高速FB-LTAGパーザとその並列化

吉田 稔　二宮 崇　鳥澤 健太郎　辻井 潤一

東京大学理学系研究科情報科学専攻

1. はじめに

本稿では、FB-LTAG (Feature-Based Lexicalized Tree Adjoining Grammar)により記述された文法の高速なパージング手法について述べる。

FB-LTAGは、TAGと呼ばれる文法枠組を「語彙化」し、「素性構造」を付加した文法枠組である[1]。TAGは、CFGよりも強力な文法枠組として知られているが、そのパージングは高コストであり、CKYスタイルのパーザで$O(n^6)$の計算時間を要することが知られている[2]。FB-LTAGでは、以上の計算コストに加え、素性構造の操作にも大きな計算コストが要求される。

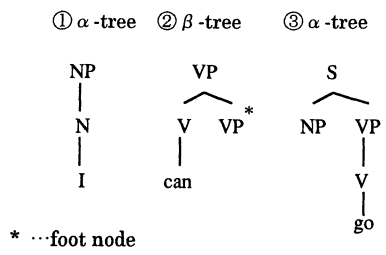
本稿では、FB-LTAGの高速なパーザの実現手法として、「フェーズ分割アルゴリズム」を提案し、さらにこれを並列化することを試みる。我々は、アルゴリズムの設計において、FB-LTAGパーザの実行時間の中で非常に大きな割合を占める「素性構造」の処理部分に注目し、パージングアルゴリズムにおける「素性構造を扱わない部分」と「素性構造を扱う部分」を完全に分割することによって無駄な素性構造計算を省き、全体の実行効率を高めることを目指した。

実験では、University of Pennsylvaniaで開発されたFB-LTAGによる英語文法[1]を用い、並列化の効率とフェーズ分割による性能向上比率を調べた。

FB-LTAGパーザの実装とその並列化については、すでに論文[8]で述べられているが、論文[8]の手法では、素性構造処理の扱いに際して、「フェーズ分割アルゴリズム」を採用していないという点において、本稿のアルゴリズムとは異なっている。

1.1. TAG

TAGでは、基本となる木（Elementary Tree…図1）を組み合わせて構文木を構成していく。Elementary Treeには、α-tree(Initial Tree)とβ-tree(Auxiliary Tree)がある。β-treeは、foot-nodeと呼ばれる、「rootと同一の記号が割り当てられたleaf」を唯一つ持つ。TAGのパージングは、これらのElementary Treeを、SubstitutionとAdjunctionと呼ばれる操作によって組上げ、構文木を構成していくことにより行われる。Substitutionとは、leaf nodeを、同一の非終端記号をrootに持つα-treeで置き換える操作であり、例えば図1における、木③のNPノードを木①のrootで置き換える操作がその例である。Adjunctionとは、β-treeのroot node及びfoot nodeと同一の非終端記号を持つノードに、そのβ-treeを挿入する操作であり、例えば図1における、木③のVPノードを、木②のrootとfootで置き換える操作がその例である。



（図1）Elementary Trees

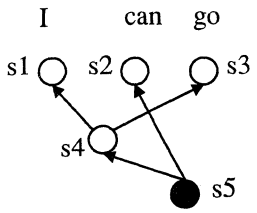
1.2. Earleyスタイルの TAGパーザー

本研究で提案するパージングアルゴリズムは、Earleyスタイルのパージングアルゴリズム[3]をもとにしている。Earleyスタイルのパーザでは、トップダウン解析により不必要な部分木の生成を抑制し、その後ボトムアップに木を構成する（Bottom-Up Parsing）ことにより、パージング全体の効率を上げている。

例として、図1のElementary Treeの集合を文法とし、入力列"I can go"をパーズすることを考える。"I can go"の構文木は、木③と木①でSubstitutionが起こり、さらに木③と木②でAdjunctionが起こることによって完成する。

パージングの各状態はステートと呼ばれる。Bottom-Up Parsingにおいて、各ステートには、そのステートの生成に寄与したステートへのリンク（Course Record）が張られる。"I can go"のパージングでは、図2のCourse Recordが生成されることになる。図2において、s4は"I"を表す木と"go"を表す木がSubstitutionによって組みあがった状態を表し、s5はさらに"can"を表す木がAdjunctionにより挿入され、"I can go"の構文木が完成した状態を表す。文として受理されるステートのことを最終ステートと呼ぶ。図2においては、s5が最終ステートになる。また、パージング開始時に存在する、各単語に割り当てられたステートを初期ステートと呼ぶ。図2においては、s1,s2,s3が初期ステートになる。

