないループの最適化とメモリを使用するループの最適化に分けて考える．

メモリを使用しないループの最適化はコンパイラと同様に，内側のループにループ不変な計算式を内側のループの外に移動する．この最適化は，外側のループにループ不変な計算であっても，内側のループにループ不変でなければ最適化はできない．それに対して，メモリを使用するループの最適化は，多重なループの外側のループに対してループ不変な計算式を，その計算の中間結果を格納しておくメモリを割り付け，外側のループの外に別の独立したループを作成して移動するものである．多量のデータを処理する場合は，後者の最適化を行うためには大量のメモリが必要となる．そこで，メモリ割り付けによる計算時間の最適化では，使用可能なメモリ量まで，効率よくメモリを割り付け，計算式を移動する．

この最適化を実現するためプログラムの構造を条件マトリックス生成トリーと名づけたトリー構造で表現する．条件マトリックス生成トリーは変数をノード，変数間の参照関係をアークで示したものである．ノードには，対応する変数の持つデータ量と計算に要する

計算時間，対応する変数の計算式をどの計画変数で繰り返すかを示すラベルをつける．図4は，図2に示す変数名和英対応表と業務論理ソースプログラムを条件マトリックス生成トリーで表現した例である．これにより，計算式間の関係と各計算式の繰り返し，計算式を移動するために必要なメモリ量を表現でき，メモリを割り付ける変数の探索や計算式の移動が効率的に実現できる．

条件マトリックス生成トリーの構造を構造マトリックス C で示す．構造マトリックス C は変数間の参照関係を 0，1 で示す2値マトリックスである．左辺の変数 i で右辺の変数 j の計算式があるとき，C_{ij} は1で，それ以外のときは0である．変数 i のデータ長を d_i，変数 i の計算時間（入力変数の場合は入力時間）を e_i と表す．また，変数の総数（ノードの数）を N，計画変数の総数（条件マトリックスの次元数）を M，各計画変数 j が持つ実現値（対象データ）の数（繰り返しの数）を K_j とする（図4参照）．さらに，ノード i に付いたラベルを M 次元2値ベクトル P_i で示す．ノード i に対応する変数が計画変数 j で繰り返し計算する必要がある場合は $P_{ij}=1$，ない場合は $P_{ij}=0$ である．

条件マトリックス生成トリーにおいて，同一のラベル P を持つノードの集合 S_P は，対応する計算式が同じ計画変数のループに対して，ループ不変な計算式であることを示している．この集合は最大 2^M-1 個存在する．図4の場合，「注文面積」の計算は材料の計画変数による繰り返しに対して，ループ不変な計算式であり，注文の計画変数による繰り返し計算が必要である．このような計算式の集合としては，図4では，材料，注文，材料と注文の計画変数をラベル P に持つ三つの集合が存在する．この集合のノードのうち，ノード間に関係があるものは条件マトリックス生成トリー内でサブトリー T_k を形成している．

メモリを必要とするループの最適化を行う場合，このサブトリー T_k に含まれる計算式ごとに計算式を移動すると，そのサブトリー T_k に含まれるすべてのノードに対してメモリを割り付ける必要はない．サブトリー T_k 内のノードで，他の集合から参照されるノード（サブトリーのルート R_k）に対してのみメモリを割り付ければよい．図4の例で，注文幅，注文長さ，注文面積を含むサブトリーでは，サブトリーのルート（注文面積）のみにメモリを割り付けるだけでよい．このサブトリー T_k の全ノードを計算するのに要する計算時間 E_k は

