U.D.C. 〔800.92:519.685〕:〔681.32.06:159.95〕

# 知識処理言語“VOS 3 /HI-UX PROLOG”の高速化方式とプログラミング環境

## High Performance PROLOG Programming Environments

広瀬　正*　*Tadashi Hirose*
迫田行介*　*Kousuke Sakoda*
中尾和夫**　*Kazuo Nakao*
礒辺　寛***　*Hiroshi Isobe*

Prologは「事実」,「規則」の羅列だけでプログラミングできる宣言形言語であり，特に知識処理システムの記述に適しているが，実用的であるためには，その高速性及びプログラミング環境の充実が求められていた。

今回開発したVOS 3／HI-UX　PROLOGは，操作性の良いインタプリタに，応用プログラムの動作特性を分析してプログラムの高速化を行う最適化コンパイラを組み込み高速化を図っている。更に，他言語(Fortranなど)，データベースシステムとの結合機能，対話形デバッガ，日本語・グラフィック処理機能などにより，既存ソフトウェア財産を活用しながら効率良く知識処理システムが開発できる。既に設計支援エキスパートシステムなどの知識処理システム構築に適用されており，その有効性が確認されている。

## 1 緒　　言

知識処理システムでは，非数値的なデータを取り扱わねばならず，また推測・試行錯誤など，いわゆる「推論」と呼ばれる処理を行わねばならない。そのため，システム記述言語として記号処理が可能で推論機能を内蔵した述語論理形言語Prolog(Programming in Logic)が期待されている。

Prologでは，従来の手続き形言語のように計算の順序を逐一記述する必要がなく,「事実」と「規則」という宣言的な形式でプログラムが記述できる。このため，プログラム記述量は従来言語(Fortranなど)に比べて$\frac{1}{10}$程度で済み，プログラムの修正・変更も容易になる。一方，言語処理系がプログラムの実行順序を逐一判断しなければならず処理時間が大幅に増大する，プログラムの実行が試行錯誤的となりユーザーにとってプログラムの動きを把握するのが容易でない，などのProlog特有の課題を持っていた。また，Prologは，従来の手続き形言語と仕様体系が全く異なるため，従来言語と組み合わせてプログラム開発する場合，既存ソフトウェア財産を利用する場合に，従来言語及びデータがそのままでは使用できないという課題があった。

今回，これらの課題を解決する論理形言語として，高速実行が可能でかつ既存ソフトウェア財産を有効に活用してプログラム開発が可能なはん(汎)用大形計算機HITAC Mシリーズ上で稼動するVOS 3　PROLOGと，高度なマンマシンインタフェースが実現できるワークステーション2050上で稼動するHI-UX　PROLOGとを開発した。言語仕様は実質的国際標準である英国エジンバラ大学開発のDEC-10 PROLOG仕様に準拠している。更に，他言語結合機能，RDB(リレーショナルデータベース)結合機能，日本語処理機能，実数・配列データ形のサポート，対話形デバッガなどプログラミング環境を充実している。以下,Prolog言語及び開発したVOS 3　PROLOG並びにHI-UX　PROLOG(以下，VOS 3/HI-UX　PROLOGと言う。)の特徴及び実現方式について述べる。

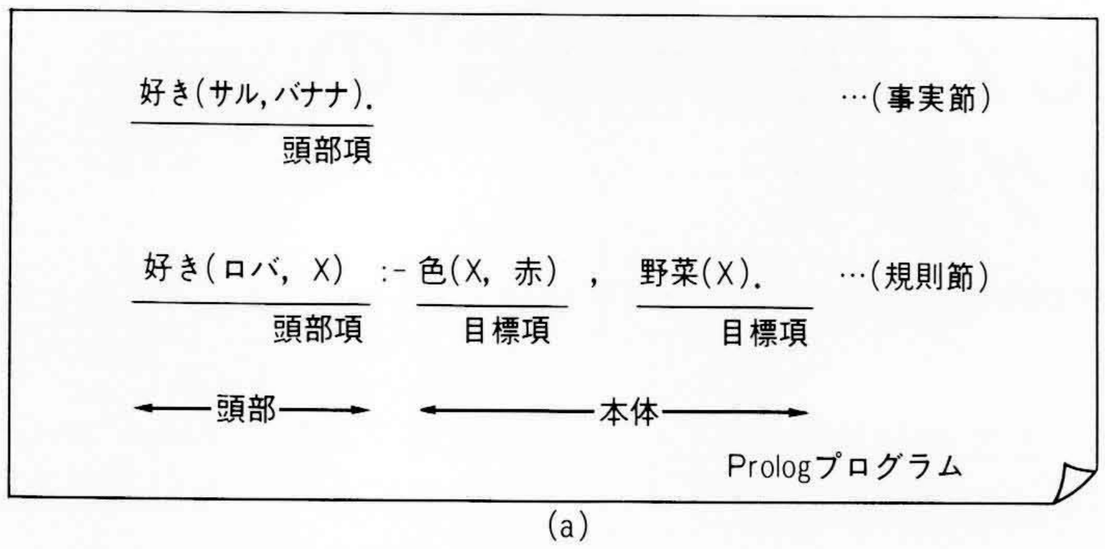
## 2 Prolog言語の特徴

Prologは1972年にフランス，マルセイユ大学のコルメラウア(Colmerauer)によって考案された言語である。その後，英国エジンバラ大学で本格的な処理系が開発され，更に1982年，我が国のICOT(新世代コンピュータ技術開発機構)が第５世代計算機の基本言語として採用するに至り，世界的に注目されている[1]。