[1] FB-LTAGについては、1.3節で解説してある。



(図 2) "I can go"をパーズした場合の
Course Record

1.3. FB-LTAG

FB-LTAG は、TAG を、「語彙化」し、「素性構造」を扱う形式に拡張した文法枠組である。ここでいう「語彙化」とは、各 Elementary Tree のある一つの leaf node に、必ず、anchor と呼ばれる終端記号を割り当てることである。パージングの際には、入力文中の単語を anchor として持つ Elementary Tree のみを使うことによって、木の数を削減する。 「素性構造」は、Elementary Tree の各 node に付加され(図 9,10 参照)、人称・時制の不一致などを検出する役割を持つ。

2. 素性付き TAG の並列パージング

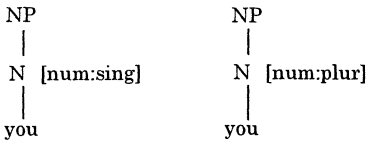
素性構造処理を TAG パーザに付加する際に、2 種類のストラテジーが考えられる。一つは、ステート生成の際に素性構造も同時に処理する方法であり、既存のパーザ[8]はこの方法を用いている。それに対し、本研究で提案する方法では、パージングの過程を 2 つのフェーズに分割し、最初のフェーズで素性構造情報の処理を伴わないパージングを行い、後のフェーズでは、その結果を利用して、必要となるステートでのみ素性構造処理を行う。この方法を採用することにより、以下の利点が生じる。

1. 素性構造処理回数の減少

パージングの最中には数多くのステートが生成されるが、パーザが最終的に出力する構文木に寄与するステートは、そのうちのごく一部である。これ以外のステートに対する素性構造処理を省くことにより、全体の素性構造処理の回数を著しく減少させることが可能となる。

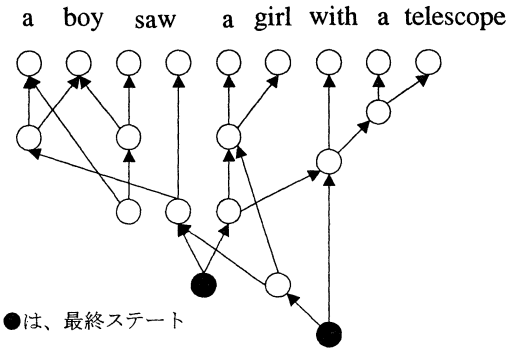
2. 素性値を起因とする曖昧性の処理の省略

パージングの際に素性構造を扱うと、「同一形状だが、付加される素性構造が異なる木」を区別する必要が生じるため (図 5) 、結果として扱う木の数が増加し、それを区別するためにステートの数も増加してしまう。素性構造の処理を後回しにすることによって、このようなステート数の増加を抑えることができる。最初のフェーズでは素性構造を考慮に入れないので、素性の値による曖昧性は発生せず、図 5 の木は一つにまとめて扱うことができる。最初のフェーズで最終ステートの生成に寄与するステートに対してのみ、素性構造による曖昧性を考慮すればよい。



(図 5)
同一形状で、付加する素性の値の異なる木

以下では、最初の、素性構造を処理しないパージングを行なうフェーズを Phase I と呼び、その後素性構造の処理を行うフェーズを Phase II と呼ぶ。そして、PhaseI の結果、図 6 の Course Record が生成されたと仮定し、PhaseII の解析過程を解説する。

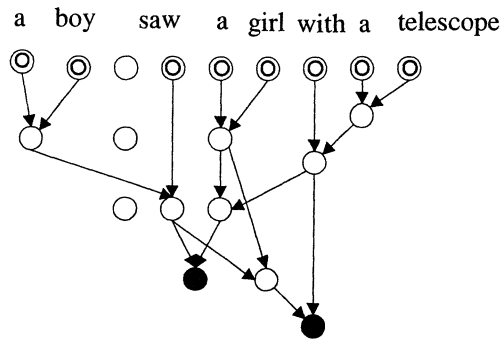


(図 6) Course Record の例

Phase II は、さらに 2 つのサブフェーズに分けて考えられる。まず、Phase I のパージング結果から、「最終ステートの生成に寄与したステート」だけを選択するサブフェーズ (Phase 2.1) と、その探索結果から実際に素性構造操作を行うサブフェーズ (Phase 2.2) である。

