# 言語処理系の生成系MYLANGの算法とデータ構造を考慮した処理速度の向上

（昭和58年11月30日　原稿受付）

情報工学教室　石川　洋
同　山之上　卓
同　安在弘幸

# Speeding Up the Language Processor Generator MYLANG through Experiments with Algorithms and Data Structures

by Hiroshi ISHIKAWA
Takashi YAMANOUE
Hiroyuki ANZAI

**Abstract**

Speeding up execution of our language processor generator MYLANG through experiments with testing various algorithms and data structures used for the automaton generation part in MYLANG are discussed.

In order to represent an automaton concretely in computers, we adopted the following three types of data structures: (1) the linked structure faithfully representing the state diagram of the automaton, (2) a three dimensional bit array, (3) the linked structure of a matrix of which each element is a bit vector. Then we made algorithms associated with the three types of data structures respectively, measuring execution times of them, and obtaind the following result. (i) When the number of states of the automaton is large, the type (3) is best, however, (ii) when the one is small, the type (1) is better than the type (3). On the basis of the result, we improved our MYLANG system. The execution time of the new system was 34% of the one of the old system.

## 1. まえがき

我々の開発した言語処理系の自動生成システムMYLANG[2),3),4)]は，現在 UCSD PASCAL システム，九工大情報処理センター，UNIX システム上で稼動している。これらの MYLANG は，生成する言語処理系を表現する中間モデルとしてオートマトンを採用しており，このオートマトンを処理することで言語処理系の自動生成を行なう。ところが現在の MYLANG には，オートマトンの状態数の増加に伴い処理時間が急速に増加するという問題がある。

この原因は，MYLANG 内部におけるオートマトンの表現方法にあると考えられる。MYLANG は半線形代数[1)]を応用してオートマトンの処理を行なう。この代数では，オートマトンは行列により表現される。MYLANG ではリンク構造を用いてこの行列を表している。[3)] このリンク構造による表現は，オートマトンを合成する演算には有利である。しかし，簡約決定性変換で必要となる演算がベクトル同士の比較およびベクトルと行列の乗算を基本とするため，リンク表現では一般にこの部分の処理が煩雑になる。

さらに簡約決定性変換において使用されるベクトルや行列は，ブール代数におけるものと等価である。ゆえにこれらをビット列によって表現すれば，演算をより自然に表現することができるとともに処理速度の向上が計れることが予想される。これに基づき，オートマトンの処理方法を以下の様に変更した。

(1) オートマトンの合成はこれまで通りリンク構造表現を用いて行なう。

(2) 簡約決定性変換時には，オートマトンの表現をリ

ンク構造表現からビット表現に変換し，処理を行なう。

(3) 簡約決定性変換後は，ビット表現からリンク表現に再変換し以後の処理を行なう。

なおビット表現では行列を表すのにビット列の配列を用いる。

次にこれと，先のリンク構造による簡約決定性変換を用いた場合との実行時間の比較を行なった。その結果，ある数以上の状態数のオートマトンでは大幅に速度が向上しているが，それ以下の状態数のオートマトンではかえって速度が低下していることが明らかになった。このことをふまえてプログラムに検討を加えたところ，上記の原因は，配列要素に対するアクセスがリンク情報によるアクセスに比べ，頻常に遅いということにあった。

そこでビット表現における行列を，ビット列をリンクで接続する形で表すこととした。これによりMYLANGの実行時間を従来より大幅に短縮することに成功した。短縮の割合は処理対象にもよるが，今回サンプルとしたMYLANG自体の生成に関しては，全体の時間が従来の約34%に，簡約決定性変換部に限れば従来の約10%になった。

本報告では，リンク構造表現，配列によるビット表現およびリンク構造によるビット表現のオートマトンについてのデータ構造，並びにそれらに関する演算のアルゴリズムについて説明する。またそれぞれのアルゴリズムを実現したプログラムの実行時間について比較した結果を示す。

## 2. リンク構造表現によるオートマトンジェネレータ

MYLANGにおいてオートマトンの合成・簡約決定性変換を行なう部分を，我々はオートマトンジェネレータと呼んでいる。この部分はMYLANGの最も主要な部分である。本章では，リンク構造表現を用いたオートマトンジェネレータで扱うデータ構造，簡約決定性変換で用いる演算の概略アルゴリズム及び簡約決定性変換部の実行時間の測定結果について述べる。