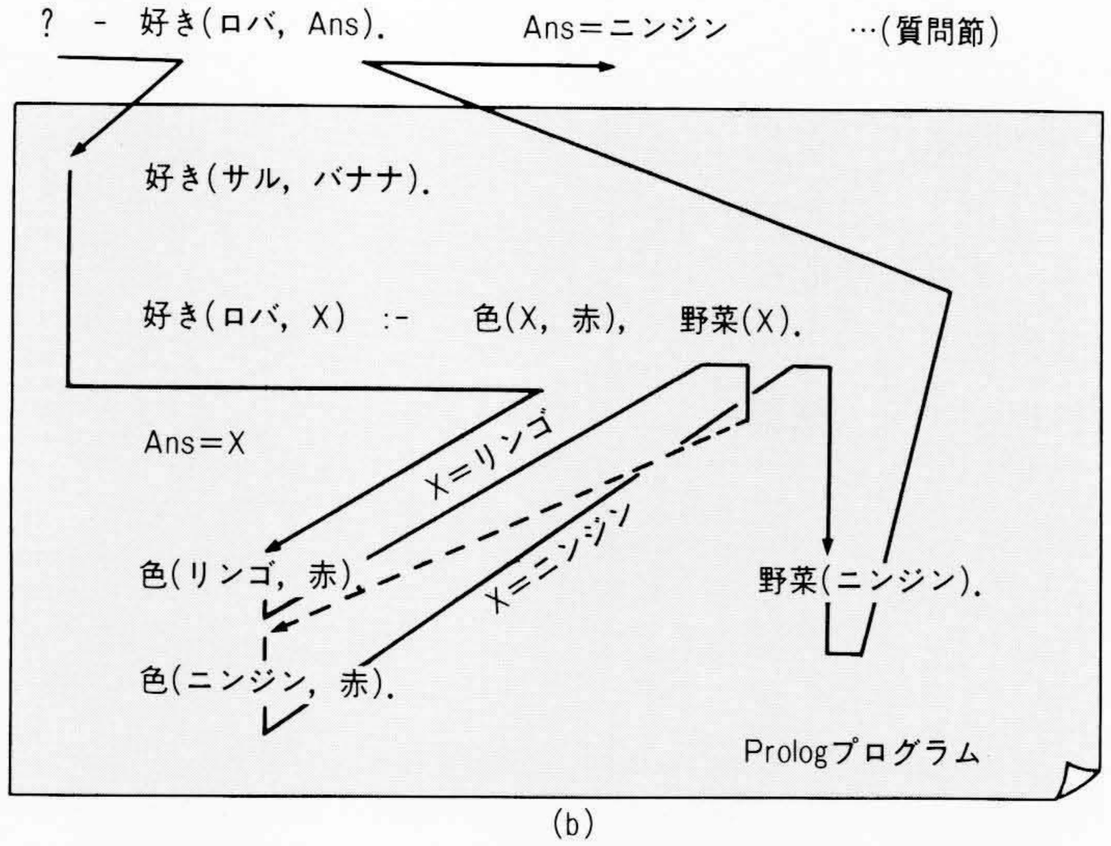
Prologでは，プログラムは事実節，規則節と呼ぶ節の並びによって表される〔**図１**(a)〕。規則節の「：-」記号の左側を頭部，右側を本体と呼ぶ。それぞれを構成する要素を頭部項，目標項と言う(Prologでは，頭部は一つの頭部項で構成される)。すべての目標項の成立が頭部項の設立条件である。事実節は頭部だけから成り，無条件に成立する項を表す。

プログラムの実行は，質問節を満たす事実節がプログラム内に存在するか否かを調べることである〔**図１**(b)〕。質問節とパターンマッチング(ユニフィケーション)可能な頭部を持つ節をプログラムの先頭から順に探す。事実節と一致すれば「成功」である。規則節と一致した場合は，更にその本体の目標項のそれぞれについてユニフィケーション処理を繰り返す。一致する節が存在しない場合は，以前に選択した節まで後戻

* 日立製作所システム開発研究所　** 日立製作所システム開発研究所 工学博士　*** 日立製作所ソフトウェア工場




(a)




(b)

注：⟶ ユニフィケーション実行，----→ バックトラッキング実行

図1 Prologプログラムの例 (a) プログラムは事実節と規則節の並びから構成される。(b) 実行は質問節から始まり，ユニフィケーションとバックトラッキングを繰り返し，解を求める。

り(バックトラッキング)し，別の節とのユニフィケーションを試みる。

以上述べたように，Prologで記述されたプログラムの実行は，ユニフィケーションと呼ぶパターンマッチング機能と，バックトラッキングと呼ぶ解探索のための試行錯誤機能の二つの基本機能で実現される。

Prologは，上述の基本機能に加えて，メタプログラミング機能と呼ぶ知識処理システムの記述に有効な機能を持つ。メタプログラミング機能とは，実行中のプログラムを動的に変更・修正する機能を言う。Prologプログラムは，本来，独立性の高い節(事実節と規則節)の並びから構成されているので，実行中にプログラムの書換えが混乱なく行える。本機能により，従来言語では記述が容易でなかった学習機能を持つプログラムなどが簡単に記述できる[2)]。

これらの特徴を持つPrologを，実用的な知識処理システムの構築に適用するためには，更に，

(1) 既開発のプログラムやデータの利用を可能にする従来言語・データベースとの結合機能の実現

(2) プログラム開発環境の充実

(3) 妥当な実行性能

が求められる。VOS 3/HI-UX PROLOGは，これらの要求を満たす機能及び性能を提供することを目標に開発したProlog処理系である。以下，本処理系の高速化技術を**3**章で，プログラミング環境を**4**章で述べる。

