拡張型言語システム PROTO-E* の設計と評価†

花田収悦†† 古山恒夫††
稲田 満†† 馬場康彦†† 徳田憲夫††

拡張型言語を実用の言語処理プロセッサの作成に適用するために開発した新システム PROTO-E の基本方式，言語仕様とプロセッサの概要およびその評価結果について述べる．

PROTO-E システムは，言語処理プロセッサ（コンパイラやコンバータ等）が処理する言語（入力言語と出力言語）の規則を入力して，それを処理するプロセッサを生成するシステムであり，次の特徴を持つ．

① PROTO-E 言語は，任意の高級言語の拡張を目的とした超言語である．②拡張された新しい言語の構文と意味を形式的に定義するための豊富な機能を持つ．意味づけは，入力言語に対応した出力言語を定義することにより行う．③構造体風の簡明な記述形式である．④ PROTO-E プロセッサは，構文と意味の規則を表形式に作成し，それを解釈実行する方式である．⑤試行錯誤の回数を減らす構文解析法を用いている．

記述実験等による本システムの評価により，① PROTO-E は，実用の言語を定義するのに十分な機能を持つ．②生産性は，手作りの場合** に比較して約2倍である．③作成した言語処理プロセッサの性能は，処理速度が手作りのコンパイラの約 1/5，また翻訳可能入力行数が約 7,500 行であり十分に実用に耐える，ことを確認した．

1. まえがき

計算機の適用領域が益々拡大される現状において，新たな領域に必要な機能や利用者の使用目的に応じた使いやすい表現を容易に追加，拡張できる柔軟性のあるプログラミング言語の提供が望まれる．これを実現するものとして拡張型言語がある．その概念は，拡張の元になる核言語とそれを拡張するための言語定義機能を設けることによって言語を拡張させるものである．

拡張型言語としてすでに多くのものが提案されている[1)~8)]．これらの拡張型言語の方式は多岐にわたり，いくつかの試みにも拘らず[9),10)]，十分な体系化がなされているとは言えない．またそれらの多くは，融通性をねらいとした拡張方式の研究に主体が置かれているために具体的な適用領域や拡張された言語を入力するプロセッサの性能等について実用性が低い．このような状況において，拡張型言語の問題点を次のように捉えることができる．

拡張型言語の問題点……拡張型言語が核言語と拡張機能から構成されるとした場合，核言語の組み合わせによる拡張では本質的に新しい機能の追加はできず適用領域が限定される．また機能追加を行うためには通常，核言語より低水準なほかの言語による何らかの記述が必要となり，利用者の負担が大きい．

そこで筆者等は，次の解決策を採用することにより，汎用かつ実用のシステムをねらいとした拡張型言語システム PROTO-E を開発した．

① 高級言語の拡張

拡張の対象を任意の高級言語とし，拡張型言語の機能を言語定義機能に限定する．すなわち核言語と拡張機能を分離し，任意の高級言語の拡張を行うのに十分な言語定義機能を実現する．

② 構文と意味の形式的定義

拡張された新しい言語の構文と意味を形式的に定義させ，翻訳の過程を記述させない．意味の定義は，新しい言語に対応した出力言語を定義することにより行う．出力言語として任意の高級言語を選択可能とする．

③ 汎用の翻訳機能

プロセッサは，構文と意味の定義を入力して翻訳規則の表を作成する機能とその表を参照しながら任意の言語の構文解析と意味づけを行う機能から構成する．

④ 実用的な構文解析法

広い適用領域と記述性および処理速度の向上を実現する構文解析法を採用する．

本論文では，PROTO-E の基本方式，特徴および概要を述べ，ひきつづき言語仕様，プロセッサの性能の

† Design and Evaluation of the Extensible Programming Language System PROTO-E by SHUETSU HANATA, TSUNEO FURUYAMA, MITSURU INADA, YASUHIKO BABA and NORIO TOKUDA (Processing Programs Section, Yokosuka Electrical Communication Laboratory, N. T. T.).