$$E_k=\sum_{i\in T_k} e_i$$

で示される．また，サブトリー T_k の計算に必要な繰り返し回数 L_k は，サブトリー T_k 内の任意のノード i を選ぶと，

$$L_k=\prod_{j=1}^{M} K_i P_{ij}$$

で計算される．サブトリー T_k の計算に必要な全計算時間 F_k は

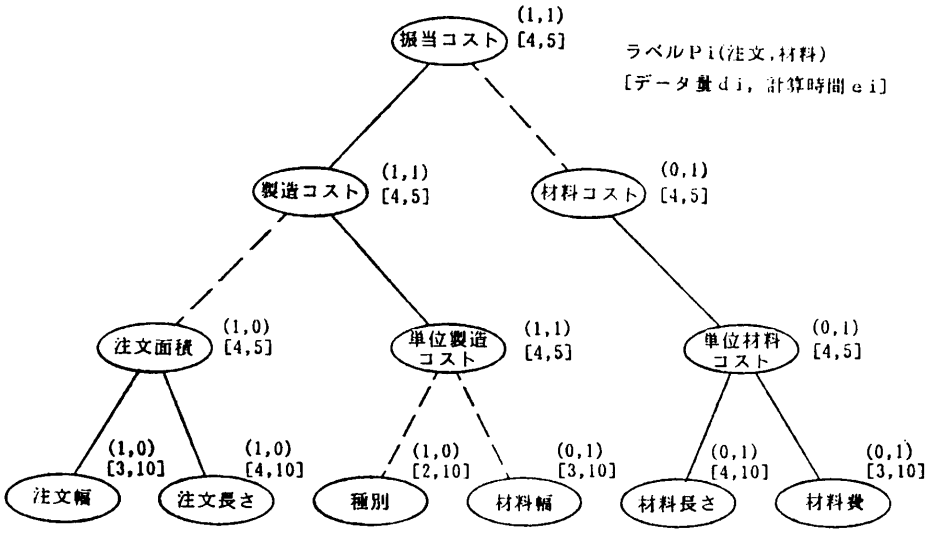


図4　条件マトリックス生成トリーの例
Fig. 4　Example of condition matrix generation tree.

$F_k=E_kL_k$

である．また，メモリを使用するループの最適化でサブトリー T_k の移動に必要なメモリ量 D_k は，サブトリー T_k のルート R_k のデータ量 $\bar{d}$ によって，

$D_k=L_k\bar{d}$

となる．

すべてのサブトリー T_k の移動に必要なメモリ量 D_k が割り付け可能ならば，条件マトリックスの計算時間は F_k の合計となる．しかし，メモリ量 D_k は各計画変数が持つ実現値の数 K_j の積となるため，実際の大きな問題ではメモリを使用する最適化によって計算式を移動することができない．そこで，メモリ割り付けアルゴリズムではメモリを使用しないループの最適化を併用し，業務論理の実行に割り当て可能なメモリ量（以下，最大メモリ量 G と呼ぶ．）まで，メモリを割り付け計算式の最適化を行う．

多重なループの計算では，ループをどのような順序にするか，すなわち，内側と外側のループをどの計画変数による繰り返し計算とするかによってメモリを使用しないループの最適化で移動できるサブトリーが異なる．図4の例で，注文を外側，材料を内側とするループの順序では注文面積をルートとするサブトリーが移動でき，材料を外側，注文を内側とするループの順序では材料コストをルートとするサブトリーが移動できる．また，このメモリを使用しないループの最適化を利用することで必要とするメモリ量を大幅に削減することができる（図5参照）．

そこで，ループの順序を順序番号 B で表す．順序番号 B_j は計画変数 j による繰り返しのループが内側から何番目のループであるかを示す．例えば，図4の例で，注文 ($j=1$) を内側，材料 ($j=2$) を外側のとき，B は (1,2) で，注文 ($j=1$) を外側，材料 ($j=2$) を内側のとき，B は (2,1) である．最も内側のループから，メモリを使用しないループの最適化で移動できるサブトリー T_k は順序番号 $B_j=1$ である計画変数 j に不変な計算式を移動することができる．例えば，図4の例で $B=(1,2)$ であるとき，注文 ($j=1$) に不変な計算式の集合（サブトリー T_k），すなわち，ラベル $P_{i1}=0$ であるノードの計算式の集合で，材料幅および材料コストをルートとするサブトリーに含まれる計算式が移動できる．

このメモリを使用しないループの最適化によって，移動されるサブトリーのルートにメモリを割り付ける必要がなく，材料幅と材料コストに対する 1,000 件のメモリを削除できる．逆に，順序番号 B が (2,1) のとき，メモリを使用しないループの最適化で移動できるサブトリーは種別，注文面積をルートとするサブトリーで，種別，注文面積に対する 100 件のメモリを削減できる．図4の例では，順序番号 B を (2,1) とするよりも (1,2) とする方がメモリを削減できる．メモリ割り付けアルゴリズムでは，削減できるメモリ量が大きいループの順序を作成する．

さらに，メモリを使用しないループの最適化で移動できなかったサブトリーはメモリを使用するループの最適化を行う．まず計算量の大きい順に最大メモリ量 G までメモリを割り付け，メモリを割り付けたサブトリーの集合 Q に含まれる計算式を上記の順序番号 B で指定されたループに独立したループに移動する．このことによって，最大メモリ量 G の範囲で，メモリを有効に割り付け，計算回数の最適化を行える．

4.2 メモリ割り付けアルゴリズム

ステップ 1：構造マトリックス C の作成

（1） 条件マトリックス生成トリーの構造マトリックス C を 0 に初期化する．

（2） 計算式を入力し，すべての計算式について，計算式の左辺の変数を i，右辺の変数を j とした C_{ij} を 1 とする．

（3） 各変数の直後に記述されている計画変数名に従い，ノード i のラベル P_i を設定する．

ステップ 2：サブトリー T_k の作成

（1） $k=1$, $j=1$ とする．

（2） $T_k=\{j\}$ とする．

（3） $i\notin \bigcup_{k=1}^{k} T_k$ となるすべてのノード i について $C_{ij}=1$ もしくは $C_{ji}=1$ かつ $P_i=P_j$ ならば T_k にノード i を追加する．

