自然言語処理　59－4
（1987. 1. 23）

CYK法構文解析の一検討：quick parsing について

鈴木克志，太細孝
三菱電機（株）情報電子研究所

Chomsky 標準形の文脈自由文法に対する構文解析アルゴリズム CYK（Cocke-Younger-Kasami）アルゴリズムを改良し，新アルゴリズム（quick parsing）を得た。本アルゴリズムは，分割統治の概念に従い，入力文を分割後，それぞれの部分を構文解析し，解を合成する。計算量のオーダは CYK 法と同等であるが，最初の解を早期に検出できる可能性を有する。

本文では，新アルゴリズムの概要，文法の性質（到達可能性）利用による解の早期検出法，分割戦略，及び簡単な日本語文法を用いての構文解析実験結果を述べる。

ON THE CYK PARSING ALGORITHM : INTRODUCTION TO QUICK PARSING

Katsushi SUZUKI , Takashi DASAI

Information Systems & Electronics Development Lab., Mitsubishi Electric Corp.
5-1-1 Ohfuna , Kamakura, Kanagawa, 247 Japan

We propose the new parsing algorithm, called 'quick parsing', for context free grammar restricted to Chomsky normal form. This algorithm is obtained by refining CYK(Cocke-Younger-Kasami) parser. It uses a dynamic programming technique similar to CYK, but is based on divide-and-conquer concept, which makes it possible to detect the first solution quickly.

In this paper, we describe (1)how to utilize transitivity in grammar rules, (2)the strategy to divide an input sentence, and (3)results of the experiments to evaluate the applicability to a simple Japanese grammar.

1. はじめに

本稿では，Chomsky 標準形の文脈自由文法に対する構文解析アルゴリズムとして著名な Cocke－Younger－Kasami（CYK法）アルゴリズム［Younger, 1967］の，一改良版について報告する。この新アルゴリズムは，分割統治の概念にもとづくので，quick sort になぞらえて quick parsing と名づけられている。そして，CYK法よりも早期に解の検出できる可能性を持ち，さらに，並列処理への応用可能性に富んでいるという特徴を持っている。以下では，quick parsing アルゴリズムの概要，文法の性質利用による解の早期検出法，及び自然言語処理への応用について考察する。

2. 背景

最近の自然言語処理の研究においては，GPSG［Gazdar, 1985］をはじめとして，文脈自由文法をベースとした文法理論の構築が盛んであり，文脈自由文法による構文解析技術もその重要度を増してきているものと考えられる。

従来，文脈自由文法の構文解析アルゴリズムは，LR(k) などのサブ・クラスを除いて汎用のものに話を限定したとしても，非常に良く研究されてきた。

まず，古くは単純な depth-first／back-track 法に始まる（［Griffiths, 1965］など）。これは，入力文の長さ n に対して，指数オーダの計算時間を必要とした。続いて，CYK法は，動的計画法により，Chomsky 標準形のルールに対して時間計算量 $O(n^3)$ を実現した。この動的計画法は，［Earley, 1970］，［Pratt, 1975］，［Graham, 1980］に継承され，計算量に多少の違いはあるが，いずれも $O(n^3)$ 程度で一般の文脈自由ルールに対してパージングが行なえる。

少し系統の異なるアルゴリズムとして，［Valiant, 1975］がある。これは，Chomsky 標準形に対するもので，構文解析操作を行列乗算に還元して $O(n^3)$ 以下の時間計算量を実現している。しかし，定数ファクタの増大のため，n が充分大きくないと実用的ではない。

このような状況において，我々はこれまで動的計画法にもとづく文脈自由文法の構文解析に関心を持ち，パーザの試作などを行なってきた。たとえば，属性集合付の木を表に保持しながら，Earley／Pratt のアルゴリズムでパージングを行なうパーザを開発済である。しかし，その過程で感じたことは，CYK，Earley，Pratt，及び Graham のパーザに共通する left_to_right パージング（入力文を左から右に one pass で処理する）の不自由さである。心理的実在性の観点からこの方式は自然であるとされているが，工学的処理の観点からは，left_to_right に固執する必要はない。現に，実在する多くのパーザは，入力文に対する前処理として，単文分割や並列句処理などを構文解析前に実行している。