†† 日本電信電話公社横須賀電気通信研究所処理プログラム研究室

* PROgramming TOol-Extensible language

** 既存のシステム記述用言語で記述する場合

両面から評価と分析を行い，本言語拡張方式の有効性を示す．

2. 方式選定と特徴

2.1 基本方式の選定

一般にプログラミング言語の拡張方式は，言語の表現（構文等），意味（概念）およびそれらの対応づけ（翻訳方法）を，データやその処理等の各言語要素ごとにどの程度かつどのような記述形式で追加，変更するかによって分類される．拡張の程度と拡張の形式は，それぞれ定義言語の機能（記述能力）と表現（記述形式）として実現される（**図1参照**）．

これらの分析結果に基づき**表1**のような基本方式を整理した．一般に各方式の特長は対立する代替案の欠点となるため，評価基準として実用性を重視して①記述能力，②記述性，③性能の優先順位を設定して方式選定を行った．

次いで，基本方式をもとに設計の詳細化を進めるとともに，平行して既存の高級言語に対する記述実験とプロセッサの試作を行って実用上の問題点を抽出し，それを PROTO-E の開発に反映させた．問題点と対策を**表2**に示す．

2.2 PROTO-E の特徴

PROTO-E の特徴および言語の性質を次に示す．

（1） 記述の対象

① PROTO-E 言語は任意の高級言語を拡張するた

表1 PROTO-E の設計目標と基本方式

Table 1 Objectives and the principles of designing PROTO-E.

設計目標	設計要因	基本方式	特長	備考
記述能力	基本言語の構成	問題記述能力あり	定義せずに使用可能	
		＊問題記述能力なし	自由な拡張可能性	
	表現の記述程度*1)	＊文脈自由言語	広い適用領域	*1) 構文解析法に依存する．
		決定文脈自由言語	容易な処理	
	意味の記述程度	＊任意の高級言語	自由かつ容易な意味記述	*1) コンパイラ・コンパイラの範疇に属する．
		任意の中間言語やアセンブラ*1)	自由かつ豊富な意味記述	*2) 基本言語として問題記述能力のある構成法をとった場合は一般にここに属する．
		特定の高級言語*2)	容易な意味記述，定義せずに使用可能	
		特定の中間言語やアセンブラ*2)	豊富な意味記述，定義せずに使用可能	
生産性	記述方式	手続的記述	多様な変換処理記述が可能	
		＊非手続的記述	変換機能のみの把握	
	表現形式	ANF 記法*1)風	行数の削減	*1) 文献 7) を参照のこと
		＊構造体データ風	容易な構造的把握	
性能	構文解析法	直構文解析法	広い適用領域	*1) 本方法の概要については本文 3.3 を参照のこと
		＊制限付き直構文解析法	広い適用領域と性能向上	
		確定解析法	性能向上	

注）基本方式の欄の＊印は PROTO-E で採用した基本方式

表2 PROTO-E 設計上の問題点と対策

Table 2 Problems occuring during designing PROTO-E and the means to solve them.

問題点	具体例	対策
固定形式言語に対する記述能力	・COBOL 言語における正書法の記述ができない ・FORTRAN 言語における文番号の認識ができない	物理的位置の表現機能
文脈依存仕様に対する記述能力	・ホラリス型定数の記述ができない ・データ属性の定義とその参照ができない	変数参照機能 構文間の情報授受機能
論理的に意味のない文字列に対する記述効率	・FORTRAN 言語の空白の記述が煩雑である ・PL/I 言語の注釈の記述が煩雑である	論理的に意味のない部分の削除機能（注釈指定）
条件判定や演算を要する処理に対する記述能力	・PL/I の構造体データの記述ができない ・目的プログラムのインデンテーションが行えない	条件判定機能 演算処理機能
定義された言語プロセッサの処理速度	・試作したプロセッサの処理速度がコンパイラの 1/30 程度である	選択表 文字列照合の高速化

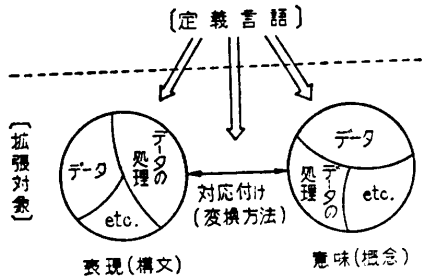


図 1 プログラム言語の拡張方式

Fig. 1 Concept of the extension mechanism in the programming language.