## 3 高速実行方式

一般にPrologプログラムの実行速度は従来形言語プログラムと比較して遅い。その理由は，プログラムが実行中に変更される可能性があるために，解釈実行という形態を採る必要があることや，木構造，定数，変数などの多種のデータ形を実行時に判定する必要があることなどによる。両機能は，いずれもプログラムの記述性を高める反面，実行時性能を低下させる原因となっている。

VOS 3/HI-UX PROLOGでは，(1) プログラム実行時の動作特性に関する情報を，プログラミング時点あるいはコンパイル時点で収集し，(2) 収集した動作特性情報に基づいて，プログラムの中で実行中に変更されない部分を高速実行できるように部分コンパイルするなど，プログラムに各種の最適化処理を施し，高速化を図っている。実際のPrologプログラムの多くの部分は実行中に変更されず，使用するデータの形式も限定されているので，本アプローチは実用プログラムの性能向上に実質的に寄与する。

VOS 3/HI-UX PROLOGのシステム構成を**図2**に示す。プログラムは，木構造データ形式あるいは機械語オブジェクト形式のいずれかの形で，内部データベースに格納される。木構造データ形式で格納されたプログラムは，インタプリタにより解釈実行される。処理速度は遅いが，動的なプログラムの変更が可能である。機械語オブジェクト形式で格納されたプログラムはインタプリタにより起動され，機械語オブジェクトが直接実行される。本機械語オブジェクトは，最適化コンパイラによりあらかじめ不要な処理が取り除かれており，動的なプログラムの変更はできないが処理速度は速い。

コンパイル時には，次の最適化技術を用いている。

(1) 節インデクシング

Prologプログラムの処理実行時間の大半は，ユニフィケーション可能な項を頭部に持つ節の選択に費やされる。節選択の高速化のため，あらかじめすべての節の頭部項内の引数に関する属性情報を収集し，これに基づき節のグループ化を行っておき，実行時にはグループ内の節だけを調べればよいようにする。この方式を節インデクシングと呼ぶ。選択されるべき節が，どのグループに属するかを高速に調べるために，ハッシュ表を利用する〔**図3** (a)〕。

(2) 木構造の事前展開

節選択処理の中で，特に頭部項に含まれる木構造データのユニフィケーションに時間がかかる。木構造の各ノードを順に探索し，ユニフィケーション処理を繰り返さねばならないからである。ノード探索の高速化のために，あらかじめデータの構造を解析しておき，これに基づき各ノードを探索順に展開し高速化を図る技法を木構造の事前展開と呼ぶ〔**図3** (b)〕。事前展開により，はん用計算機のパイプライン機構を有効に活用できる機械語オブジェクトが得られる。