### 2.1 データ構造

図-1はオートマトンを状態遷移図で表現した例である。これを半線形代数における $\Sigma$-右線形方程式

$$x = Ax + c \tag{1}$$

により表現すると

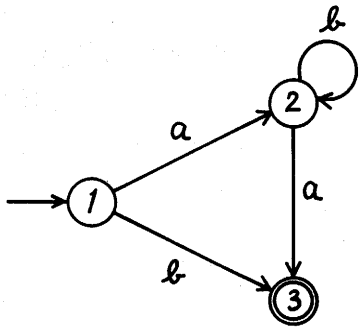



図-1 オートマンの状態遷移図

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \phi & a & b \\ \phi & b & a \\ \phi & \phi & \phi \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \phi \\ \phi \\ \lambda \end{bmatrix}, \quad \Sigma = \{a, b\} \tag{2}$$

となる。ここで $\Sigma$ はオートマトンにおける遷移（入力）記号の集合を表す。また $\lambda$ は無条件の遷移を，$\phi$ は遷移が存在しないことを表す。$c$ における $\lambda$ は，その行が表す状態が終状態であることを表す。

$\lambda, \phi$ は半線形代数の演算・，＋に対して以下の性質を持つ。

$$\begin{cases} a\lambda = \lambda a = a & (3) \\ a\phi = \phi a = \phi & (4) \end{cases}$$

ただし $a \in \Sigma$

$$\begin{cases} \lambda \cdot \phi = \phi \cdot \lambda = \phi & (5) \\ \lambda \cdot \lambda = \lambda & (6) \\ \phi \cdot \phi = \phi & (7) \end{cases}$$

$$\begin{cases} \lambda + \lambda = \lambda + \phi = \phi + \lambda = \lambda & (8) \\ \phi + \phi = \phi & (9) \end{cases}$$

リンク構造表現によるオートマトンジェネレータにおいて，行列およびベクトルの要素は図-2の様に表ろれ

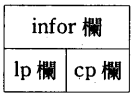



図-2 ベクトルと行列の要素

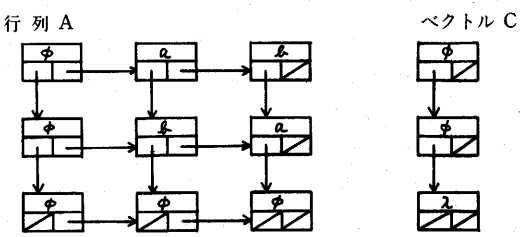



図-3 リンク表現によるオートマトン

る。ここで *lp* 欄，*cp* 欄はそれぞれ行方向，列方向へのリンク情報を持ち，infor 欄は遷移記号の集合を持つ。これにより式-(2)に示した行列 *A*, ベクトル *c* は，図-3の様に表現される。

### 2.2 アルゴリズム

簡約決定性変換における基本的演算は次の2つである。

1) 2つのベクトルが等しいか否かの比較。

2) ベクトル *v* と行列 $\partial_\sigma A$ の乗算。

ここで $\partial_\sigma A$ は，半線形代数における微分を表す。*A* の $(i,j)$ 成分を $a_{ij}$, $\partial_\sigma A$ の $(i,j)$ 成分を $(\partial_\sigma A)_{ij}$ とすれば，これは以下の様に定義される。

$$(\partial_\sigma A)_{ij}=\begin{cases}\lambda & if \quad \sigma\in a_{ij}\\ \phi & if \quad \sigma\notin a_{ij}\end{cases}$$

式-2において $\partial_a A$ を求めれば，

$$\partial_a A=\begin{bmatrix}\phi & \lambda & \phi\\ \phi & \phi & \lambda\\ \phi & \phi & \phi\end{bmatrix}$$

となる。これをリンク構造表現により表せば，図-4の様になる。

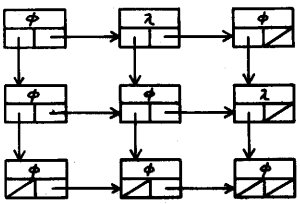


図-4 リンク表現による ∂aA

さて，リンク構造表現による上記2つの基本的演算1)，2) は以下に示すアルゴリズム A1, A2 で表現される。

アルゴリズム A1)