（4） いずれかのサブトリーに含まれないノードが存在しなければ（5）へ行く．そうでなければ，$j\notin \bigcup_{k=1}^{k} T_k$ なる任意のノード j を定め，$k=k+1$ として（2）へ戻る．

（5） サブトリー数 $A=k$ とする．

（6） 各サブトリー T_k について $C_{ij}=1$ ($i\notin T_k$, $j\in T_k$) もしくは，$\sum C_{ij}=0$ となるノード j を $R_k=j$ とする．

ステップ 3：サブトリー T_k のデータ量 D_k，計算量 F_k の計算

$$D_k=dr\prod_{j=1}^{M} K_jP_{rj}$$

$: r=R_k$

$$F_k=\sum_{i\in T_k} e_i \prod_{j=1}^{M} K_j P_{rj}$$

ステップ 4：繰り返し順序の作成

（1） 順序番号 $B_j=0$, ワークベクトル $V_j=0$ (j: 1～M), ループ内計算トリー集合 $U=\phi$ とする.

（2） $V_j=0$ となる任意の j において, $V_j=1$ とし, V と同じラベルをもつサブトリー T_k の集合 U'_j に含まれるサブトリーのメモリ量の合計 ω_j を計算する.

（3） 最も大きい ω_j を選択し, $B=B+V$, $U=U+U'_j$ とする.

（4） $V_j=0$ となる任意の j があれば（2）へ戻る.

ステップ 5：計算式の移動

（1） 割り付けメモリ量 $W=0$, 移動サブトリー集合 $Q=\phi$ とする.

（2） ループ内計算サブトリー集合 U もしくは移動サブトリー集合 Q に含まれないサブトリーの集合を Q' とする.

（3） Q' に含まれるサブトリー T_k で, E_k が最も大きい k を選択する.

（4） $W+D_k \leqq G$ であれば Q に k を追加し, $W=W+D_k$ とする. $W+D_k \leqq G$ でなければ U に k を追加する.

（5） すべてのサブトリーが U もしくは Q に含まれていなければ（2）へ戻る.

4.3 メモリ割り付けアルゴリズムの動作例と評価

図4に示す条件マトリックス生成トリーの例（図2の業務論理）を用い, メモリ割り付けアルゴリズムの動作例を**表1**に示す. メモリ割り付けアルゴリズムのステップ1～3では, 構造マトリックス C_{ij}, サブトリー T_k および各サブトリーのデータ量 D_k, 計算量 F_k が計算される. ステップ4では, 最適化に必要となるメモリ量が少なくなるループの順序が決定される. 表1の例では, 外側のループとして「材料」のループが選択され, 7,000 バイトのメモリを節約できる. さらに, ステップ5では, 計算量の大きいサブトリーは最大メモリ量を越えない限り多重なループの外へ移動される. **図5**に最適化の動作結果を示す.

データの入力, 実行可能な順序に並べられた計算のすべてを多重なループの内側で繰り返すプログラムでは振当コスト, 単位材料コスト等の計算式, 注文幅, 材料幅等の入力の処理がすべて 100,000 回繰り返される. 一方, メモリ割り付けアルゴリズムを適用すれば, 注文面積と種別に 600 バイトのメモリを割り付けるだけで, 振当コスト, 製造コスト, 単位製造コストの計算式は 100,000 回, 材料コスト, 単位材料コストの計算式と材料幅, 材料長さ, 材料費の入力は 1,000 回, 注文面積の計算式, 注文幅, 注文長さ, 種別の入力は 100 回で済む. このようにメモリ割り付けアルゴリズムにより, 重複した計算（同じ中間結果を得る計算）がなく, 必要なメモリ量を少なくできる. また,

表 1 最適化の動作例
Table 1 Example of optimizing.