そこで，今回はもっとも基本的な文脈自由パーザである CYK法を取上げ，left_to_right によらないアルゴリズムの検討を試みた。改良により高速化できれば，GPSG など属性を扱う最近の文法理論も，属性記述に相当する部分を非終端記号にあらかじめ展開しておく手法で文脈自由パーザにより解析できるから，適用が期待できるようになる。*

---

* 展開によるルール数，及び非終端記号数の増加，という問題はあるが。

3. CYK法とquick parsing

以下では，CYK法を紹介したあと，quick parsing の概要を紹介する。

3－1．記法

以下では，文法として chomsky 標準形を考える。通常の記法に従い，文法を $G=(P, S, V_t, V_n)$ とする。Pは生成規則の集合，Sは初期記号，V_t, V_n は終端記号の集合と非終端記号の集合である。以後，ストリングとして，α, β, γ, ... ，終端記号として a, b, c, ... ，非終端記号として A, B, C, ... を用いる。ただしストリングとは，$(V_T \cup V_N)^*$ の要素で，$*$ は集合の閉包である。初期記号 S は V_n の要素であり，生成規則は $A \to B$, C または $A \to a$ の形に制限されている。また，直接的導出 $\alpha \to \beta$ とは，ストリング α の一部に生成規則を適用して β に書換えることであり，その繰返しにより導出 $\alpha \overset{*}{\to} \beta$ を定義する。$\alpha \overset{*}{\to} \alpha$ とする。与えられたストリング α に対して，$S \overset{*}{\to} \alpha$ かどうか判定することを，α の認識と呼ぶ。以下では，構文解析という言葉で，この認識処理を表わすことにする。

3－2．CYK法

構文解析の対象として与えられたストリング（入力文）を，$w_{1,n} = w_1, w_2, \cdots w_n$ とおく。これは長さnの入力文である。$w_{1,n}$ の部分ストリングを，$w_{i,j} = w_i, w_{i+1}, \cdots w_j$ と書く。

CYK法では認識行列 t_{ij} ($i, j = 1, 2, \cdots, n$) を作業表として，動的計画法により構文解析を行なう。構文解析終了後の t_{ij} の値は，$w_{i,j}$ を導出することのできるような非終端記号の集合となる。すなわち，$A \in t_{ij}$ ならば $A \overset{*}{\to} w_{i,j}$ であり，その逆も成立する。

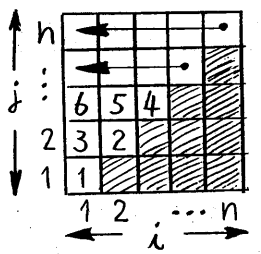

図1．認識行列 t_{ij}

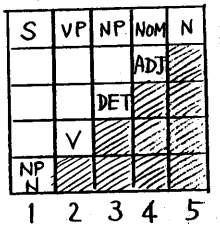

図2．解析例

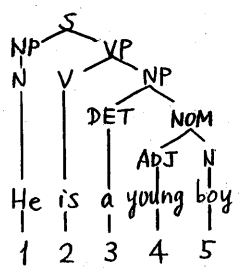

図3．解析木

CYK法では，この認識行列を，入力文を左から右に読込みながら，t_{ik} と $t_{k+1\,j}$ の値から t_{ij} を求めることによって，ボトムアップに作成していく。行列の各要素を埋める順番は，図1のようになる。最終的に t_{1n} に初期記号Sが求まれば，構文解析の成功である。図2に，図3の解析木に対応する認識行列を示す。

3－3 Quick Parsing

CYK法の特徴は，すべての解析結果をパラレルに求めていって，最後に入力文全体に対する解をいちどきに得ることである。これは，Earley, Pratt, 及び Graham などの方法にも共通する特徴であり，バックトラック制御のアルゴリズム（Prolog ベースの DCG や BUP など。）と異なる点である。

しかし，現実のシステムでは，ともかくも解をひとつだけでよいから，はやく

求めたいことが多い。そこで，ひとつの解の早期検出をめざして，図2の認識行列を見直してみると，$t_{15} \ni S$ という値は t_{11} と t_{14} から求まっているから，t_{12}，t_{13}，及び t_{14} の値は実は不要であったことがわかる。すなわち，図4のように，$t_{1n} \ni S$ の値が t_{1i} と $t_{i+1\,n}$ から求まるときには，斜線で示した四角形の部分は計算が不要である。このことから，次の分割統治法的アルゴリズムAが自然に思付く。