めの言語定義機能を持ついわゆる超言語であり，数値計算等の特定の問題を記述する能力はない．問題記述のための表現と意味は PROTO-E プロセッサが入力および翻訳出力する言語によって定められる．

② 文脈自由言語と一部の文脈依存言語の和集合を入力言語として定義できる．文脈依存言語の範囲は，入力言語の文字列を左から右に向かって走査して構文解析を行う任意の時点において，すでに解析された部分の参照である左文脈と先読み指定 (3.2 節参照) による限られた範囲の右文脈のものである．

③ プログラミング言語のデータ属性，演算子，文および制御構造について自由な表現を定義できる．

④ 入力言語および出力言語のプログラムは，それぞれ構文定義と意味定義により1つの木構造に組み上げられるものでなければならない．((4) 項参照)

(2) 言語機能

構文の形式的定義は，バッカス記法 (BNF) やそれを拡張した ALGOL-N 記法 (ANF) 等で実現される．PROTO-E では ANF をさらに拡張し，文脈に依存した部分を含めた意味の形式的定義と BNF 系では表現できないものの定義を可能とした．

① 超記号として超結合子* が表現する状態を保持する超結合子変数を導入した．

② 超変数，超結合子変数の参照機能，構文解析で作成される木構造の探索機能，超変数，超結合子変数，文字列，数値をオペランドとする超演算機能を持つ．

③ プログラム文字列の物理的位置を表現する機能，注釈や区切り記号を効率的に定義する機能，複雑あるいは特殊な翻訳を可能とするオウン・コーディング機能を持つ．

```
3  超変数
3  超結合子 (超結合子変数)
4  超定数
4  超変数
4  超結合子 (超結合子変数)
5  超変数
```

図 2 PROTO-E 言語による構造体風の定義形式

Fig. 2 Structured notation in PROTO-E language.

(3) 記述形式

① 超言語による言語の定義では，言語構造を階層的に捉えて構文の要素ごとに記述する．PROTO-E ではその際，構文と意味を対にして定義する．これにより翻訳の対応づけをとる．

② 構文定義として SYNTAX データと呼ぶものを記述する．SYNTAX データは入力言語の構造を規定する．また意味定義として BY データと MSG データを記述する．BY データは構文の適用* に対応して出力する言語の構造を規定する．MSG データは構文の適用に応じて印刷出力するメッセージを記述する．

③ 一般に BNF による言語定義では記述量が多くなる．また ANF では，超結合子による表現が連続したり，入れ子構造になり記述性，読解性が悪い．

PROTO-E では，定義の詳細をレベル番号を伴った PL/I 言語の構造体データ風の形式で記述する．これにより超結合子変数等を導入した構文と意味の形式的定義において簡明な表現を可能とした．(図2参照)

(4) 翻訳の方法

翻訳は構文解析，意味づけおよび入出力編集の段階から成る．構文解析では入力言語すなわち原始プログラムを構文定義にしたがって解析し，その構造を1つの木構造の形で表現する．意味づけでは構文解析で作成された木構造上で，意味定義にしたがって節や各種の情報を置き換えて新たな木構造を作成する．入出力編集では原始プログラムの入力，意味づけで作成された木構造から目的プログラムへの編集と出力，各種の印刷リストの出力を行う．

3. システムの概要

3.1 プロセッサの構成

* 超言語は超定数，超変数および超結合子と呼ばれる超記号を持つ．超定数，超変数はそれぞれ言語の基本記号と構文要素を表す．超結合子は超定数や超変数の省略，選択，繰り返し等の状態を表現する．

* 構文の適用とは，構文解析時の木構造の作成において，その構文定義を参照することによって原始プログラムのある部分の文字列を解析し，木構造に1つの節を追加することを意味する．