2.1. Phase 2.1

まず、図 6 の結果から、最終ステートをすべて選ぶ。それらのステートの生成に寄与した初期ステートを、リンクを辿って探索する。リンクを辿る際には、逆向きのリンクを新たに張る。その結果、図 7 のリンクが生成され、最終ステートに到達可能な初期ステートの集合が求まることになる。



◎は、最終ステートに到達できる初期ステート

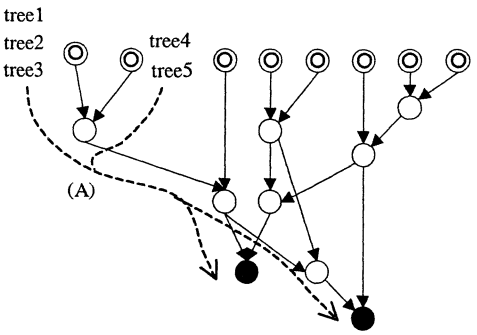
(図 7) Course Record から生成された
新しいリンク

2.2. Phase 2.2

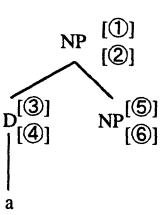
Phase 2.2 は、Phase 2.1 で作成されたグラフをもとに、木の構成をシミュレート[2]していくサブフェーズであり、構文木の生成と同時に、木に付加された素性構造の処理も行なう。

図 8 に、Phase 2.2 の動作の一部を図示する。パーザは、Phase 2.1 で得られた初期ステートそれぞれに、Elementary Tree を割り当てる(図 8 の tree1,2,3,4,5)。図 9 は、Elementary Tree の一つを表す。木の各ノードにはそれぞれ素性構造が付随する (図 9 の①～⑥) 。例として、図 9 の②に付随する素性構造の詳細を図 10 に示す。Phase 2.2 のパージングは、各 Tree がリンクを辿ってステートからステートに渡されることにより行われる。図 8 では、木の進路は破線で示されている。このとき、ステートが複数のステートから生成されている場合 (図 8 の(A)) は、木が複数ステートから進んでくることになるので、Substitution や Adjunction によって複数の木を 1 つに組上げる。以上は局所的な例であるが、各初期ステートに割り当てられた木全てを同様に処理し、最終ステートに到達した木が正解となる。

[2] 「木の構成のシミュレート」とは、木の構成に伴う素性構造の変化を計算してゆくことである。本文中の記述では、便宜上、実際に木が組上がるものとして説明している。



(図 8)Phase 2.2 の動作の一部



(図 9) tree1 の木構造

wh:<1>[]
decreas:<2>[]
compl:<3>[]
gen:<4>[]
card:<5>[]
quan:<6>[]
definite:<7>[]
const:<8>[]
agr:<9>[]
case:<10>nom/acc
conj:<11>[]
displ-const:<12>[]

(図 10)tree1 の素性構造(②)の詳細

2.3. 並列処理

並列化は、上で述べた Phase I, Phase 2.1, Phase 2.2 の 3 段階それぞれについて行うことになるが、現在は Phase2.2 の並列化のみが実装されている。

Phase 2.2 の動作は、図 8 のように、素性構造付き部分構文木 (図 9 参照) を、リンクに従って、初期ステートから最終ステートまで流して行くというモデルでとらえることができる。本研究の並列化においては、各ステートの各構文木ごとにスレッドを一つずつ割り当てる。つまり、各構文木が並列の単位になっており、並列度は同時に存在する Tree の数に比例する。

(表 1) フェーズ分割の有無による違い

	一文あたりの平均時間(sec)	一文あたりの平均 State 数
分割無	87.453	25964.6
分割有	3.082	
(Phase I)	1.367	15275.0
(Phase II)	1.715	507.0

(表 2) 異なる台数での実行時間(単位：sec)

台数	Phase I	Phase 2.1	Phase 2.2
1	7.338	0.03	6.441
10			0.789
20			0.452
30			0.360
40			0.342
50			0.330
60			0.351