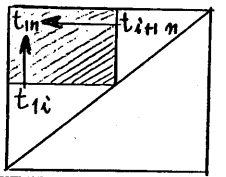

図4．計算不要部分

<アルゴリズムA>

(A1) 入力文を適当な位置iで $w_{1,i}$ の部分と $w_{i+1,n}$ の部分とに分割する。
(A2) それぞれの部分文字列 $w_{1,i}$ と $w_{i+1,n}$ をCYK法で構文解析する。
(A3) t_{1i} と $t_{i+1\,n}$ から t_{1n} にSが合成できれば終了。駄目ならば(A4)へ。
(A4) 他の分割点をひとつ適当に選び，(A2)以降の処理を繰返す。全ての分割点で試みたならば終了。

アルゴリズムAには，分割点としてどこを選ぶかという問題があるが，その決定法はまた後で述べることにする。それよりも，このアルゴリムには明らかな無駄がある。というのは，2度目以降の分割では，新しく選んだ分割点がそれまでの分割点のうちもっとも外側に位置する場合のみ，新たに認識行列の要素を埋める計算が必要なのである。すなわち，たとえば図5で番号順に分割点を選んだとしたとき，分割点1では斜線の部分をCYK法で計算し，分割点2では○印を2つ付けた部分の長方形を，分割点3では☆印をつけた部分の長方形をそれぞれ新たに計算すればよい。この長方形の部分の計算は，CYK法と同様に，左から右へ，そして下から上へと要素を埋めて行けばよく，単純なアルゴリズムなので詳細は省略する。

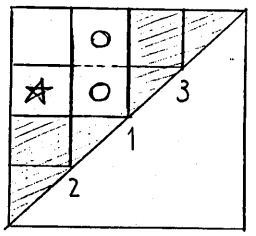

図5．再分割時の計算

4．文法の性質による高速化

アルゴリズムAは，(A3)でSが合成されても終了せずに他の分割点をすべて試みるようにすれば，CYK法と同等の計算能力を持つようになる。つまり，時間計算量のオーダが $\Theta(n^3)$ ですべての解を求めることができる。しかも，はじめの解を早期に検出できる場合がある点が魅力的である。しかし，以下では，さらなる高速化の工夫として，LRパーザでよく行われるような文法ルールの到達可能性（transitivity）の利用を考えよう。

4－1．分割点の候補検出

まず，初期記号Sから枝分れを起すすべてのルール（$S \rightarrow A, B$ の形をしたもの）の各々について，Aからの右到達可能性，及びBからの左到達可能性を調べることにより，到達可能な終端記号の集合 Right(A) 及び Left(B) を得ることができる。ただし，

$$Right(A) = \{ a \mid A \Rightarrow^{*} \alpha a \}$$
$$Left(B) = \{ b \mid B \Rightarrow^{*} b\beta \}$$

$(A, B \in V_N,\ a, b \in V_T,\ \alpha, \beta$ はストリング$)$

である。$Right(A)$ の意味をわかりやすく図示すると，図6になる。$Right(A)$は図6のような構文木を作り得るすべての a を集めた集合である。$Left(B)$ も同様である。このとき，入力文 $w_1 n$ が与えられると，$w_i \in Right(A)$, $w_{i+1} \in Left(B)$ なる i を，ルール $S \to A, B$ に対して分割の可能性を持つ点として検出することができる（図7）。この i をルール $S \to A, B$ に対する分割候補点と呼び，その集合を $Dpoint(S \to A, B)$ と書く。すなわち，

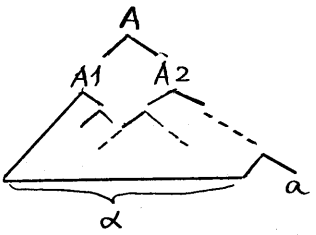

図6. 右到達可能性

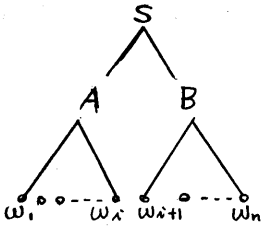

図7. 分割候補点

$Dpoint(S \to A, B) = \{ i \mid Right(A) \in w_i, Left(B) \in w_{i+1} \}$ である。また，$\langle w_i, w_{i+1} \rangle$ を分割可能ペアと呼ぶ。分割可能ペアは，ルールのみから定まる。

