CYK法構文解析の一検討

3K-2

---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　ソフトウエア開発部

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(1)の，一改良版について報告する．この新アルゴリズムは，分割統治法の概念にもとづくので，quick sortになぞらえてquick parsing と呼ばれる．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，このquick parsing についての概要を述べる．

2．quick parsing の概要

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，Vt，Vn）とする．また，長さnの入力文を，$w=w_1,w_2,\ldots,w_n$ とし，部分文字列を，$w_{ij}=w_i,\ldots,w_j$ と書く．

CYK法では，ダイナミック・プログラミングにより構文解析を行うが，そのための作業領域として，認識行列 t_{ij} を用いる．構文解析終了後，t_{ij} の値は w_{ij} に対する解析結果としての非終端記号の集合になる．CYK法では，この認識行列を，入力文を左から右に順に読み込みながら，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の番号順のようになる．最終的に，t_{1n} の値としてSが求まれば，構文解析の成功である．解析例を図２に示す．

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，Early 法(2)やPratt 法(5)にも共通する特徴であり，バックトラック・ベースのDCG やBUP 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の認識行列を見なおしてみると，t_{15}＝Sという値は t_{11} と t_{25} から求まっているから，t_{12}，t_{13}，および t_{14} の値は実は不要であったことがわかる．すなわち，図３のように，t_{1n}＝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．

アルゴリズムA
（Ａ１）入力文を適当な位置ｉで w_1 から w_i の部分と w_{i+1} から w_n の部分とに分割する．
（Ａ２）それぞれの部分文字列をＣＹＫ法で構文解析する．
（Ａ３）t_{1i} と $t_{i+1\,n}$ から t_{1n}＝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，それはまた後で述べることにする．それよりもこのアルゴリズ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４で番号順に分割点を選んだとしたとき，斜線の四角形の部分を新たに計算すればよい．この四角形の部分の計算は，ＣＹＫ法の一部パラメータの変更で行なえるので，アルゴリズムの詳細は省略する．

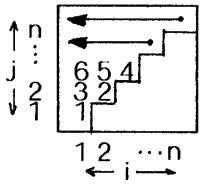


図１．認識行列 t_{ij}

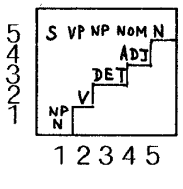
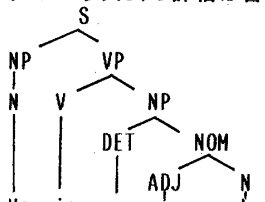


図２．解析例



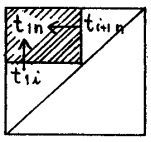


図３．

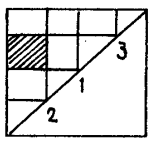


図４．

3．文法の性質の利用による高速化

上記で述べたアルゴリズムは，（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．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のtransitivityの利用を考える．

3－1．分割点の候補検出

まず，初期記号Ｓから枝分れを起こすすべてのルールＳ－＞Ａ，Ｂについて，Ａからの右到達可能性，およびＢからの左到達可能性を調べ，それぞれ到達可能な終端記号の集合Right(Ａ)，Left(B) を得る．ただし，Right (A) ＝｛a｜$A \overset{*}{\Rightarrow} \alpha a$｝である（Leftも同様）．このとき，入力文 w_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－＞Ａ，Ｂに対して分割点の候補として検出することができる（図５）．そこで，入力文の中のすべての分割候補点を，真中に近いものから順に外側に向かって選んでいくことが，よりはやく解を求めるための有効な戦略として考えられる．以下では，この戦略を仮定することにしよう．

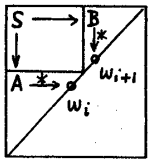


図５．分割候補点

On the CYK Parsing Algorithm
Katsushi SUZUKI, Takashi DASAI
Mitsubishi Electric Corp.

3－2．部分解の利用

さらに，それまでに求めた部分解を利用して，分割候補点の妥当性と，解の早期検出をチェックできる．いま，図6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．このとき，点jを分割点の候補として持つルールの中に，すでに求まっている部分解 $t_{j+1\ m}$（図6の★印）のいずれかの要素を右辺の第2シンボルに持つものが存在しなければ，点jでの分割は無駄に終わることがわかる．（これは，いわゆるPratt チェックに相当する．）さらに，そのようなルールがもし存在すれば，それらのルールの右辺第1シンボルを t_{1i}（図6の○印）に予測できるから，すでに求まっている部分解 t_{1i} と $t_{i+1\ j}$ との値から，その予測値を（jでの分割を試みる前に）たまたま合成できる可能性もある．これは，解の早期検出の可能性をさらに高める．

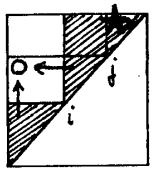


図6．部分解利用

4．例題による検討

4－1．演算子優先順位文法について

式の構文は図7の生成規則で表現される．これは，演算子の優先順位を文脈自由文法で記述したものである．この文法に対して，quick parsing を適用してみよう．　まず，Chomsky 標準形に直したあと，さらに，E－>Tのような右辺がひとつのシンボルのみからなるChain ルールをfolding することにより，初期記号Eに対して枝分れルールを与えるようにすると，図8の生成規則が得られる．これから，3－1で述べたRight(A)とLeft(B) について，その直積を計算すると，E－>A，Bなるすべての A，Bについて，