ほかに，以下の最適化技法も組み込んでいる。

(1) 冗長スタック処理削除(バックトラッキングしない節に対

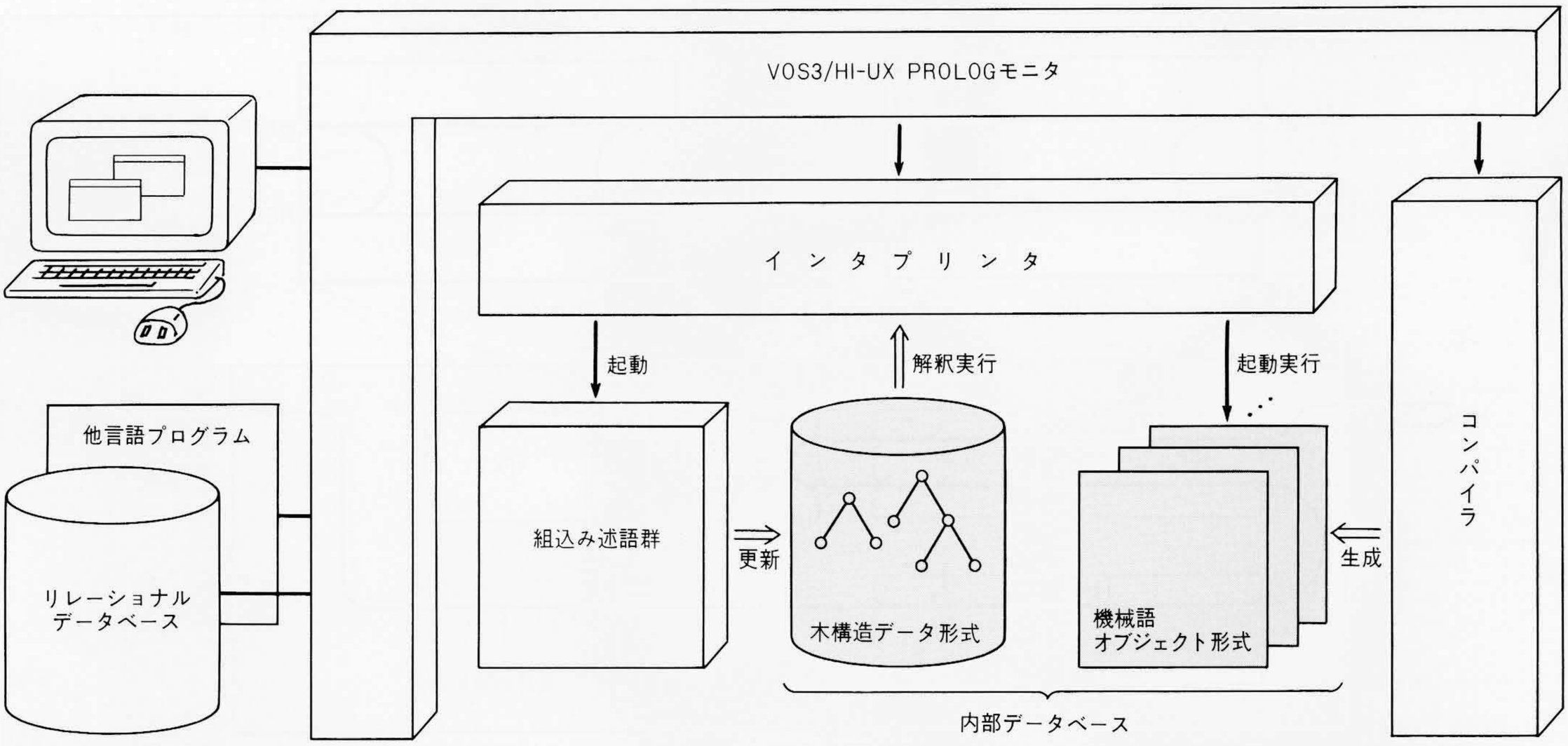



図2　VOS 3/HI-UX PROLOGのシステム構成　実行プログラムは，変更容易な木構造データ形式と機械語オブジェクト形式で内部データベースに格納される。

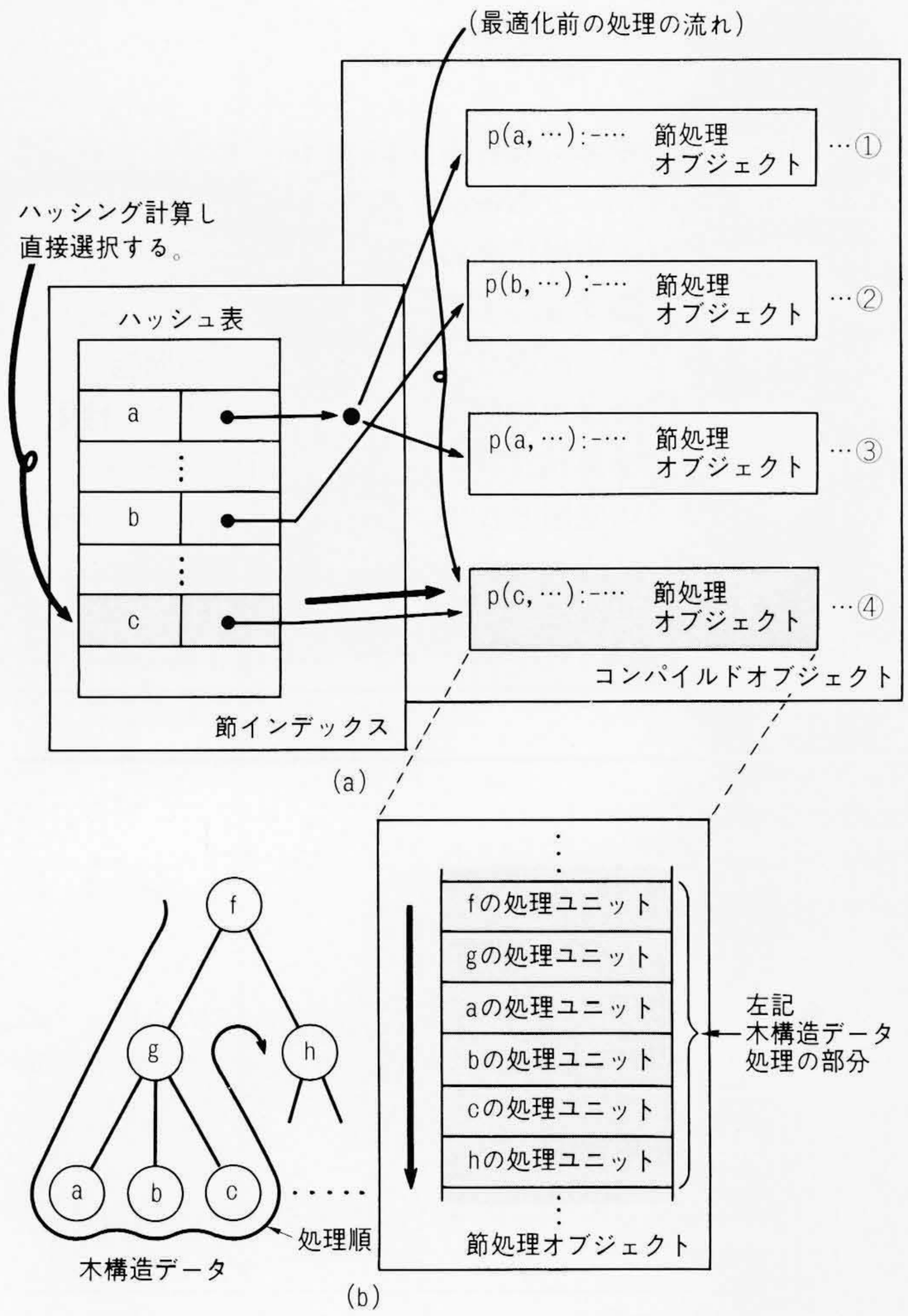



図3　コンパイル時最適化技法　(a) 節インデクシング：ハッシュ表により選択すべき節の属するグループを選ぶ。他グループの節(①，②，③)は調べる必要がない。(b) 木構造事前展開：木構造データの処理は，あらかじめ処理順に展開し高速化を図る。

する実行状態管理の省略)

(2)　データ形判定の簡略化(ユニフィケーション対象データのデータ形の宣言に基づき，データ形判定処理を簡略化)[3]

(3)　組込み述語のインライン展開(算術演算など出現頻度の高い組込み述語処理を機械語オブジェクトに展開)

(4)　レジスタ割当て最適化(レジスタの最適利用によるメモリアクセス回数の削減)

(5)　節の直接呼出し(動的に変化しないプログラム間での呼出しオーバヘッドの削減)