以上のことから，入力文中の各 w_i ($i = 1, 2, \cdots, n$) のうち，いかなる Dpoint 集合にも属さない点については，そこで分割しても構文解析が成功する可能性が全くないので，アルゴリズムAにおいて分割を何度も繰返していくとき，無駄な分割を防ぐことができることがわかる。

そこで，分割の戦略，すなわち，入力文中の分割候補点の中からどのような順番で分割候補点を選んでいくかを考えた場合，図8のように候補点の真中のものから右左，右左と，順に選んでいくやり方が浮かび上る。というのは，図5で述べたように，このようにすると分割点を選んだとき必ず四角形を埋める計算が入るからであり，単に終了チェックを行うためだけの分割が起きない点が好ましいからである。以下では，この戦略を仮定することにしよう。

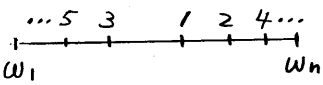

図8. 分割候補点を選ぶ順

4－2. 部分解の利用

さらに，再分割時に，それまでに求めた部分解を利用して，分割候補の妥当性と，解の早期検出をチェックできる。いま，図9のような状況を考えよう。すなわち，点 i で分割がすでに行なわれており，解が求まらなかったので，そのあと点 j で再び分割しようとしている状況である。

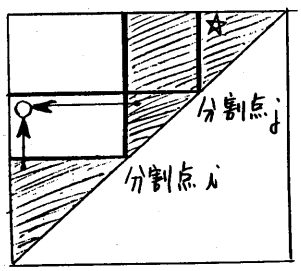

図9. 部分解の利用

このとき，すでに求まっている部分解 $t_{j+1,n}$（図9の☆印）の値と，点 j が属する Dpoint 集合とから，点 j が再分割に値する点かどうかのチェックが行なえる。詳しく言えば，点 j が属する Dpoint 集合を，$Dpoint(S \to P_1, Q_1)$, $Dpoint(S \to P_2, Q_2)$, …, $Dpoint(S \to P_k, Q_k)$ の k 個とすると，$\{Q_1, Q_2, \cdots, Q_k\}$ と集合 $t_{j+1,n}$ が共通の要素を持たなければ，点 j で再分割しても解が求まらない

ことがわかる。このチェックは，いわゆるPratt チェックと同等の意味を持っている。

さらに，もし{$Q_1, Q_2, \ldots, Q_k$}と集合 $t_{j+1,n}$ が共通の要素 Q_m を持つならば（共通の要素はひとつと限らないことに注意），P_m を t_{ij}（図の○印）に予測できるから，すでに求まっている部分解 t_{ik} と $t_{j+1,j}$ との値から，その予測値を合成することができる可能性もある。その場合，図9の○印を含む四角形の計算を省略できたことになる。

5．自然言語（日本語）への適用

quick parsing の有効性を考察するために，日本語の文法規則について適用を試みた。以下では，そのために行なった実験結果について述べる。なお，演算子優先順位文法への適用については，式の構文を例として，簡単な考察が[鈴木,1986]にあるので，本稿では省略する。

5－1．日本語文法の文脈自由文法による記述

対象とする日本語文法は，文法記述言語 Aglaia [鈴木, 1987] [大槻, 1987] で記述されている。この言語は，A→B,C の形式の文脈自由文法ルールに属性操作を付加した形式を有し，Shiebar の PATR-II や Kay の Unification Gramar と同等の記述能力を有する言語である。ここでは，属性操作の部分を無視し，文脈自由スケルトンの部分のみに着目することにした。

```
smpsen -> smpcl.
smpsen -> lsmpcl.

lsmpcl -> cl,bunmatsu.
lsmpcl -> nsp,bunmatsu.
lsmpcl -> nsp.

smpcl  -> cl.

cl     -> pp,cl.
cl     -> vmodif,cl.
cl     -> vp.
cl     -> conjp,cl.
cl     -> exp,cl.


pp     -> cl,pp.
pp     -> nmodif,pp.
pp     -> modif,pp.

pp     -> vseq,cl.
pp     -> ppseq,comma.
pp     -> ppseq.
pp     -> pp,pp.
pp     -> bc,ecp_ec.


ppseq  -> np,postp.
ppseq  -> ppseq,postp.
```