入力はベクトル *va*, *vb* とし，この両者を比較する。これらは図-5の様な形をしている。ただし next 欄は実際には図-2に示した *lp* 欄，*cp* 欄のいずれかにあたる。

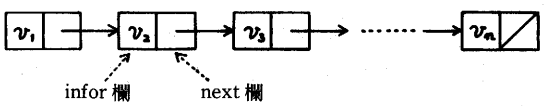



図-5 リンク表現によるベクトル

出力は論理型の変数 *b* とする。*va*, *vb* が等しければ *b*='真'，等しくなければ *b*='偽' となる。

またこれ以外に，*va*, *vb* の要素に対するポインタ *la*, *lb* を用いる。

アルゴリズムは次の通りである。

(1) *va*, *vb* の先頭の要素を *la*, *lb* で指す。*b* を '真' とする。

(2) *la* と *lb* の指す infor 欄の内容が等しくなければ，*b*='偽' とする。

(3) *la*, *lb* が指す要素を，現在 *la*, *lb* が指す要素の next 欄が指す要素とする。

(4) *la*, *lb* が空または *b* が '偽' ならば終了，そうでなければ(2)にもどる。

ここで，*va*, *vb* と *la*, *lb* の関係は図-6の様になる。

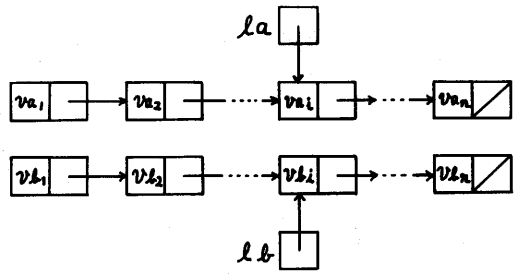


図-6 アルゴリズム A1 におけるベクトルとポインタの関係

アルゴリズム A2)

ベクトル *v*, 行列 *A* 及び遷移記号 $\sigma$ を入力とし，$\partial_\sigma A$ を計算する。結果はベクトル *x* に出力される。またその他に *v*, *x* の要素を指すポインタ *vp*, *xp*, 行列 *A* の要素を指すポインタ *lp*, *cp* を用いる。ただしポインタ *lp* は，現在注目している行を示す目的で使用される。アルゴリズムは以下の通りである。

(1) *v*, *x* の先頭の要素を *vp*, *xp* で指す。また *A* の $(i,j)$ 成分を *lp* で指す。

(2) *cp* で *lp* の指す要素を指す。

(3) *vp* の指す要素の infor 欄$=\lambda$ かつ *cp* の指す infor 欄$\ni\sigma$ ならば，*xp* の指す要素の infor 欄を $\lambda$ とする。

(4) *cp* が指す要素を，*cp* が現在指している要素の *cp* 欄が指す要素とする。*xp* が指す要素を，*xp* が現在指している要素の next 欄が指す要素とする。

(5) *cp*, *xp* が空でなければ(3)にもどる。

(6) *lp* が指す要素を，*lp* が現在指している要素の *lp* 欄が指す要素とする。*vp* が指す要素を，*vp* が現在指している要素の next 欄が指す要素とする。*xp* を

$x$の先頭にもどす。

(7) $lp$, $vp$ が空でなければ(2)にもどる。

各ベクトル，行列およびポインタの関係を図-7に示す。

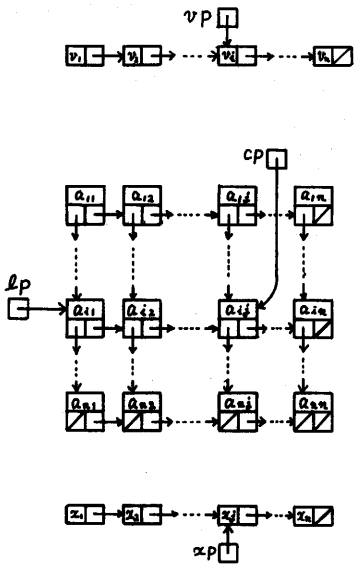



図-7 アルゴリズム A2 における行列・ベクトル・ポインタの関係

### 2.3 測定結果

MYLANG 全体の実行時間のうちでオートマトンジェネレータの簡約決定性変換部が占める割合は，かなり大きいと以前から予想されていた。しかしこの割合は実際どの程度のものかは不明であった。そこでまず，簡約決定性変換部と MYLANG 全体のそれぞれの実行時間を測定し，その比率を求めた。