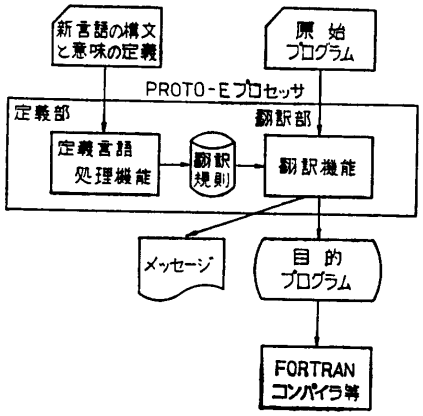


図 3 PROTO-E プロセッサの構成
Fig. 3 Construction of PROTO-E processor.

PROTO-E プロセッサは，大きく定義部と翻訳部の機能に分けられる．定義部は PROTO-E 言語で記述された新しい言語の構文と意味の定義を入力し，それを処理してファイル化した内部形式の表を作成する．翻訳部は，定義部で作成された翻訳規則の表を参照しながら構文解析と意味づけを行うものであり，構文定義にしたがった原始プログラムを入力し，それを意味定義にしたがった目的プログラムに翻訳して出力する．PROTO-E プロセッサの構成を図 3 に示す．

3.2 言語機能

PROTO-E 言語の主要な機能を示す．(以下，(1)～(4)については図 4 を参照)

(1) 超結合子

類似の構文をまとめて定義できる機能であり，記述量の削減に役立つ．

① 省略指定：OPT (# n)

構造体の要素が省略可能であることを示す．

② 正順選択指定：ORDER (# $n(l:m)$)

構造体の要素のいくつかが選択され，それらが記述順に存在することを示す．

③ 繰り返し指定：ITER (# $n(l:m)$)

構造体に含まれる部分が繰り返されることを示す．

n で表したものは超結合子変数であり，省略，選択の状態や繰り返し回数が設定される．l と m は，選択の個数や繰り返し回数の上下限値を指定するパラメータである．

```
構文定義名KAGENの定義
  SYNTAXデータ
    0 KAGEN ---------- 1
    1 SYNTAX ---------- 2
    2 CMT(COM) ---------- 3          ――注釈指定 (25～28 参照)
     3 $1 VARIABLE ---------- 4     ――別の構文定義 VARIABLE で解析する
     3 "ノアタイハ" ---------- 5    ――原始プログラム中の文字列「ノアタイハ」を解析する
     3 $2 EXPRESSION ---------- 6
     3 OPT(#1) ---------- 7         ――8～12 が省略可能であることを示す
      4 "ト" ---------- 8
      4 $3 EXPRESSION ---------- 9
      4 ORDER(#2(1:1)) ---------- 10  ――11,12 のいずれかを選択することを示す
       5 "トノワ" ---------- 11
       5 "トノヲ" ---------- 12
  BYデータ
    1 BY ---------- 13
    2 COLUMN(7) ---------- 14       ――15 以降を 7 カラムから出力することを示す
    2 $1 ---------- 15              ――VARIABLE の翻訳結果を出力する
    2 "=" ---------- 16             ――文字「=」を出力する
    2 $2 ---------- 17
    2 OPT(#1) ---------- 18         ――構文解析の結果によって 19～22 を省略する
     3 ORDER(#2) ---------- 19      ――  〃  〃  20, 21 を選択する
      4 "+" ---------- 20
      4 "-" ---------- 21
     3 $3 ---------- 22
  MSGデータ
    1 MSG ---------- 23
    2 "KAGENﾌﾞﾝ ｶﾞ ｼｮｳｶｲﾚﾀ" ---- 24   ――28 が 0 回以上繰り返されることを示す
COMの定義
    0 COM ---------- 25
    1 SYNTAX ---------- 26
    2 ITER(#1(0:*)) ---------- 27   } COM は 3 で注釈指定として参照されている．この例では 4 と 5，5 と 6，6 と 8，8 と 9，9 と 11 あるいは 12 の間にある 0 個以上の空白を解析する
     3 " " ---------- 28
```