図10．日本語の文脈自由文法ルール（一部）

ルールの一部を図10に示す。ルール数は全部で144ある。このルールは，機械翻訳システムの翻訳プログラム開発のために試作されたものであり，quick parsing の評価を意識したものではなく，作成者も，著者以外の人物であることを付記しておく。

実際のルールは，ほとんどChomsky標準形であったので，実験のためにChomsky標準化を行なったがその際の恣意性は問題にしないことにした。しかし，標準化の仕方がルールの分割性に影響を与える[鈴木, 1986]ので，一般には注意が必要である。

5－2．初期記号の選定

図10のルールの初期記号は，smpsen (simple sentence) である。しかし，これは枝分かれルールを持たない。また，chain ルール（A→B のように，右辺がひとつの非終端記号から成る）をたどって得られる lsmpcl や nsp も，それが枝分かれするとき，文末記号に対して分かれる，など，文法の実体とは無関係な枝分かれしか持たない。そこで，本実験では，初期記号として cl (clause，節) を選ぶことにした。図11に示すように，clは文末記号などを除いた文の本体を導出する非終端記号である。

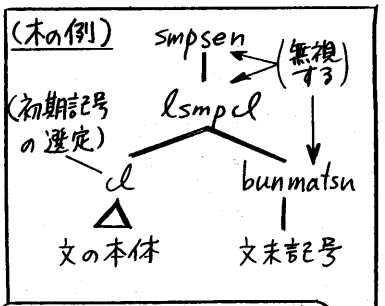

図11．初期記号の選定

5－3．分割可能ペアの計算

以下では，clについて分割性を調べてみよう。clを左辺に持つルールには，(R1)～(R5)の5つがある。

(R1) cl → pp, cl
(R2) cl → vmodif, cl
(R3) cl → vp
(R4) cl → conjp, cl
(R5) cl → exp, cl

pp：後置詞句
vmodif：述部修飾句
vp：述部
conjp：接続詞
exp：感動詞

(R3)を除く枝分れルールについては，到達可能性 Right集合と Left集合とを求めてやる。また，cl→vpについては，chainルールをたどっていって，はじめて枝分れしたところで Right集合と Left集合とを求めてやる。その結果，図12に示す分割可能ペアが求まる。ペアの総数は 609 個になった。

```
<Right(pp),Left(cl)>      ——— (R1)より計算される分割可能ペア
<Right(vmodif),Left(cl)>  ——— (R2)より計算される分割可能ペア
<Right(np),auxv>          ┐
<Right(np),f>             │
<Right(pp),auxv>          │ (R3)より計算される分割可能ペア
<Right(pp),f>             │
<adv,auxv>                │
<adv,f>                   │
<whnoun,f>                │
<verb,a>                  │
<verb,auxv>               │
<verb,conjc>              │
<verb,f>                  ┘
```

(R3)より計算される分割可能ペア：
<Right(pred),Left(vaux)>
<Right(vseq),Left(vaux)>
<adj,Left(vaux)>
<adjv,Left(vaux)>
<Right(vp),vqlif>
<verb,comma>
<Right(vseq),comma>
<adj,comma>
<adjv,comma>

(R4)より ——<Right(conjp),Left(cl)>
(R5)より ——<comma,Left(cl)>

ただし，

```
Right(np) = comma,whnoun,suf,symbol,noun,num2,num,sahen,
            verb,fnl,fn2    のうちのいずれか。(他も同様。)
Right(pp) = ec,cl,comma,q,auxv,conjc,f
Right(pred) = auxv,f,a,conjc
Right(vseq) = auxv,a,conjc,f
Right(vp) = vqlif,comma,verb,auxv,a,conjc,f,adj,adjv
Right(vmodif) = comma,whnoun,suf,symbol,noun,num2,num,
                sahen,verb,fnl,fn2,ec,cl,q,auxv,conjc,
                f,vqlif,adj,adjv
Right(conjp) = comma,conj
Left(cl) = bc,adv,fn2,whnoun,verb,pref,sahen,noun,symbol,
           num_qlif,num,fnl,adj,adjv,det,conj,ex
Left(vaux) = a,auxv,conjc,f
```