これらの最適化技術の適用の結果，はん用大形計算機HITAC M-680H上で最大実行性能700k LIPS※1)を達成した。

## 4　プログラミング環境

### 4.1　デバッガ

Prologプログラムの実行は，バックトラッキング機能により試行錯誤的となるので，ユーザーにとってプログラムの動きの把握が難しい。VOS 3/HI-UX　PROLOGでは，バックトラッキング時もプログラムの動きを容易に把握できる表示形式を導入し，作業効率の良いデバッグ環境を実現している。

(1)　「Clauseモデル」によるプログラム実行状態表示[4]

従来，プログラム動作理解のために「Boxモデル」が提案されていた[5]。「Boxモデル」は，述語※2)呼出し(CALL)，成功終了(EXIT)，再呼出し(REDO)，失敗終了(FAIL)の4種のプログラム制御点を逐次表示する動作理解法である。4種の制御点の表示により，述語単位でのマクロな実行順序の把握はできるが，ユニフィケーションによる節選択の過程，すな

※1)　LIPS：Logical Inference Per Secondの略で，1秒間に実行される成功ユニフィケーション回数を言う。

※2)　述語：同種の頭部を持つ事実節，規則節の並びを述語と呼ぶ。

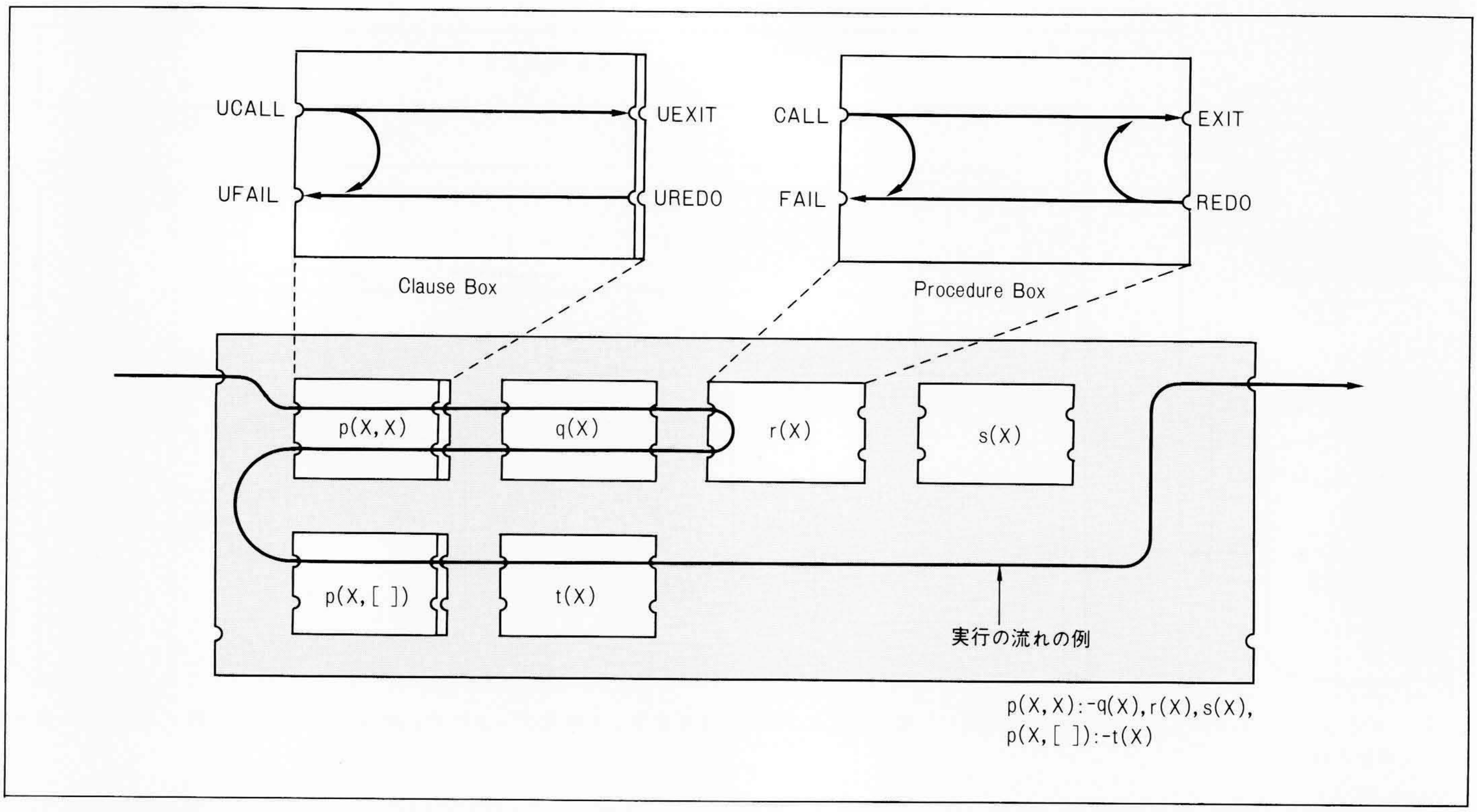



**図4 Clauseモデルによるプログラム動作の理解** 各節をClause Box, Procedure Boxの並びとみなし，各々の四つの制御点の通過順序で動作の流れを理解する。

わちミクロな実行順序の把握ができず，プログラムの誤りの摘出を困難にしていた。

VOS 3/HI-UX PROLOGでは，節選択の過程も把握できる「Clauseモデル」を導入した(**図4**)。「Clauseモデル」は，各節の頭部(Clause Boxと呼ぶ。)に対する4種の制御点(UCALL, UEXIT, UREDO, UFAIL)を表示するものである。これにより，節選択のミクロな動きについても明確に把握することが可能となる。