なお，測定は九工大情報処理センターの計算機 MELCOM COSMO 800 III 上の MYLANG について行ない，入力には MYLANG 自体を定義した RTF (Ragular Translation Form) を用いた。測定方法は，測定の対象となる部分の先頭と末尾で計算機の内部タイマ（精度 1ms）を呼び出してその値を出力させ，両者の差をとって実行時間とした。以後の測定もすべて上の条件通りに行なった。

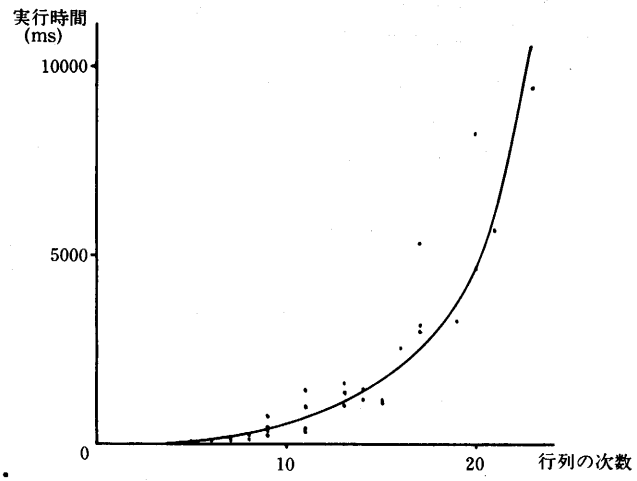
表-1が測定の結果である。簡約決定性変換部のプログラムは，その長さが全体の約10%であるが，その実行時間は全実行時間の3/4を消費している。

次に，簡約決定性変換に要する時間が，行列の大きさによってどれほどの影響をうけるかを調べた。その結果

表-1 MYLANG の実行時間にしめる簡約決定性変換の割合

| | 実行時間(ms) | 比 率(%) |
|---|---|---|
| 簡約決定性変換 | 91112 | 74.78 |
| 全 体 | 121832 | 100.00 |

グラフ-1 リンク表現による簡約決定性変換の実行時間




をグラフ-1に示す。

これは，各オートマトンを表す行列の次数の変化に対する実行時間の変化を示している。このグラフから，行列の次数（すなわちオートマトンの状態数）が大きくなると簡約決定性変換にかかる時間が急速に増加することがわかる。

## 3. 配列によるビット表現を使用したオートマトンジェネレータ

簡約決定性変換で扱うベクトルおよび行列の要素はすべて $\lambda$ か $\phi$ のどちらかをその値とする。これらは式-(5)～(9)の様な性質を持つ。よって簡約決定性変換で用いるベクトル，行列および演算は，ブール代数におけるベクトル，行列および演算とそれぞれ等価となる。このことから，ベクトルや行列をリンク構造で表すよりは，ビット列で表現した方が演算によくなじむはずである。またビット表現ではベクトルを1つのビット列で表すことができるために演算にかかる手間を減らすことができ，従って実行時間を短縮することができる。この観点から，新たにビット表現による簡約決定性変換部を作成した。本章では，前章と同様に，ビット列を用いた

データ構造，演算のアルゴリズムおよび実行時間の測定結果について述べる。

### 3.1 データ構造

ビット列を用いると，ベクトルは，1つのビット列で表現される。また $\partial_\sigma A$ の様な $\{\lambda, \phi\}$ 行列はビット列の一次元配列で表される。行列 $A$ は，すべての遷移記号 $\sigma_k$ についての $\partial\sigma_k A$ をつなげた形で表される。これは，ビット列の二次元配列となる。以上に基づき式-(2)の $A, c$ および $\partial_\sigma A$ を表すと，図-8の様になる。

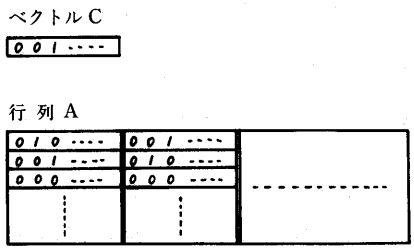



図-8　配列を用いたビット表現によるオートマトン

### 3.2 アルゴリズム

前節で示したデータ構造を採用すると，アルゴリズム A1) は，2つのビット列を比較する演算になる。また，アルゴリズム A2) は次の様になる。

アルゴリズム　A2)