E－>E＋T　E－>T
T－>T＊F　T－>F
F－>（E）　F－>i

図7．式の構文

Right(A)×Left(B)={<),+>,<i,+>,<),*>,<i,*>,<(,(>,<(,i>}

E－>E，PT．E－>T．
E－>T，MT．E－>i．
E－>(，ER．
T－>T，MF．T－>F．
PT－>+，T．
F－>(，ER．F－>i．
MF－>＊，F．
ER－>E，）

図8．標準形化された規則

が得られる．すなわち，これは $<w_i, w_{i+1}>$ なるiが分割点の候補となることを示している．そこで，たとえば入力文として，((1+2)*3+4)*(5+6+7) の入力文に対しては，$(_7(\ 1_5+2)_3*3_{/}+4)_2*(_4 5_6+6_8+7)$ の添字で示した位置が，その順で分割点の候補として選ばれることになり，2度目の分割で解が求まる．

このように，上記の文法については，分割点の候補が多数求まるので，たまたま上例のごとく，入力文に対する構文木が中央付近で分割される形をしていない限り，それほどの効率向上は望めない．このことは，なかば予想されたことであり，quick parsing の本質的限界でもあるが，この性質を逆に利用することも考えられる．すなわち，入力文を人間が一部修正し，パーザが効率的に解析してくれるようにするのである．たとえば，$1_5+2_3*3_{/}+4_2*5_4+6_6*7$　の入力文に対して，かっこを加えて，$(1_7+2_5*3)_3+(_{/}4_2*5_4+6_6*7)$　のようにすれば，かっこの追加により分割点の候補数は増えたが，3回目の分割で解が求まる（修正前は4回目）．このような修正は，当然のことながら，解析結果の強等価性を保存しないが，そもそも，はじめの解をひとつ求めればよいという状況では，強等価性が問題にされないはずだから，かまわないであろう．また，この修正は，パーザに対する構文解析戦略の指示とみなせる．この指示を機械的に行なうことは，興味ある課題である．

4－2．自然言語の解析について

きわめて単純であり，どれほどの意味があるかはよくわからないが，図9のような日本語の文法規則（骨組みのみ）を考えてみよう．　このとき，Right(A)×Left(B)={<n,v>,<n,n>,<c,v>,<c,n>} になり，「私が$_4$昨日$_2$見た少女は$_{/}$妹$_3$だった」のような分割候補点ができる．しかし，NPの反復に着目してルールを図10のように直すと，分割候補点は{<n,v>,<c,v>} になり，候補の数が減少する．先ほどの例文では，「私が昨日$_{/}$見た少女は妹$_2$だった」になる．すなわち，NPの反復をVPルールで記述していたのを，NGをあらたに非終端記号として設けて，そこで反復させるようにすることにより，NP群とVPの切れめを解析木上で明確にしたのである．

このように，ルールをChomsky 標準化する過程で分割性が定まるので，分割性を考慮した標準化の方法を考える必要がある．

また，上の例は確かに単純であったが，日本語の仮名漢字混じり文の場合，文節間の切れ目がかなり明確であり，文節の反復により文を構成するように文法規則を書くことにより，文節間を分割候補点にすることができて効果が大きいと思われる．しかし，さらに深い検討が必要であろう．

S->NP,VP.
NP->n.
NP->n,COMP. NP->S,NP.
COMP->C.　COMP->c,COMP.
VP->NP,VP.
VP->v.　VP->v,AUXG.
AUXG->a.　AUXG->a,AUXG.
(V ={n,c,v,a},
n:名詞，c:助詞
v:動詞，a:助動詞)

図9．日本語文法の骨組

S->NG,VP.
NG->NP　. NG->NP,NG.
VP->v.　VP->v,AUXG.

図10．NPとVPの分割

5．結論と今後の課題

quick parsing は，文法の性質の利用により，解の早期検出をねらえるのが特徴である．しかも，LRパーザやEarley,Pratt,およびGraham(4)などのleft-to-right のパーザと異なり，右到達可能性も利用できる．その代わり，部分解を用いたPratt チェックの対象になるのは，初期記号を左辺に持つルールの右辺の非終端記号のみである．quick parsing の再帰的適用や，一般の文脈自由文法への拡張は，今後の課題である．

興味深いのは，並列処理への応用が考えやすいことである．入力文を分割し，それぞれについて並列パージングを行うわけであり，その際，認識行列を共有メモリとして，そこで同期をとることになると思われるが，詳細は未検討である．

参考文献

(1) Younger,D.H.:Recognition of Context-free languages in time n^3,Inf.Control,vol.10,no.2(1967).
(2) Early,J.: An efficient context-free parsing algorithm,C.ACM.,vol.13,no.2(1970).
(3) Valiant,L.:General context free recognition in less than cubic time,J.Comput.Syst.Sci.,vol.10(1975).
(4) Graham,S.L.:An Improved Context-Free Recognizer,ACM Trans. on Prog. Lang. and Sys.,vol.2,no.3(1980).
(5) Pratt,V.R.:LINGOL-A progress report,Advance Papers 4th Int.Joint.Conf. on Artificial Intelligence, Tbilisi,Georgia,USSR(1975).