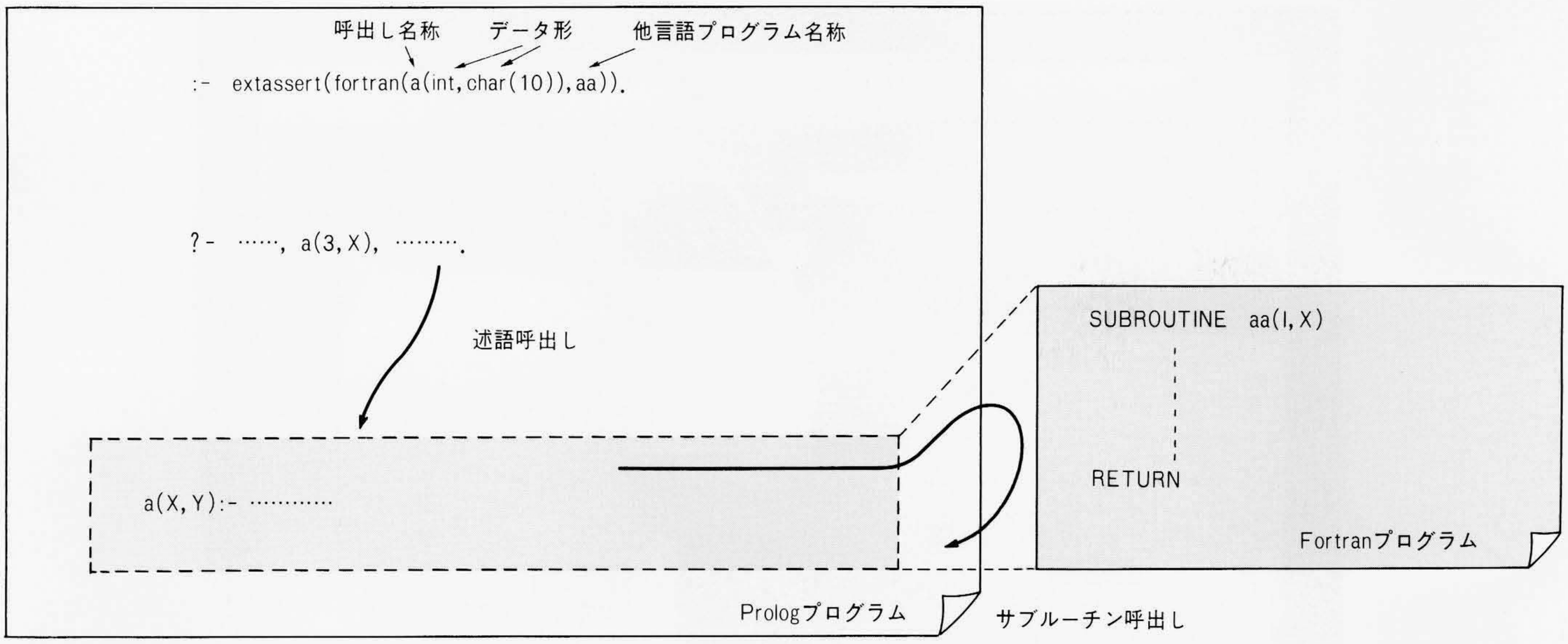
(2) 操作性の良いデバッグコマンド(**表1**)

以下に述べるデバッグコマンドを提供している。これらを用いてプログラムを実際に動作させながら，プログラム不良部分を効率的に摘出することができる。

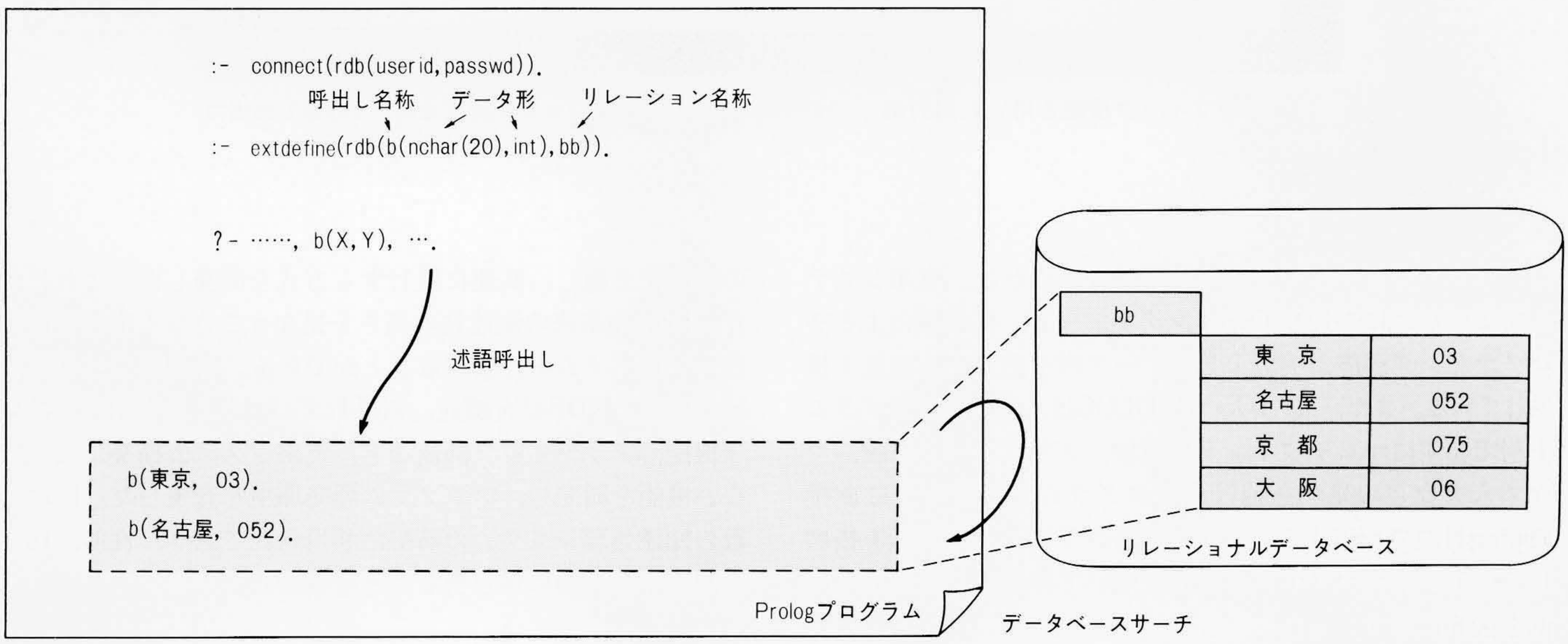
(a) プログラムを一制御点ずつ進めるステッパ機能
(b) 特定の述語だけの動作を追うブレークポイント機能
(c) 任意の部分を再試行する後戻り再実行機能
(d) オペレータの意図どおりに処理を進める強制実行機能