ステップ 4	順序番号 B_1	順序番号 B_2	ワークベクトル V_1	ワークベクトル V_2	ループ内計算トリー集合 U	ワーク集合 U'	データ量 $\sum D$
(1)	0	0			ϕ		0
(2) $j=1$			0	1		2, 5	7000
$j=2$			1	0		3, 4	600
(3)	0	1			2, 5		
(2) $j=2$			1	1		1	40000
(3)	1	2			1, 2, 5		
(4)	1	2			1, 2, 5		

ステップ 5	計算量 F_k	データ量 D_k	メモリ量 W	ループ内計算トリー集合 U	ワーク集合 Q'	ループ外移動トリー集合 Q
(1)			0	1, 2, 5		ϕ
(2)					3, 4	
$k=3$	2500	400				
$k=4$	1000	200				
(3) $k=3$	2500	400	400	1, 2, 5	4	3
(3) $k=4$	1000	200	600	1, 2, 5		3, 4
(4)			600	1, 2, 5	ϕ	3, 4

注) サブトリー：1振当コスト, 2材料コスト, 3注文面積, 4種別, 5材料幅

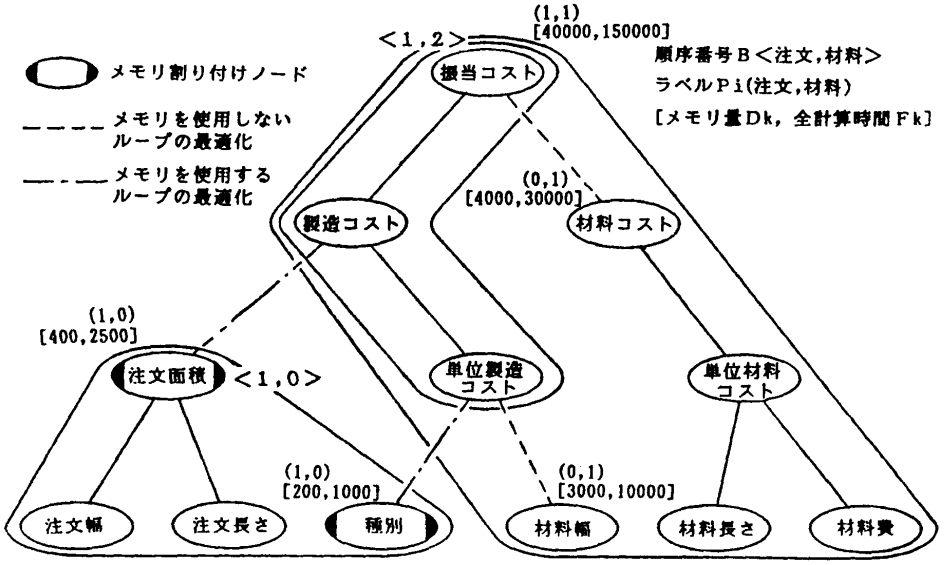


図 5 最適化の動作結果
Fig. 5 Result of optimization.

メモリ割り付けでは，最大メモリ量を越えないようにメモリを割り付けるので，データ量が大きくなっても実行可能な手続き型プログラムを作成できる．図6に最適化の動作結果により得られるコード生成の例を示す．

5. プリコンパイラの全体構成

プリコンパイラの全体構成を図7に示す．以下，各部の処理内容を説明する．なお，論理マスタにはあらかじめプリコンパイルされた業務論理ソースプログラムが格納されている．

```
APPL :PROC OPTIONS(MAIN);                          /*宣言文*/
 DCL (注文幅,注文長さ,種別(100))...;
 DCL (材料幅,材料長さ,材料費)...;
 DCL (振当コスト,製造コスト,材料コスト,
      単位製造コスト,単位材料コスト,注文面積(100))...;
 DO X=1 TO 100;
  READ(注文幅,注文長さ,種別(X));                   /*入力文*/
  注文面積(X) = 注文幅 * 注文長さ;                  /*計算文*/
 END;
 DO Y=1 TO 1000;
  READ(材料幅,材料長さ,材料費);                    /*入力文*/
  単位材料コスト = 材料長さ * 材料費;               /*計算文*/
  材料コスト = 単位材料コスト * 12
  DO X=1 TO 100;
   単位製造コスト = 製造テーブル(種別(X),材料幅);
   製造コスト = 注文面積(X) * 単位製造コスト;
   振当コスト = 製造コスト + 材料コスト;
   WRITE(振当コスト);                               /*出力文*/
  END;
 END;
END APPL;
```

図 6 コード生成の例
Fig. 6 Example of procedural program.

（1） 字句解析部

字句解析部は業務論理ソースプログラムを入力し，字句解析を行い，業務論理ソースプログラムに含まれる定数を定数ファイルに出力する．さらに，定数以外の業務論理ソースプログラムをあらかじめ登録されている論理マスタの内容と比較し，一致していれば，定数のみが変更されたとしてプリコンパイラの処理を終了する．

（2） 構文解析部

構文解析部は変数名和英対応表を入力し，業務論理ソースプログラムを構文グラフに基づき，構文チェックを行い，構文エラーを出力する．構文エラーがなければ，ブロック名，業務テーブル名，関数名，変数名等の登録を行う．

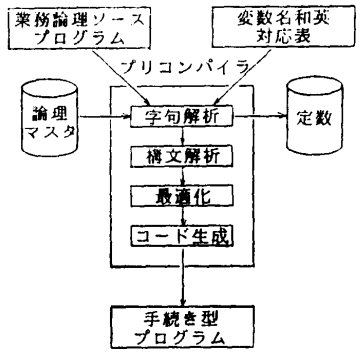


図 7 プリコンパイラの全体構成
Fig. 7 The whole structure of precompiler.

