4F-2

$O(n)$時間・$O(n^2)$プロセス数の並列構文解析

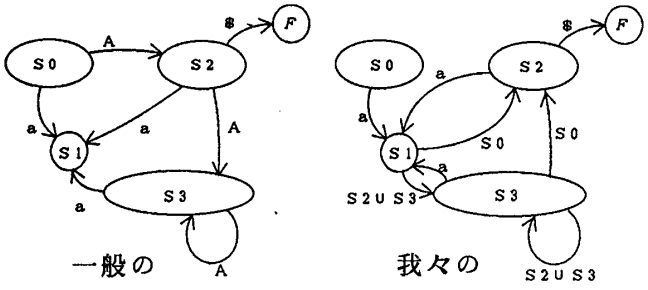
峯　恒憲　　谷口　倫一郎　　雨宮　真人
九州大学大学院　総合理工学研究科

1．はじめに

いままでに並列構文解析アルゴリズムについての報告が多数なされているが、それらには①入力に同期した解析を行うために解析時間が$O(n^2)$（nは入力文の長さ）となる、②プロセス数が指数オーダーになる、といった問題があった。我々のアルゴリズムは一般の文脈自由文法を対象とし、一般のLR法[1]で使用されるshift reduceテーブルに改良を加えたLR状態遷移図を使用して、全ての可能性を並列に試しながら解析を行う。このアルゴリズムは解析時間が$O(n)$ですみ、プロセスの数と使用メモリ領域をどちらも$O(n^2)$に抑えている。

2．LR状態遷移図

一般のshift reduceテーブルから作成したLR状態遷移図には、reduce操作の時に次にどの状態へ遷移すればよいかが明示的でなく、しかも次に遷移する状態へ直接遷移することができないという問題がある。そこでreduce時の遷移情報として状態番号を使用し、この問題を解決した。図1の左に一般のLR状態遷移図を示し、右に我々のLR状態遷移図[2]を示す。



一般のLR状態遷移図　　我々のLR状態遷移図

図1　LR状態遷移図の比較

以下、特に断わらずにLR状態遷移図と述べた時には、我々のLR状態遷移図を指すことにする。

3．アルゴリズムの概要

LR状態遷移図は、非決定性のプッシュダウンオートマトンであり、解析はこの状態遷移図をトレースしながらすすむ。解析にはスタックの代わりにリストをポインタで結んだチェーンリストを使用する。チェーンリスト中のリストをパケットとよぶ。パケットは状態遷移図中の状態番号Siと入力数（読み込んだ単語数）k、チェーンリストの長さfの3つの情報をもっている。本手法では、1つのパケットに1つのプロセスを割り当てている。あるパケット(これをパケットAとする)がポインタで前のパケットを指す時、パケットAに対応するプロセスは活性状態となり、その活性状態はパケットAのポインタをはずすまで続く。活性状態にあるプロセスにはshift/reduceの各操作を行うものと、行わないもの(単にポインタをもつだけのもの)の2種類がある。一般の文脈自由文法から作成したLR状態遷移図では、ある状態でshiftとreduceまたはreduceとreduceの両動作が可能となる場合がある。そのような場合には両動作を並列実行し、その度にshift/reduceの各操作を行うプロセスを生成する。shift操作とreduce操作を並列に実行するため、各プロセスが読み込んだ入力文の長さにずれが生じることがある（図2）が、各プロセスは入力ポインタを持ち次にどの入力を解析するか知っているので、入力に同期しなくても解析には影響しない。

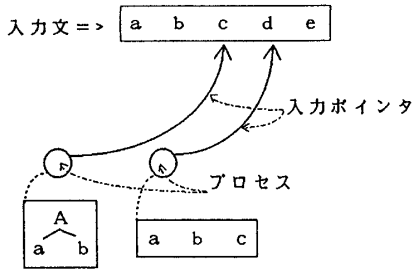


図2

曖昧性の数だけプロセスを生成していくと最悪指数オーダーのプロセスが必要となるので、プロセス間で同期をとって冗長なプロセスの削減を行う。即ち、ある時刻tで、複数のプロセスが同じ状態にきて同じパケットをもつ時、それらのプロセスをひとつだけ生かし、そのプロセ

The Evaluation of Datarol Programs
Tohru TACHIBANA　Rin-ichiro TANIGUCHI　Makoto AMAMIYA
Graduate School of Engineering Sciences, Kyushu University

スに他の殺されるプロセスがもっていたポインタをいっしょにもたせる。

shift操作の時、パケットの入力数が1増えるので、チェーンリスト内のパケットの入力数を統一するために、パケットがもつポインタをコピーする。その様子を図3に示す。図3の表の1つの升目は1つのパケットを表している。