この例では新しい加減文　　X ﾉｱﾀｲﾊ Y ﾄ Z ﾄﾉﾜ

が入力されると目的プログラムとして　　X=Y+Z　が 7 カラムから

出力され，またメッセージとして　　KAGENﾌﾞﾝ ｶﾞ ｼｮｳｶｲﾚﾀ が印刷出力される

図 4 PROTO-E 言語による定義と翻訳の例
Fig. 4 Example of the definition in PROTO-E language and the translation using it.

```
0 DCL  ------------------------------ 1
 1 SYNTAX  --------------------------- 2
  2 CMT(COM)  ------------------------ 3
   3 "DECLARE"  ---------------------- 4
   3 $1 VARIABLE  -------------------- 5
   3 $2 ATTRIBUTE  ------------------- 6
   3 ";"  ---------------------------- 7

0 ASSIGN  --------------------------- 8
 1 SYNTAX  --------------------------- 9
  2 CMT(COM)  ------------------------10
   3 $1 VARIABLE  --------------------11
   3 "="  ----------------------------12
   3 $2 EXPRESSION  ------------------13

 1 MSG  ----------------------------14
  2 OPT(^@PRESENT
     (@SEARCH(SX(DCL.$1),KEY($1)))) --15
  3 "センゲン サレナイ ヘンスウ カ゛ ショウ サレタ"  ---16
```

15は，代入文 (ASSIGN) の左辺に指定された変数が宣言 (DCL) されたものか否かをチエックしている．

(^@PRESENT: 否定 / 要素が存在する場合 1 / 〃 存在しない場合 0 を返す関数)

図 5　構文間の情報授受の例

Fig. 5　Example of passing information between definition units.

（2）　変数の参照

PROTO-E 言語には，構文を適用した結果の状態を保持するための変数がある．変数には(1)で示した超結合子変数と BNF や ANF における超変数がある．BNF で左辺に記述される超変数は，PROTO-E において構文定義名として記述される．また右辺の超変数は「$n 構文定義名」の形で記述され，参照は $n で表現される．これらの変数をBYデータ中で参照することにより，構文定義から意味定義へ情報を引き渡すことができる．また左文脈の値が確定した変数については SYNTAX データ中からも参照できる．

（3）　物理的位置の表現

位置指定：COLUMN（i）

COBOL 等の固定形式記述の言語について，プログラム文字列の物理的位置を表現する機能である．i は，次に処理する文字のカラム位置を示す式である．

（4）　注釈等の表現

注釈指定：CMT（構文定義名）

注釈，空白，継続行指定など論理的に意味のないプログラム部分を効率的に記述する機能である．この指定がある構造体中の各要素間に，構文定義名で指定された構文定義あるいは意味定義が含まれていることを示す．構文解析では，指定された構文を適用するが，解析した文字列は木構造の情報としては保存されない．意味づけでは，指定された構文定義名の意味定義部分を各要素間に出力する．

（5）　構文間の情報の授受[11]

文脈依存部分の定義のために導入した機能であり，構文解析で作成される木構造上において，各構文間で情報の授受ができる．

①　木構造探索関数：@ SEARCH (SX（構文定義名．パラメータ），KEY（構文定義名．パラメータ））

構文間にわたる変数の参照であり，SX で指定した構文中のパラメータの中から KEY で指定した構文中のパラメータに等しいものを探し，それを部分木として返す(図5参照)．このほかに木構造上で親，祖先，子孫等を探索する関数がある．

②　部分木設定指定：SET（構文定義名．$n）

この指定のある構造体を部分木に編集して，指定された作業用の超変数 $n に設定する．

（6）　演 算 処 理

超定数，超変数，超結合子変数，数値および各種の関数をオペランドとする四則演算，比較，論理演算ができる．関数には(5)で説明したもののほかに，構文の存在の有無，文字列，数値等を返すものがある．