図12．分割可能ペアの計算結果

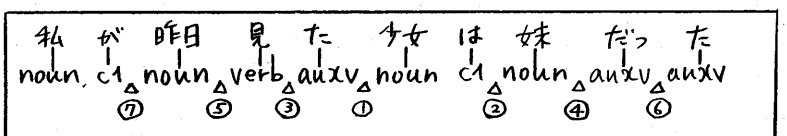

図13．例文の分割結果（数字は分割試行順を示す）

5－4．ルールの修正

図12の分割表を用いて，典型的な例文に対して分割候補点を求めてみた結果を，図13に示す。

図13では，単語数10に対して，分割候補点が7個検出された。これでは，quick parsingを行なった場合，分割の試行回数が多くなるものと予想される。すなわち，到達可能性の利用により，分割の候補がほとんど枝れなかったわけである。

しかし，ここで分割可能ペア計算のもとになった(R1)～(R5)を再検討してみると，(R3)が多様なペアを生成するもととなっていることがわかる。しかも，このルールの意味は，pp, vmodif などの要素が0個以上 vp に係り得るための再帰ルール（R1, R2, R4, R5）の終端子にしかすぎない。そこで，いま仮に，トップ・レベルの cl（文中に埋込まれていない cl）については，必ず1個以上の pp, vmodif 等の係り要素を持つものとしよう。

<Right(pp), Left(vp)>
<Right(vmodif), Left(vp)>
<Right(conjp), Left(vp)>
<comma, Left(vp)>

図14. (R3')から計算される分割可能ペア

私 が 昨日 見 た 女 は 妹 だっ た
noun c1 noun verb auxv noun c1 noun auxv auxv
③ ① ② ④

図15. 例文の分割結果（ルール修正後）

とすると，(R3)の代わりに

$$(R3')\begin{cases} cl \rightarrow pp, vp \\ cl \rightarrow vmodif, vp \\ cl \rightarrow conjp, vp \\ cl \rightarrow exp, vp \end{cases}$$

のルールから分割可能ペアを計算すれば良いことになる。その結果を図15に示すが，Left(vp) = Left(cl) なのでこれらのペアはすでに図12に含まれていることがわかる。従って，ペアの個数はR3派生分(99個)がそっくり減少する。図13の例文で分割候補点を求めてみると，図15のように改善される。

5－5. 解析実験の内容

以上述べたルールと分割可能ペアにもとづき，例文の解析実験を行なった。例文は，比較的単純な文法テスト用例文126と，やや複雑な文28の，計154個である。後者は，たまたま手元にあった情報処理学会誌 1986年11月号の巻頭言[関, 1986]を対象とさせていただいた。154文について1文あたりの平均語数は，12～13語程度である（付属語を含む）。

例文は，あらかじめ図16のような形式で，終端記号の列として用意される。入力文を終端記号化するときの曖昧さは無視して，適当なカテゴリを選んだ。

コンピュータがこの世に出現してから今日までの発達の著しさはあらためて
[noun, c1, det, noun, c1, sahen, auxv, conjc, noun, c1, c2, sahen, c2, adj, c1, adv,
⑬ ⑪ ⑨ ⑦ ⑤
指摘するまでもないが，発達の度合とその内容はハードウェアとソフトウェア
sahen, auxv, a, q, adj, conjc, comma, sahen, c2, noun, c1, det, noun, c1, noun, c1, noun,
③ ① ② ④ ⑥ ⑧
では大分異なっている
c1, adv, verb, auxv]
⑩ ⑫

（[関, 1986]より引用）

図16. 実験対象文の例

解析実験は，各文について次の手順で行なった。まず，図16のように，分割候補点を検出して4－1で述べた戦略に従って分割順序を決定する。続いて，CYK法構文解析を行ない計算コストを求める。さらに，quick parsing を行ない，解が求まるまでの分割回数と計算コストを求め，CYK法と比較する。解析結果の例を図17に示す。

```
[noun,noun,c1,noun,c1,verb,auxv,conjc,comma,noun,c1,noun,c1,verb,auxv] ← 入力文
15 words. ← 入力文中の単語数
CYK starts...
258 elements. ← CYK法構文解析の結果，作成された認識行列中の要素数
                                        （計算コスト）
Quick Parser starts...
8 points. ← 分割候補点の個数＋1
dividing_point(c1,5,9)
reconquer to left.
success_of_reconquer2 th. trial. ← 2度目の分割で解が求まったことを示す。
182 elements. ← quick parsingの結果。
```