入力を，ベクトル $v$, 行列 $A$ および行列の次数 $m$ とする。出力はベクトル $x$ となる。また $v$ のビットの位置を $i$ $(i=0\cdots m-1)$ で表す。アルゴリズムは以下の通りである。

(1) $i$ が0から $m-1$ まで(2)を繰返す。

(2) もし $v$ の第 $i$ ビットが1ならば，$x$ と $A[i]$ のビットごとの *or* を取り，$x$ に代入する。

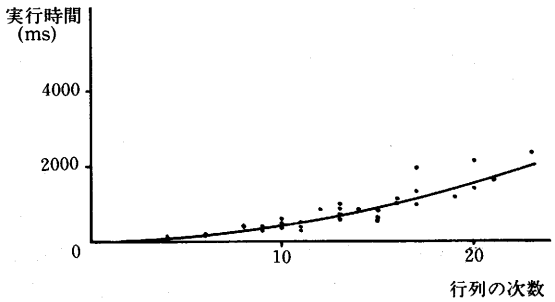
### 3.3 測定結果

配列によるビット表現を用いた簡約決定性変換についても，2.3で行なったと同様に行列の次数による実行時間の変化を測定した。その結果をグラフ-2に示す。

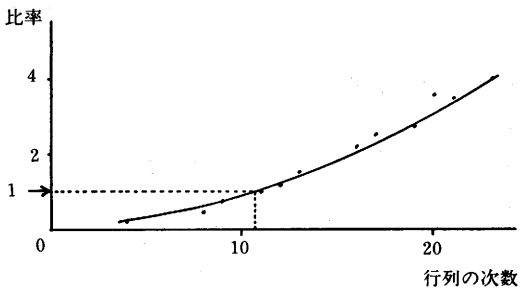
同図に示すように，次数に対する実行時間の増加の割合はリンク表現による簡約決定性変換の場合に比べて非常に低くおさえられている。しかしある程度の次数以下の行列についてはリンク表現を用いた場合の方が実行時間が短くなっている。ビット表現の実行時間に対するリンク表現の実行時間の比をグラフ-3に示す。同図には上述の関係が明確に示されている。

上の原因を見つけるべく種々の分析を行なったところ，この原因は配列要素のアクセスがリンク情報によるアク

グラフ-2　配列を用いたビット表現による簡約決定性変換の実行時間




グラフ-3　リンク表現／ビット表現の変化




```
program test(output);
  type slp=@sel;
       bit=set of 0..63;
       sel=record infor:bit; lp,cp:slp end;
       mat=array [1..100,1..100] of bit;
  var  x:slp; m:mat;
       i,j:integer;
       data:bit;
begin
  new(x); x@.infor:=[]; x@.cp:=x; x@.lp:=x;
  data:=[0];
  writeln(' ***** link start *****',clock);  { call interval timer }
  for i:=1 to 100 do begin
    for j:=1 to 100 do begin
      x@.infor:=data; x:=x@.cp
    end;
    x:=x@.lp
  end;
  writeln(' *****  link end  *****',clock);
  writeln(' ***** bit start  *****',clock);
  for i:=1 to 100 do for j:=1 to 100 do m[i,j]:=data;
  writeln(' *****  bit end   *****',clock)
end.
```

図-9　ベンチマーク用プログラム

表-2 ベンチマークの結果

| 要素のアクセス法 | 実行時間(ms) |
|---|---|
| リンクによるアクセス | 250 |
| 配列によるアクセス | 594 |

セスに比べて大幅に時間がかかることにあることが明らかになった。図-9にこの結果を得るために使用したベンチマークテスト用プログラムを，またその測定結果を表-2に示す。この理由としては，配列の場合，添字からアドレスを計算する必要があるのにくらべ，リンク構造の場合直接アドレスが書かれていることが考えられる。

## 4. リンク構造によるビット表現を用いたオートマトンジェネレータ

前章で，配列要素のアクセスに思いのほか時間がかかることが明らかになった。従って前章で配列を用いた部分をリンク構造で記述すればよりいっそうの高速化を実現できると考えられる。そこでこれに基づきデータ構造，演算のアルゴリズムを改良し，実行時間の測定を行なった。本章では，これらについて述べる。