(3) 最適化

4章で説明したアルゴリズムに従い，メモリを割り付ける量が小さくなるようにループの順序を決定し，計算時間が大きい変数に，メモリを割り付け，計算式を移動し，計算時間を最小化する．

(4) コード生成部

コード生成部は出力する手続き型プログラムの言語仕様に合わせて，業務論理に含まれる入力変数，出力変数，中間変数の宣言文とパラメータ化した定数データを定数ファイルから入力するための入力文，繰り返し計算の指定(Loop文)と対象データの入力文，計算式，条件マトリックスの出力文を出力する．

6. プリコンパイル方式の評価

6.1 オブジェクト性能の評価

プリコンパイラのオブジェクト性能を測定するため，二つの対象データで対象項目（入力変数）が19個，中間変数が8個，2次元の条件マトリックス（出力変数）が1個，計算式が9式なる業務論理（一つの業務テーブルを含む）を記述し，その記述からプログラムを作成した．業務論理ソースプログラムの記述は対象間の関係をそのまま記述すればよく，容易であった．また記述者による業務論理の説明がなくても，業務論理の内容の理解は容易であった．

業務論理ソースプログラムを3人のプログラマにより手続き型言語でプログラムした場合と，業務論理記述言語で記述しプリコンパイラで手続き型プログラムを自動生成した場合のプログラムの実行時間の比較結果を図8に示す．(1)のプログラムをのぞいて，実行時間はほぼ同等である．(1)のプログラムは，データの入出力や計算の中間結果を記憶して置くメモリがなく，入力データをファイルから1件ずつ読み込み処理されている．このため，処理するデータ量を考慮しなくても，大量のデータの処理ができる．しかし，ほかのプログラムより，データの入力が5.5倍となるため，実行時間がかかる．(2)～(4)のプログラムで実行時間を比較すると，プリコンパイラ生成のプログラム(4)に対して，(2)は条件分岐が考慮されていないため，計算量が大きく実行時間がかかる．ま

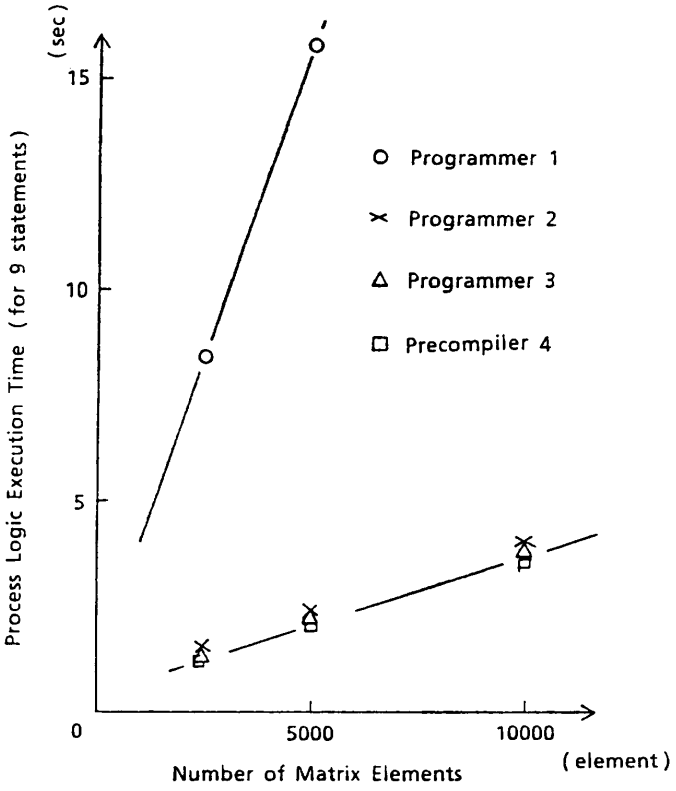


図8 プリコンパイラのオブジェクト性能
Fig. 8 Object performance of precompiler.