表1 デバッグコマンド一覧 デバッグコマンドは，ユーザープログラムの各制御点で投入でき，対話的なデバッグを効率良く進めることができる。

| 種別 | コマンド | 処理 |
|---|---|---|
| 表示・トレースコマンド | c, C | 実行中のプログラム(節)の表示 |
| | p, P | 実行中のプログラム(述語)の表示 |
| | g, G | 呼び出しているプログラム(節)の表示 |
| | (指定せず) | 次の制御点へ進む。 } ステッパ機能 |
| | n, N | 次の同一節内の制御点まで進む。 } ステッパ機能 |
| | * | 次の指定述語の制御点まで進む。 } ブレークポイント機能 |
| | j, J〔番号〕 | 指定識別番号の述語呼出しまで進む。 } ブレークポイント機能 |
| 実行制御コマンド | r, R | CALL制御点へ戻る。 } 後戻り再実行機能 |
| | ; | REDO制御点へ戻る。 } 後戻り再実行機能 |
| | s, S | EXIT制御点へ進む。 } 強制実行機能 |
| | f, F | FAIL制御点へ進む。 } 強制実行機能 |
| ユーティリティコマンド | @ | 任意のユーザー部分プログラムの実行 |
| | a, A | 処理中断 |
| | e, E | 処理中止(システム終了) |
| | h, H | デバッグコマンドの説明表示 |




(a)




(b)

図5　他言語プログラムやデータベースとの結合　(a) Fortranで記述されたプログラムがPrologプログラムの規則節と同様に取り扱える。(b) リレーショナルデータベース内のデータをPrologプログラムの事実節と同様に取り扱える。

### 4.2　他言語プログラムとの結合[4)]

VOS 3　PROLOGは，Fortran, Cobol, PL/I言語で記述されたプログラムを，HI-UX　PROLOGはC言語で記述されたプログラムを，Prologプログラムと同一形式のインタフェースで呼び出すことができる〔**図5** (a)〕。他言語プログラムとの結合は，Prologプログラムからの呼出し名称と，他言語プログラムの名称及び引数のデータ形をあらかじめ宣言しておく(extassert述語)だけでよく，他言語プログラムを変更する必要は全くない。このような単純な記述で，他言語で記述された既存のプログラムをそのまま活用することができる。数値計算部分は，数値処理に適したFortran言語を用いて記述するなど，プログラムの一部分をそれに向いた言語を用いて作成することもできる。

### 4.3　データベースとの結合[6)]

VOS 3　PROLOGはRDBに格納されているデータを，Prologプログラム内の事実節と同様に取り扱うことができる〔**図5** (b)〕。データベースとの結合はRDBオープン処理(connect述語)の後に，Prologプログラムからの呼出し名称とRDB内データのリレーション名称及びフィールドのデータ形を宣言しておく(extdefine述語)だけでよい。これにより，具体のデータ参照・追加・削除処理がPrologプログラムの事実節の参照・追加・削除処理と全く同じ形式で行える。

### 4.4　言語仕様の拡張

VOS 3/HI-UX　PROLOGの言語仕様は，実質的な世界標準とみなされている英国エジンバラ大学開発のDEC-10 PROLOG仕様に準拠している。更に，上述のデバッガ，他言