（7）　構文解析の制御

文脈に依存した構文解析を行うための機能である．

①　条件判定指定：COND（論理式）

式が成立すれば解析を進め，不成立のときはその処理を中断し，後戻りしてほかの代替処理を試みる．

②　先読み指定：TEST

入力文字列を先読みして，この指定のある構造体の要素の適用を試みる．適用できれば処理を進め，適用できないときは処理を中断し，後戻りしてほかの代替処理を試みる．

（8）　オウン・コーディング機能

PROTO-E 言語で定義できない部分あるいは処理効率上問題がある部分について，利用者がオウン・コーディングできる機能である．オウン・コーディングは関数として表現され，構文定義中のオウン関数の値は原始プログラムの文字列として扱われ，意味定義中のものは目的プログラムの文字列として扱われる．通常，構文定義中のオウン関数のルーチンではパラメータとして渡された超変数等から各種の情報を登録し，空の値を返す．意味定義中のオウン関数のルーチンでは，それらの情報を参照して目的プログラムの文字列を値として返す．

3.3　構文解析法

PROTO-E の構文解析法は，制限付き下向き直構文解析法[12]をベースにしたもので，次の特徴を持つ．

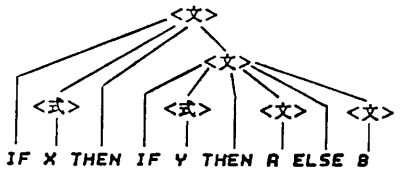
(a) 構文定義:

```
0 STATEMENT  ------------- 1
 1 SYNTAX  ---------------- 2
  2 CMT(COM)  ------------- 3
   3 "IF"  ----------------- 4
   3 $1 EXPRESSION  -------- 5
   3 "THEN"  --------------- 6
   3 $2 STATEMENT  --------- 7
   3 OPT(#1)  -------------- 8
    4 "ELSE"  -------------- 9
    4 $3 STATEMENT  -------10
```

(b) 入力文字列:

```
IF X THEN IF Y THEN A ELSE B
```

(c) 解釈 1:



(d) 解釈 2 (最深最長の解釈)

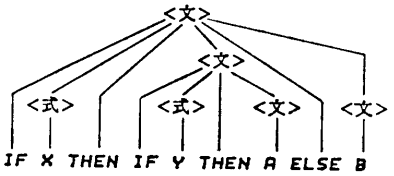


図 6 構文解析における曖昧性の除去

Fig. 6 Elimination of ambiguity at the syntax analysis.

(1) 解析の途中の試行錯誤 (後戻り) は構文定義の集合および超結合子変数の構造体に対しては行わない.

例: 3 ORDER (#1(1:1))
4 $1 VARIABLE
4 $2 EXPRESSION

$1の適用が成功すると, その後, 後戻りによって$2の適用を試みることはない.

(2) OPT ではその要素が省略できない場合を優先し, ITER では許されるだけ繰り返して要素の適用を試みる. したがって図6に示すように構文解析上2通りの解釈ができるような場合は, 最深かつ最長のものが優先して適用され, (d)が一意の解釈となる.

(3) 各構造体に対応するルーチンの再帰的呼出しで構文解析を行うので(図7参照), 新しい機能追加は新しい構造体形式を言語仕様に追加し, 新しいルーチンを再帰的ルーチン群に追加するだけで良い[13].

(4) 後戻り量を減らすために, 構文の適用時にあらかじめ構文適用の可能性を判断する選択表[14]を導入した. 選択表の作成で問題となる処理量の増大については, 高速なアルゴリズムを導入し解決した[15].

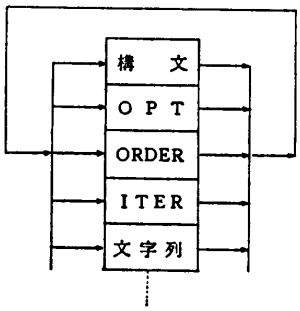


図 7 構文解析を行う再帰的ルーチン群

Fig. 7 Set of routines calling each other recursively for syntax analysis.

(5) 入力文字列は, できる限りまとめて定義データの文字列と照合する. このために, ハードウェアの可変長データ論理演算命令やマクロ命令に対応した定義機能を導入した.

4. 評価と分析

PROTO-E の実用性を確認するために記述能力, 生産性および性能について評価した. 評価は実用の COBOL コンバータ* を PROTO-E で記述することにより行った. 以下, 評価結果を示し分析する.