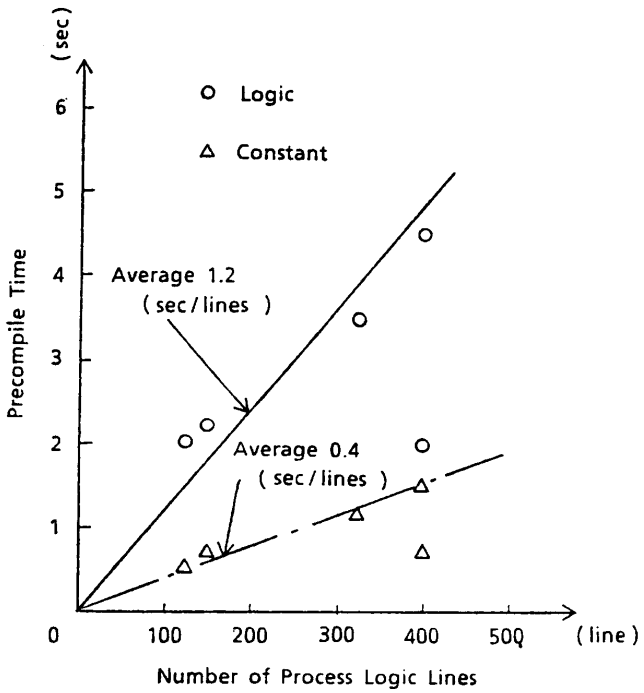


図9 プリコンパイラのコンパイル時間
Fig. 9 Compile time of precompiler.

た，(3)のプログラムは余分なメモリを割り付けており，若干実行時間がかかっている．このような点から，プリコンパイラが自動生成した手続き型プログラムとプログラマが作成した手続き型プログラムは同等の処理性能であることがいえる．

なお，業務論理の実行形態には TSS とバッチが考えられる．これらの実行形態を選択するためには，業務論理の実行時間を業務論理の大きさや処理するデータ量から推定する必要がある．そこで，業務論理の大きさを計算式の数と業務テーブルの行数の和とし，条件マトリックスの 1,000 要素に対する業務論理の実行時間（M 180, CPU タイム）を**図 9** に示す．適用した業務論理の大きさの平均が 30 ステートメントであり，それに対する条件マトリックス 1,000 要素の実行時間（M 180, CPU タイム）は約 0.5〜1.3 秒かかった．

6.2 コンパイル性能

プリコンパイラのコンパイル性能評価のため，記述した業務論理ソースプログラムのプリコンパイル時間等を測定した．

（1） プリコンパイル時間

業務論理ソースプログラムの大きさ（ソースプログラムの行数）と論理変更（プログラム，定数ファイル作成），定数変更（定数ファイル作成）のプリコンパイル時間（M 180, CPU タイム）を**図 10** に示す．適用した五つの業務論理で論理変更時のプリコンパイル時間は 2〜4 秒，定数変更時のプリコンパイル時間は 0.5〜1.5 秒であった．

（2） 変 換 率

業務論理の大きさ（ステートメント）とプリコンパイラが出力したプロブラムの大きさ（ステートメント）の関係を**図 11** に示す．手続き型プログラムでは，変数の宣言文，データ入出力文，繰り返し計算のループ文等のステートメントが必要であり，業務論理の大きさ（ステートメント)/手続き型プログラムの大きさ（ステートメント）の変換率は約 1/20 であった．

7. 結 言

業務部門の担当者が業務の知識を用いて対

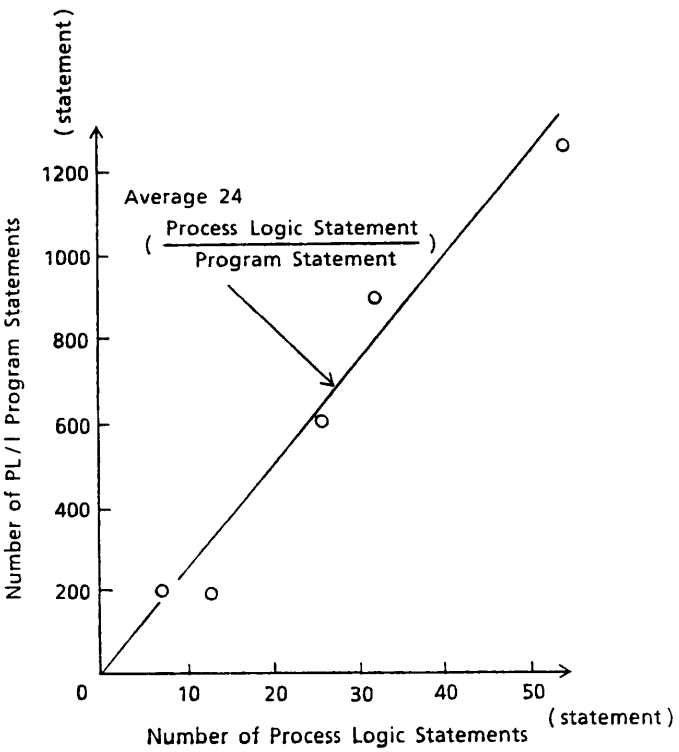


図 10 プリコンパイラの変換率
Fig. 10 Exchange rate of precompiler.