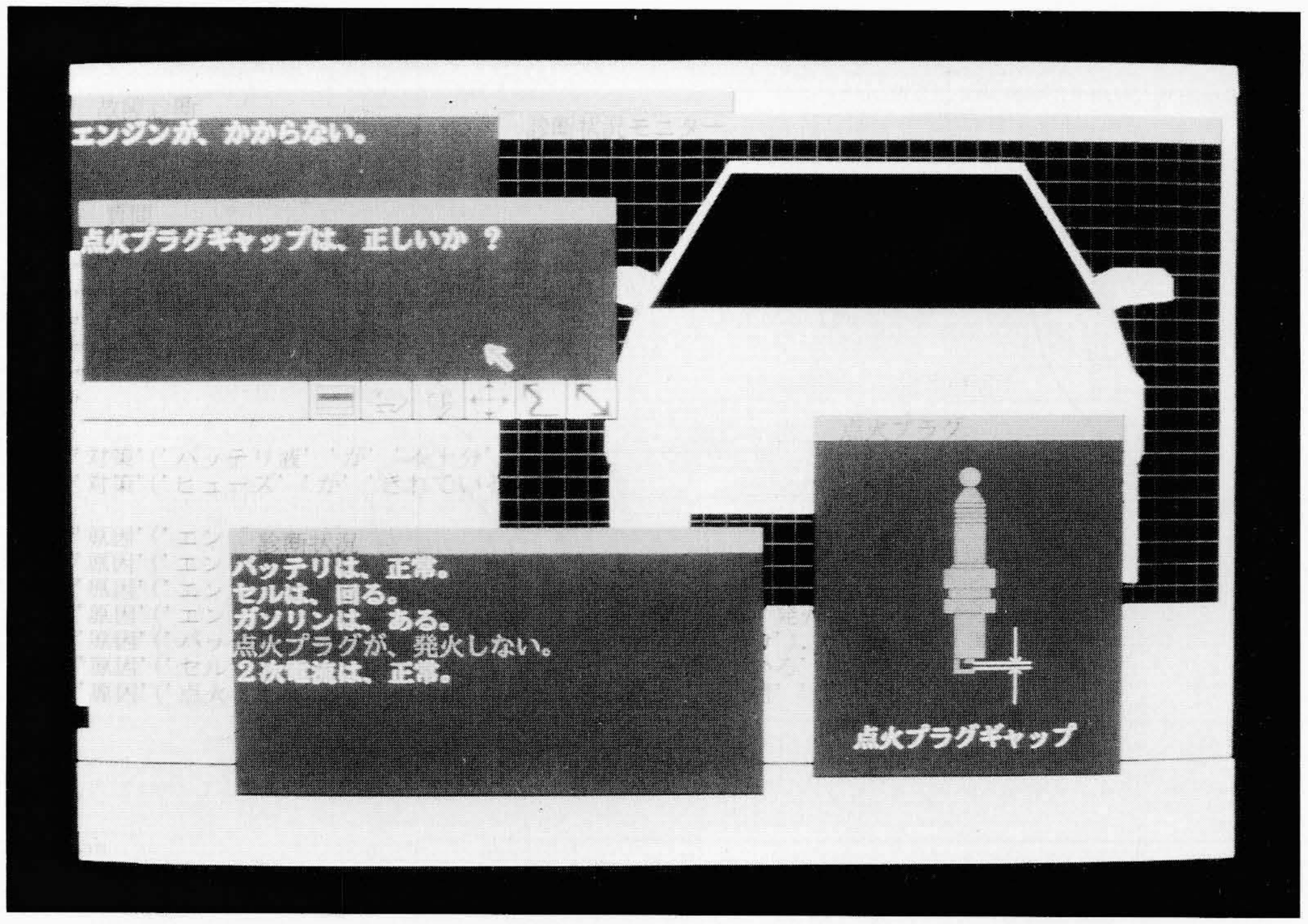



図6 グラフィック機能を用いた出力例 HI-UX PROLOGのグラフィック機能を用いて作成した出力例を示す。

語・リレーショナルデータベース結合のほかに，実用システム構築のために必要なTSSインタフェース，日本語によるプログラミング機能，実数・配列データ形を記述する機能を提供している。また，HI-UX PROLOGは，他言語プログラムを利用せずにマルチウインドウ操作，グラフィック描画，マウス入力などの処理を他言語プログラムを利用せずに直接Prolog言語で記述することができるグラフィック機能を提供している。本グラフィック機能を用いて作成した画面表示の例を**図6**に示す。

## 5 適用例と効果

VOS 3/HI-UX PROLOGは，既に設計支援，故障診断などのエキスパートシステム[7]，自然語処理システム，生産管理システムなど広い分野への適用をみており，その実用性が確認されている。従来形言語で記述した場合と比べて$\frac{1}{10}$程度の記述量でプログラムが作成できる[8]，プログラムの修正・変更が容易であるなど，開発，保守の面でも従来言語に比べて顕著な効果があることを確認している。また，実用プログラムの開発では，Prologだけで構築できるシステムは皆無で，他言語プログラム，外部データベースを含めて統合的にシステムを開発することが必要であり，これらとの結合機能が多用されていることを確認している。

## 6 結 言

以上，知識処理言語VOS 3/HI-UX PROLOGの機能と高速実行方式について述べた。VOS 3/HI-UX PROLOGは，動的に変化しないプログラム部分を部分的にコンパイルするとともに，動作特性を事前に解析し，この情報に基づき各種の最適化を施し，高速に実行する方式を開発した。これにより，Prolog本来の記述性の高さを損なうことなく実行性能の良いプログラムを作成することができる。機能面では他言語結合機能，RDB結合機能，対話形デバッガ，日本語・実数処理機能，グラフィック機能など，実用システム構築に欠かせない機能を開発し，プログラム開発環境を充実した。既に多数の知識処理システムの構築に利用されており，性能，機能両面で十分実用に供し得ることを確認している。

### 参考文献

1) Robinson, J.A.: Logic Programming—Past, Present and Future—, New Generation Computing, Vol. 1, No. 2 (1983)
2) Bowen, K.A.: Meta-Level Programming and Knowledge Representation, New Generation Computing, Vol. 3, No. 4 (1985)
3) 竹内，外：論理型言語処理系「LONLI」における最適化コンパイル方式の提案と評価，情報処理学会第32回大会 3F-6 (1985)
4) 広瀬，外：論理型基本処理系「LONLI」の機能と効果，情報処理学会第29回大会 4P-10(1985)
5) Clocksin, W.F. et al.: Programming in Prolog, Springer-Verlag(1981)
6) 山口，外：論理型言語処理系「LONLI」の開発—RDBインターフェース—，情報処理学会第31回大会 5M-5(1985)
7) Watanabe, et al.: Knowledge-Based System Optimal IIL Circuit Descriptions, 23rd ACM/IEEE Design Automation Conference, 6 (1985)
8) 広瀬，外：知識情報処理のための高機能言語「LONLI」，Computer Design **2**, 2 (1985)