4.1 記述能力

変換機能の中核部分を PROTO-E 言語で, また PROTO-E では記述できない部分についてはシステム記述用言語で記述することにより, 変換率が 99.9% であるコンバータを開発できた.

PROTO-E では記述できず, システム記述用言語で記述した項目とその記述の割合を表3に示す. このうち前処理・後処理ルーチンは, プロセッサの操作回りの機能であり, 本質的に PROTO-E 言語による定義の対象とならないものである. したがって, この部分を除いて考えれば, PROTO-E がねらいとしている定義領域の 14% が性能あるいは記述性等の問題から定義不可能であったといえる. また本コンバータでは実現しなかったが, デバッグ行の変換をオプションとする等の外部パラメータの指定による翻訳処理の切り分けは PROTO-E では対処できない. これを実現するためには, 複数の翻訳規則の表を作成し, それを選択して翻訳させる前処理を作成すれば良い.

4.2 生産性

本コンバータは PROTO-E 言語 3.6k ステップと

* ANSI' 68 COBOL から ANSI' 74 COBOL への変換プロセッサであり, システム記述用言語の記述で約 15k ステップの規模のものである.

表 3 定義不可能項目

Table 3 Data which can't be defined in PROTO-E language.

項目	内容	対処	記述量*
散在化した多くの情報を必要とする複雑な変換	COBOL の乱呼出し機能から相対入出力機能への変換	オウン関数ルーチン	4%
実行頻度が高く，処理効率の向上を必要とするもの	予約語のチェック	オウン関数ルーチン	8%
コンパータ特有の機能を実現する必要があるもの	変換プログラム・リストの作成，COPY 句の展開，コンパータのスタック処理制御	前処理**後処理ルーチン	12%

* 全仕様の記述（15 k ステップ）に対する割合

** 入出力処理，リストの作成等について利用者固有の処理を行う場合，それを処理するプログラムを作成して PROTO-E と連結する

システム記述用言語 3.5 k ステップで記述されており，同一の機能を持つものをシステム記述用言語 15 k ステップで記述する従来の手法による場合に比較して，約 2 倍の生産性で開発できた．工数削減の要因を**表 4** に示すが，記述量の削減と非手続的記述によるところが大きい．

4.3 性　能

COBOL コンパータを用いた PROTO-E 翻訳部の性能評価結果を示す．

（1） 処 理 速 度

PROTO-E 翻訳部において構文解析，意味づけおよび入出力編集が占める処理時間の割合は，それぞれ 50, 15, 35% である．構文解析においては，A 個の構文の適用によって解析される原始プログラムの入力に対し 1.4A 個の構文の適用を試みており，後戻りの割合は約 3 割である．また翻訳部全体の処理速度は，コンパイラ* の約 1/5 であり，実用に十分な性能である．**表 5** に処理速度の改善に効果があった要因と改善率を示す．

（2） 翻訳可能入力限界

翻訳可能な原始プログラムの最大行数は約 7,500 行であり，通常のプログラムの変換に支障はない．なお，プロセッサのメモリ管理方式を拡張することにより，この制限は緩和できる．

5. む す び

高級言語の拡張，構文と意味の形式的定義，汎用の翻訳機能および実用的な構文解析法を採用した拡張型言語システム PROTO-E を開発した．具体的な利用法としてはコンパータを作成，評価することにより，本言語拡張方式が実用上，十分に有効であることを確認した．

* ANSI' 68 COBOL のコンパイラ

表 4 生産性向上の要因

Table 4 Factors which affect the improvement in productivity.

要因	説明
記述量の削減	記述量は既存のシステム記述用高級言語による場合の 1/4〜1/8 である．*1)
非手続的記述	変換仕様の機能のみの把握でよい．
モジュール化	構文ごとの分割記述，構造体風記述により，定義対象となる言語を階層的に把握できるので，記述性，保守性にすぐれる．
変数の値の固定化	変数はただ 1 カ所でのみ値の設定が行われ，その後変更されることがないので変数処理の誤りが少ない．

*1) 文献 16) 参照

表 5 処理速度改善の要因と改善率

Table 5 Factors which improved the processing speed and their contributions.