### 4.1 データ構造

1個のビット列は図-10の様に表される。ここで，

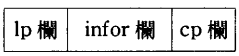



図-10 リンク構造を用いたビット表現におけるベクトルと行列の要素

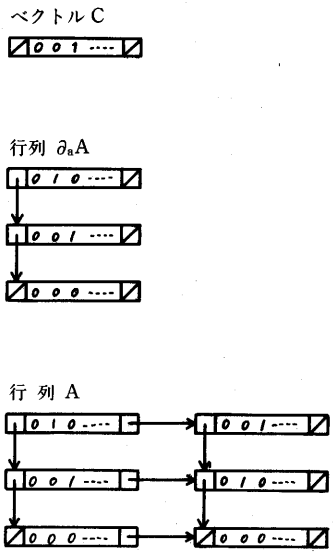



図-11 リンク構造を用いたビット表現におけるオートマトン

infor 欄はビット列，*lp* 欄は行方向へのリンク，*cp* 欄は $\partial_a$ についてのリンクを表す。これで式-(2)の $A$, $\partial_a A$, $c$ を表すと図-11の様になる。

### 4.2 アルゴリズム

ベクトルの比較は，前章と同様にビット列の比較で表される。またアルゴリズム B2) は以下の様になる。

アルゴリズム C2)

入力を，ベクトル $v$, $\{\lambda, \phi\}$ 行列 $A$ とし，出力をベクトル $x$ とする。また $A$ の行に対するポインタ $P$ を用いる。アルゴリズムは次の通りである。

(1) $A$ の1行目を $P$ で指す。またベクトル $v$ のビットの位置を示す変数 $i$ を0とする。

(2) $v$ の第 $i$ ビットが1ならば，$x$ の infor 欄と $P$ の指す要素の infor 欄のビットごとの *or* を取ったものを $x$ の infor 欄に代入する。

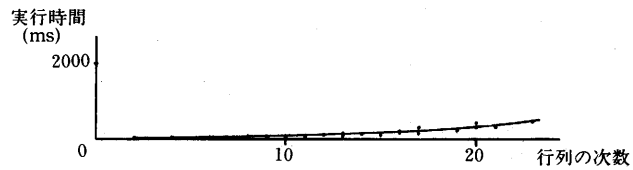
(3) $P$ が指す要素を，現在 $P$ が指す要素の *lp* 欄が指す要素とする。また $i$ に1を加える。

(4) $P$ が空なら終了。そうでなければ(2)にもどる。

### 4.3 測定結果

2章，3章と同様に，前節のアルゴリズムによる簡約決定性変換部について行列の次数に対する実行時間の測行を行なった。その結果をグラフ-4に示す。

グフラ-4 リンク構造を用いたビット表現による簡約決定性変換の実行時間




これをグラフ-1と比較すると，実行時間が大幅に短縮されているのがわかる。また表-1と同様に簡約決性性変換部と MYLANG 全体のそれぞれの実行時間とその比率を表-3に示す。さらに表-1の結果に対する表-3の結果の比を表-4に示す。

表-3 改良後の実行時間

| | 実行時間(ms) | 比 率(%) |
|---|---|---|
| 簡約決定性変換 | 8844 | 21.12 |
| 全 体 | 41872 | 100.00 |

表-4　改良前に対する改良後の実行時間の比率

| | 比　率(%) |
|---|---|
| 簡約決定性変換 | 9.71 |
| 全　　体 | 34.37 |

## 5. あとがき

本報告では，簡約決定性変換部におけるデータ構造と演算のアルゴリズムの改良により，MYLANG の高速化に成功したことについて述べた。MYLANG の使用方法や，半線形代数については参考文献を参照されたい。[1),2),3),4)]

現在，改良した MYLANG システムは九工大情報処理センターで稼動している。他のシステムについても同様の改良を行なう予定である。

## 参考文献

1）安在他：“半線形代数”，第1報～第8報，電子通信学会報告 AL 80-60, 61, 62, AL81-4, 5, 38, 39, 74 (1981)
2）山之上，安在：“属性付正規翻訳記法と属性付構文向き翻訳”，九州工業大学研究報告 (1983)
3）竹中豊弘：“言語処理系の自動生成に関する研究－MYLANG システムの開発”，九工大卒業論文 (1983)
4）鷺田慎一：“言語処理系の自動生成に関する研究－mini-BASIC 実現を例として”，九工大卒業論文 (1983)