図17. 解析結果の例

計算コストとしては，構文解析の結果，作成された認識行列の要素の総数をもって代用する。たとえば，図2の解析例では，計算コスト＝10になる。(別の計算コストを用いることも可能であり，そのための実験を計画中である。)

5－6．実験結果

154文中，文の長さが長すぎたため解析が妥当な時間で終了しなかった8文を除き，146文について図18の解析結果を得た。解析失敗の原因はルール不備である。

- 候補点選定率（154文）――― 34.6%
  1804個所の単語間の場所について，625個所の分割可能点が検出された。
- 平均分割回数（115文）――― 1.23回
  解析成功文115について，合計144回の分割で解が検出されている。

- 計算コストの総和（146文）

	CYK法	quick parsing	両者比
全文 (146)	22213	15575	70.1%
成功文 (115)	18039	11628	64.5%
失敗文 (32)	4174	3947	94.6%

図18．実験結果

6．評価とまとめ

上記の実験結果から，用意したルールと例文については，quick parsingの有効性が確認された。すなわち，解析成功の場合，ほとんどの文で1回目の分割時に解が早期検出されている。好結果の得られた原因は，対象としたルールが，属性操作の部分を含まず，荒いレベルで文法カテゴリを扱っていた（たとえば，動詞は活用形の区別を考えず一律にverbとした）ため，解析時の曖昧性が多くなっており，各分割点で解析が成功しやすくなっていたことにある。この点に関しては，今後，文法カテゴリの構造化により属性を取り込んで，より精緻なルールのもとで実験を行なう必要がある。にもかかわらず，今回の実験は，到達可能性の利用により分割候補点が絞れることを確認した点で，有意義だったものと考える。

また，5－4で示したような，分割性の観点からルールの記述を見なおしてみる試みは，文法の静的評価の一手法とみなせる。この修正により，分割可能ペアが減少し，日本語文法の特徴である文節という文の構成単位が，自然に浮かびあがっている。このような静的評価（部分計算により，ルールの性質を抽出する）の枠組み利用による構文解析アルゴリズムの改良は，今後の興味ある課題である。

さらに，quick parsingは並列処理化が考えやすい。認識行列を共有メモリとした複数プロセッサによる処理を，今後検討していきたい。

参考文献

[関，1986] 関栄四郎：ハードウェアとソフトウェアの発達，情報処理学会誌 vol.27, no.18 (1986)
[Gazdar, 1985] Gazdar, G. et.al.: Generalized Phrase Structure Grammar, Basil Blackwell, (1985).
[鈴木，1986] CYK法構文解析の一検討 ―quick parsingについて，情処33回全国大会(1986).
[鈴木，1987] MELCOM PSE上の自然言語処理開発支援環境について―(1)設計思想と構造変換系
[大槻，1987] 同上 ―(2)文章解析系，情報処理学会 34回全国大会 (1987) (to appear).
[Griffiths, 1965] Griffiths, I. et.al.: On the relative efficiencies of context free grammar recognizer, C.ACM., vol. 8, no. 5 (1965).
[Younger, 1967] Younger, D.H.: Recognition of Context-free languages in time n^3, Inf. Control, vol. 10, no. 2 (1967).
[Earley, 1970] Earley, J.: An efficient context-free parsing algorithm, C.ACM., vol. 13, no. 2 (1970).
[Valiant, 1975] Valiant, L.: General context free recognition in less than cubic time, J. Comput. Syst. Sci., vol. 10 (1975).
[Graham, 1980] Graham, S.L.: An Improved Context-Free Recognizer, ACM Trans. on Prog. Lang. and Sys., vol. 2, no. 3 (1980).
[Pratt, 1975] Pratt, V.R.: LINGOL-A progress report, Advance Papers 4th Int. Joint. Conf. on Artificial Intelligence, Tbilisi, Georgia, USSR (1975).