改善要因	改善率（倍）(*)
選択表の導入	2.1
文字列照合の高速化	2.7
最適化機能の導入	1.3
全体	7.3

(*) 改善要因を採用しないときを基準値（1）とする．

一般に，超言語における記述能力と記法の簡明性は相反する要求であるが，PROTO-E では，意味の形式的定義のために ANF を上回る記述能力を保持しながら，構造体風の定義形式を導入することにより簡明性の低下を補っている．これにより，従来の拡張型言語の実用性の低さを改善し，記述能力と簡明性に優れた超言語を実現したことが PROTO-E の特徴といえる．

PROTO-E は言語の拡張，コンパータの作成のみならず，文字列のパターン認識や各種データの生成等に適用でき，現在，各種ツールの作成に使用されている．今後は本研究の成果をふまえて，新しい拡張方式と適用分野の開拓を目指す．

謝辞 本研究の遂行にあたりご指導を賜った東京芝浦電気(株)筑後道夫氏，武蔵野電気通信研究所寺島情報通信サービス研究室長に感謝する．なお本研究は DIPS プロジェクトの一環として実施したものであり，共同研究にあたられた日本電気(株)の関係各位に感謝する．

参 考 文 献

1) Brown, P. J.: The ML/I Macro Processor, Comm. ACM, Vol. 10, No. 10, pp. 618-623 (1967).

2) Wijngaarden, A. van et al.: Report on the Algorithmic Language ALGOL 68, MR 101, Mathematishe Centrum, Amsterdum (1969).
3) Mailloux, B. J. and Peck, J. E. L.: ALGOL 68 as an Extensible Language, SIGPLAN Notices, Vol. 4, No. 8, pp. 9-13 (1969).
4) Arden, B. W., Galler, B. A. and Graham, R. M.: The MAD Definition Facility, Comm. ACM, Vol. 12, No. 8, pp. 432-439 (1969).
5) Standish, T. A.: Some Features of PPL-A Polymorphic Programming Language, SIGPLAN Notices, Vol. 4, No. 8, pp. 20-26 (1969).
6) Standish, T. A.: Some Compiler-Compiler Techniques for Use in Extensible Languages, SIGPLAN Notices, Vol. 4, No. 8, pp. 55-62 (1969).
7) 島内剛一：プログラム言語論——ALGOL 60 から ALGOL N へ——，共立出版 (1972).
8) 雨宮真人，竹内郁夫：プログラミング言語 FORMAL-2 とその処理系，通研実報，Vol. 22, No. 8, pp. 2195-2214 (1973).
9) Solntseff, N. and Yezerski, A.: A Survey of Extensible Programming Languages, Computer Science & Technology & their Application, Pergamon Press, pp. 267-307 (1974) (Annual Review in Automatic Programming 7).
10) Brown, P. J. 著，鳥居，杉藤，真野訳：マクロ・プロセッサとソフトウェアの移植性，近代科学社 (1977).
11) 古山恒夫，稲田 満，徳田憲夫：言語記述システムにおける構文間情報連絡の一方法，昭和53年度電通学会全国大会予稿集.
12) Aho, A. V. and Ullman, J. D.: The Theory of Parsing, Translation and Compiling, Vol. 1: Parsing, Prentice-Hall (1972).
13) 古山恒夫，稲田 満，徳田憲夫：汎用性を持つ言語記述システムにおける構文解析の一方法，第18回情処学会全国大会論文集 (1977).
14) 井上謙蔵：コンパイラ・コンパイラ，p. 260，産業図書 (1970).
15) 古山恒夫，稲田 満，馬場康彦：下向き直構文解析に用いる選択表作成の効率的アルゴリズムについて，第19回情処学会全国大会論文集 (1978).
16) 古山恒夫，稲田 満，馬場康彦，徳田憲夫：拡張型言語 PROTO-E の設計と評価，信学会研資 EC 78-5 (1978).

(昭和54年8月29日受付)
(昭和55年11月20日採録)