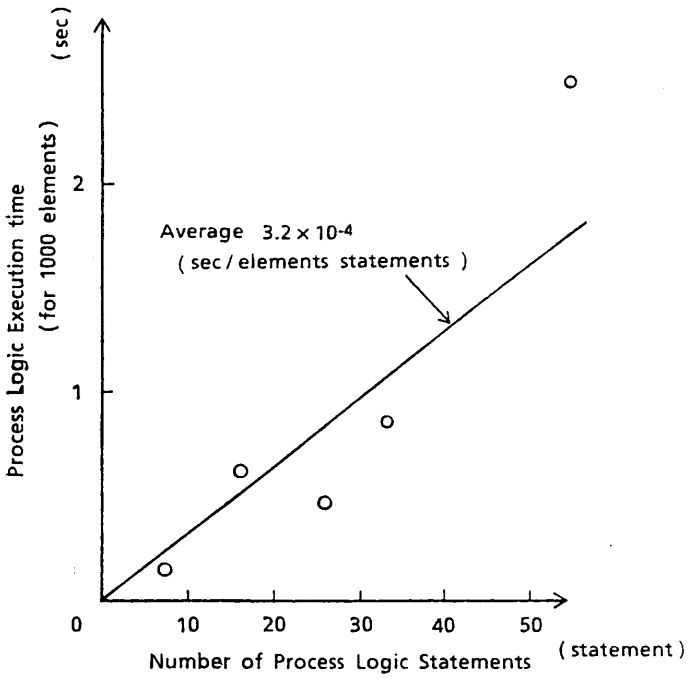


図 11 業務論理の実行時間
Fig. 11 Execution time of process logic.

象となるデータを評価する業務論理を手続き型プログラムに変換するプリコンパイラについて提案した．本プリコンパイラを組み込んだ知識型計画支援システムは，実際の生産計画業務に適用されており[6),7)]，処理内容の理解の容易さ，システムの開発，拡張，保守の容易化の効果を上げている．

謝辞　本研究の機会を与えていただいた(株)日立製作所システム開発研究所 川崎淳博士，本研究の推進にあたり御指導をいただいた同所 春名公一博士に深謝いたします．また，知識型計画支援システムの開発にあたり御指導，御協力をいただいた大森ソフトウェア工場の多くの方々に深謝いたします．

参考文献

1) Boehm, B. W.: Software Engineering, *IEEE Trans. Comput.*, Vol. C-25, No. 12, pp. 1226-1241 (Dec. 1976).
2) 薗田ほか：柔軟な計画を可能とする計画システム開発方式とその支援システム，第32回情報処理学会全国大会論文集，4 W-4 (Mar. 1986).
3) 川嶋ほか：柔軟な計画を可能とする計画システム開発支援システム用の業務論理記述言語とプリコンパイル方式，第32回情報処理学会全国大会論文集，4 W-6 (Mar. 1986).
4) 川嶋ほか：知識型計画支援システム向業務論理記述言語用プリコンパイラ，第33回情報処理学会全国大会論文集，3 W-5 (Oct. 1986).
5) 中田育夫：コンパイラ，産業図書，東京 (1981).
6) 山崎，冲中：鉄鋼業における AI の応用―生産計画システムに対する知識工学の応用―，日本機械学会誌，Vol. 89, No. 815, pp. 1194-1197 (Oct. 1986).
7) 冲中，山崎：鉄鋼業における生産計画への知識工学適用事例，第5回知識工学シンポジウム，pp. 45-50，計測自動制御学会 (Mar. 19, 1987).

(昭和62年4月1日受付)
(昭和62年6月11日採録)

川嶋　一宏（正会員）

昭和35年生．昭和57年東京理科大学理工学部機械工学科卒業．昭和59年同大学院修士課程修了．同年(株)日立製作所入社．現在同社システム開発研究所にて，離散型システムの計画・制御技法の研究に従事し，知識型計画支援システムに関する研究を担当．日本機械学会，IEEE各会員．