3. 実験

並列プログラミングライブラリ StackThreads/MP を用い、パーザの実装を行なった[3]。StackThreads/MP は、スレッドの生成に伴うオーバーヘッドが極めて軽いという特長を持つため、ダイナミックなスレッド生成を頻繁に行う本稿のアルゴリズムに適している。また、素性構造処理の実装は、高速素性構造処理機構 LiLFeS[4]と、並列素性構造処理機構 PSTFS[5]を用いて行なった。

実行環境は、Sun Ultra Enterprise 10000 (UltraSPARC 250MHz×64, 8GByte)である。フェーズ分割を行なわない場合と行なった場合の速度を比較するため、5 文（平均語数 4.8）をパーズさせた時の、実行時間と生成ステート数を計測した（表 1）。ただし、Phase II の平均ステート数は、Phase I で生成されたステートのうち、Phase 2.1 で枝狩りされなかったステートの総数を表している。現在は、フェーズ分割を行なった場合と行なわない場合で、約 30 倍の速度差がある。

次に、Phase 2.2 の台数効果を測定する。10 語の文を、実行台数を 1 台から 60 台まで変化させて、パーズ時間を測定した結果が表 2 である。台数効果は最大で 19.5 倍に達した。

また、現存する FB-LTAG パーザで最も有名な XTAG システム[9]のパーザとの比較を行う。XTAG システムを UltraSPARC 167Mhz の上で実行し、6 語の文をパーズさせたところ、約 50 秒でパーズが終了した。同じ文を、UltraSPARC 250Mhz 上で本稿のパーザを用いてパーズしたところ、3.94 秒でパーズが終了した。このことより、マシン環境の違いを考慮に入れても、既存のシステムと比較して十分高速なパーザが実現できたといえる。

[3] 現在、Phase 2.2 における Course Record 記録部の実装は完了していない。しかし、Course Record の記録は、ステート生成の際に若干量の配列書き込みで実現可能であるため、計算時間に大きな影響はないと推測される。

4. 結論と今後の課題

本稿で解説したフェーズ分割アルゴリズムを用いることにより、高速な FB-LTAG パーザを実現した。また、一部を並列化し、約 20 倍の台数効果を得ることができた。

Phase I の並列化についても現在実装を進めており、これを完成させることが当面の課題である。また、素性構造処理をより効率化することにより、さらなる高速化が期待できる。

5. 謝辞

パーザの実装に関して多くの助言を頂いた本学の田浦助手と、英語文法に関して教えを頂いた研究室の建石氏に感謝致します。

参考文献

[1] The XTAG Research Group. *A Lexicalized Tree Adjoining Grammar for English.* Technical Report MS-CIS-90-24, Department of Computer and Information Science, University of Pennsylvania.

[2] K.Vijay-Shanker and Aravind K.Joshi. *Some computational properties of Tree Adjoining Grammars.* Proc. ACL'85

[3] Yves Schabes and Aravind K.Joshi. *An Earley-Type Parsing Algorithm for Tree Adjoining Grammars.* Proc. ACL'88

[4] Takaki Makino and Minoru Yoshida and Kentaro Torisawa and Jun'ichi Tsujii. *LiLFeS --- Towards a Practical HPSG Parser.* Proc. COLING-ACL'98.

[5] Takashi Ninomiya and Kentaro Torisawa and Jun'ichi Tsujii. *An Efficient Parallel Substrate for Typed Feature Structures on Shared Memory Parallel Machines* . Proc. COLING-ACL'98

[6] Tom Nurkkala and Vipin Kumar. *A Parallel Parsing Algorithm for Natural Language Using Tree-Adjoining Grammar.* Proceedings of the International Parallel Processing Symposium, 1994.

[7] Tom Nurkkala and Vipin Kumar. *The Performance of a Highly Unstructured Parallel Algorithm on the KSR1.* Scalable High Performance Computing. Conference, May 1994, Knoxville.

[8] Tom Nurkkala. *Parallel Algorithms for a Highly Unstructured Problem: Natural Language Parsing Using Tree Adjoining Grammar.* Ph.D. Thesis, University of Minnesota, 1997.

[9] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. *XTAG System - A Wide Coverage Grammar for English.* Proc. COLING '95.