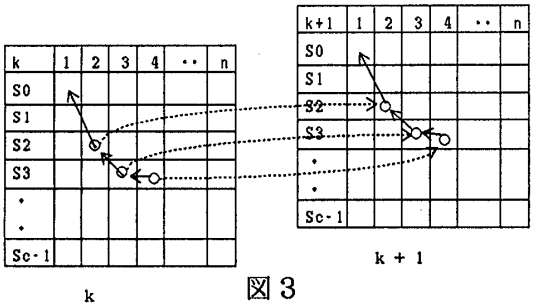
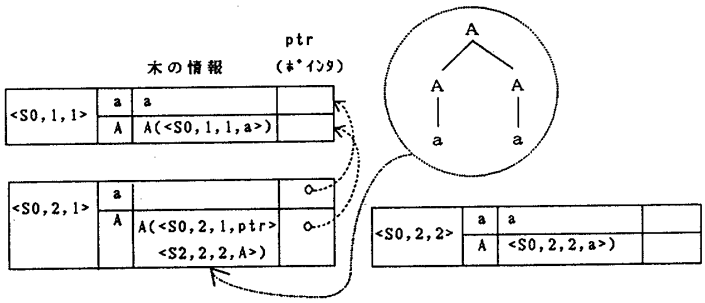


図3

4. 構文木の生成

shift操作の時に、入力数kのポインタをもつパケットのプロセスから、入力数k+1の対応するパケットのプロセスにポインタを渡したが、それといっしょに構文木を貯えているメモリアドレスを指すポインタを渡す。構文木の場合には、(非)終端記号で区分されたメモリにポインタを渡していく。メモリは、構文木情報を格納する場所とポインタを格納する場所の2つをもっている。このメモリは同時読みだし可能な共有メモリで、パケットが絶対アドレスを表している。2章で示した文法A->AA|aの状態遷移図を使って解析を行った時、どのように構文木の情報がメモリに残されていくかを図4に示す。



文法A->AA|aから作成した状態遷移図で
S0->S1->S2->S1->S3->S2と遷移した時に生成される構文木

図4　構文木の作成

5. 評価

入力文の長さnを使って評価する。

①時間　　O(n)

入力に非同期な解析を行っているため、入力に同期した解析と異なって、他のプロセスのreduce動作を終えるのを待つ必要がない。従って、解析時間は一つの構文木を作成する時間に等しく、その時間は木の節の数に比例する、即ち入力文の長さに比例する。木の節を作成するのはshift/reduceの各操作で、両操作とも一定時間で行うことができる[2]ので、解析時間はO(n)である。

②プロセス数　　$O(n^2)$

プロセスは<Si,k,f>に割り当てている。Siの数は文法に依存し、k,fはそれぞれnに比例するので、プロセスの総数は$O(n^2)$である。

③メモリ空間　$O(n^2)$

構文木を作成する時には、パケットを絶対アドレスとし、そのアドレスから(非)終端記号の数だけ領域をとる。各記号に対して、構文木情報とポインタを格納する2つの領域をとる。従ってメモリ空間はパケットの数に依存し、その数は$O(n^2)$であるから、メモリ空間は$O(n^2)$である。

6. 他の手法との比較

M.Tomitaのアルゴリズム[3]とPAX[4]の2つと本手法とを比較した結果[2]を表1に示す。

	時間	プロセス数	メモリ空間
Tomita's	$O(n^2)$	$O(n^2)$	O(n)
PAX	O(n)	$O(C^n)$	--------
本手法	O(n)	$O(n^2)$	$O(n^2)$

表1

7. まとめ

本稿では、全ての構文木の作成をO(n)時間で行いプロセス数とメモリ空間をどちらも$O(n^2)$に抑えるアルゴリズムを示した。今後は、並列に意味処理を行うモデルを考え、プロセス間で同期をとらない並列構文解析アルゴリズム[2]との融合を計り、その実現に努力していく。

参考文献

[1]D.E.Knuth:"On the Translation of Languages from Left to Right", Infomation and Control, Vol. 8, No. 6, pp. 607-639(1965)
[2]峯,谷口,雨宮:"文脈自由文法の並列構文解析", 信学技報, Vol. 89, No.113,pp.1-8 (1989.6.29)
[3]M.Tomita:"An Efficient Augumented_Context_Free Parsing Algorithm", Computational Linguistic, Vol. 13, No. 1-2(Jan-Jun.1987)
[4]Y.Matsumoto:"A Parallel Parsing System for Natural Language Analysis", New Generation Computing, pp. 63-78(May 1987)