薗田　憲久（正会員）

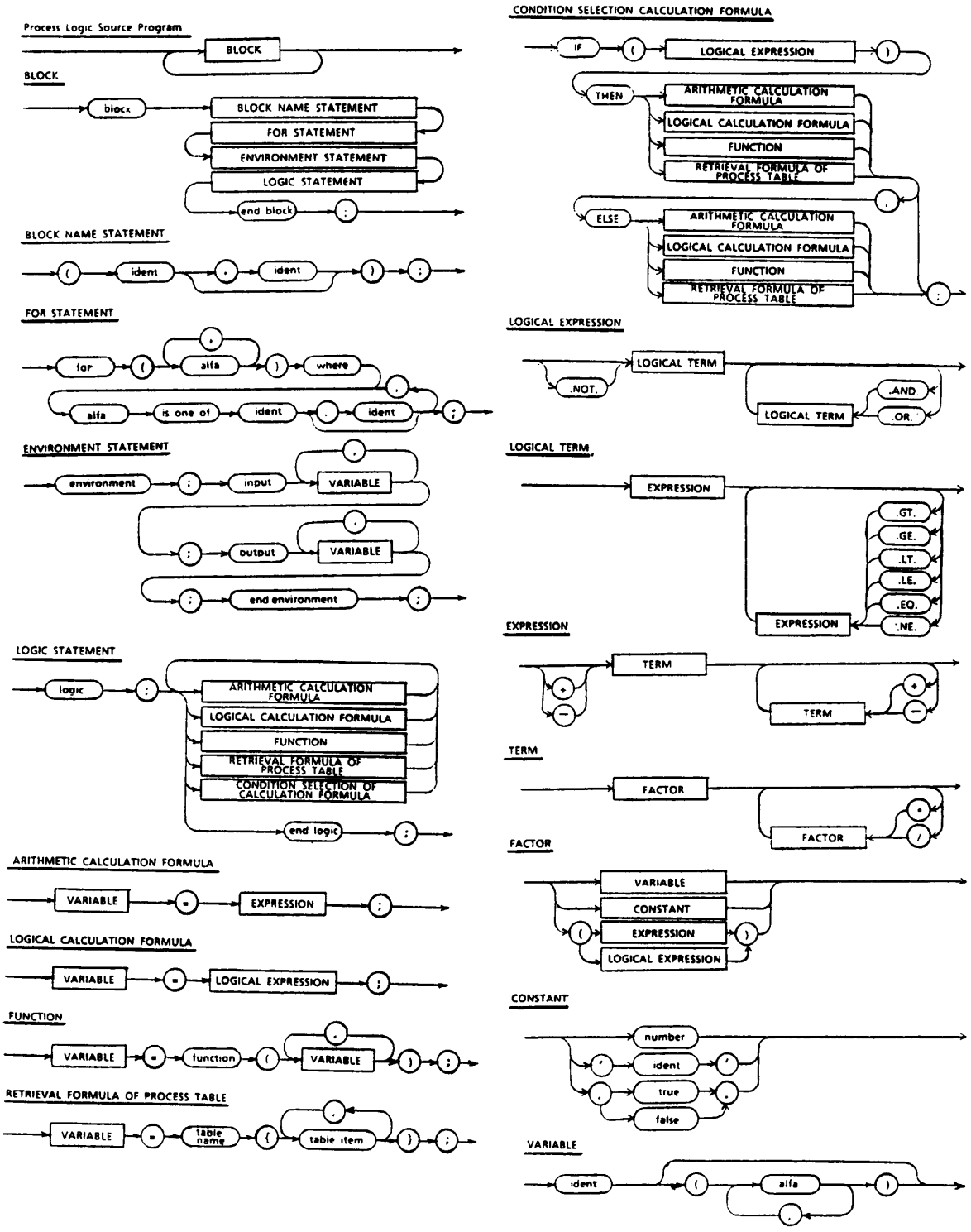
昭和25年生．昭和47年大阪大学工学部電気工学科卒業．昭和49年同大学院修士課程修了．同年(株)日立製作所に入社，システム開発研究所にてシステム計画技法，システム構造化技法，ペトリネット等事象駆動型システム，生産・流通業向情報処理システムなどに関する研究に従事．現在，同所第1部主任研究員．昭和56～57年UCLAに留学．工学博士．IEEE，電気学会，計測自動制御学会などの会員．昭和61年計測自動制御学会論文賞，昭和62年計測自動制御学会技術賞受賞．

原田　俊一（正会員）

昭和33年生．昭和57年九州大学工学部機械工学科卒業．同年(株)日立製作所入社．以来大型コンピュータを中心としたアプリケーションプログラムの設計・開発に従事．現在，同社大森ソフトウェア工場在籍．

三森　定道

昭和12年生．昭和36年京都大学工学部電気工学科卒業．昭和38年同大学修士課程，昭和41年同大学博士課程修了，共に，電子工学専攻．同年(株)日立製作所入社，中央研究所に配属．現在，同社システム開発研究所主管研究員．工学博士．この間，機械翻訳，スケジューリング手法，情報システムの高信頼化技術，OA，特に，非定形処理のための DB，会話言語，知識工学的アプローチによるソフトウェア生産・保守技術の研究に従事．IEEE，電気学会，日本オペレーションズリサーチ学会各会員．



付録 業務論理記述言語の構文グラフ

Appendix Syntax graph of process logic description language.