HPSGの複数の文脈自由文法へのコンパイル

西田 健二　鳥澤 健太郎　辻井 潤一
東京大学大学院情報科学専攻
{nishiken,torisawa,tsujii}@is.s.u-tokyo.ac.jp

1 はじめに

本研究では，HPSG[1]の枠組みで書かれた文法からコンパイルされた複数の文脈自由文法を組み合わせて構文解析時に使用し，構文解析速度の向上を図る．

HPSGは素性構造[2]と呼ばれるデータ構造を語彙項目，文法規則の表現に使用している．そのため，HPSGを用いた構文解析では，素性構造の単一化という時間のかかる計算を多用する必要がある．鳥澤らのHPSGのための構文解析アルゴリズム[3][4]では，HPSGの構文解析における効率に主眼をおき，あらかじめHPSG文法を文脈自由文法に近似的にコンパイル，文脈自由文法によって素性構造の単一化の成否を単一化前に予測する2フェーズパージングの手法によって，HPSG文法を使った構文解析の高速化に成功している．

しかしながら，オリジナルのHPSGが大規模かつ複雑になるにつれて，コンパイルされた文脈自由文法のサイズが巨大化し，使用メモリ等において実用的ではなくなる．従来は文脈自由文法の解析性能を落とすことによって，文脈自由文法のサイズを小さくしてこの問題を回避していた．しかしながらこの方法では構文解析の速度が低下し，高速化の障害となる．この問題に対し，論文[5]ではArray Unificationという文脈自由文法とは別の解析機構を加えることでこの問題に対処している．また，論文[6]では異なるコンパイル手法を用いて文脈自由文法のサイズを小さくする試みを行っている．今回提案する手法では，HPSGを複数の互いに異なる文脈自由文法にコンパイルし，構文解析時に組み合わせて使用することにより，使用する文脈自由文法のサイズを大きくすることなく構文解析の高速化を図る．

実験では，XHPSG[7]文法を文脈自由文法にコンパイルして使用し，1つの文脈自由文法を使用した場合と，2つの文脈自由文法を使用した場合を比較検証する．

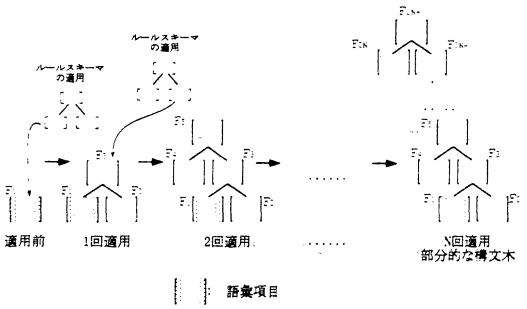

図 1: 部分的な構文木

2 HPSG の文脈自由文法へのコンパイルと構文解析

HPSGにおける構文解析は，素性構造で記述される語彙項目に対し，文脈自由文法でいうところの書き換え規則に相当するルールスキーマと呼ばれる素性構造を再帰的に繰り返し適用することで行われる．

鳥澤のアルゴリズムにおいて，HPSGの文法は，次の2ステップで文脈自由文法にコンパイルされる[1]．

Step 1 単一の語彙項目を主辞として，適用可能なルールスキーマを再帰的に繰り返し適用し，部分的な構文木を生成する (図1) [2]．

Step 2 部分的な構文木を文脈自由文法の非終端記号とし，部分的な構文木同士の単一化可能性をチェックし，文脈自由文法の書き換え規則を導出する．書き換え規則には適用されるルールスキーマ名が，例えば $A \xrightarrow{r} BC$ (r はルールスキーマの名前) のように付属している．

構文解析は，2つのフェーズから構成される．フェーズ1はコンパイルされた文脈自由文法を用いて，ボトムアップに構文木の候補を構成し出力する．フェーズ2では，フェーズ1で出力された文

[1] この部分の詳細については，論文[4]を参照されたい．
[2] 無限に適用可能な場合もありうるが，$F_{2n-1}, F_{2n}, F_{2n+1}$ が同一なものは1つにまとめて数える．

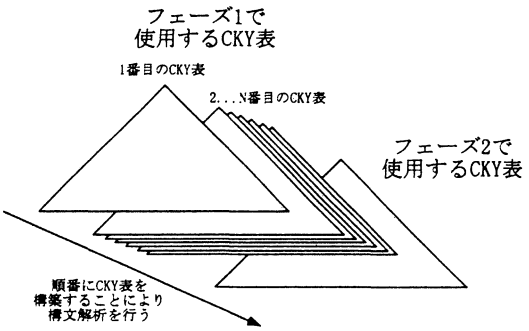

図 2: 構文解析に使用する CKY 表

脈自由文法の構文木を利用して，より少ない単一化の使用で構文木を完成させる．

2.1 非終端記号数と構文解析速度

鳥澤のアルゴリズムを用いた HPSG 文法のコンパイルでは，実用上の問題から生成される文脈自由文法の非終端記号数を抑制するため，語彙項目とルールスキーマからユーザーが選んだいくつかの素性を無視してコンパイルする．このことによって，文脈自由文法の非終端記号数を抑制することはできるが，取り除かれた素性は生成された文脈自由文法には反映されない．反映されなかった部分は，フェーズ 2 で実際に単一化を使用して単一化の可否を判定することになるので，構文解析の速度を低下させる要因となる．

本研究では，取り除く素性を違えて複数の別個の文脈自由文法にコンパイルして，構文解析に使用する．

3 複数の文脈自由文法を利用する 2 フェーズパージング

コンパイルされた複数の文脈自由文法を構文解析において扱うために，2 フェーズパーザーのフェーズ 1 の部分を変更する．新しいフェーズ 1 は，複数の文脈自由文法をそれぞれ別個の CKY 表において用いて構文解析をし，いずれの文脈自由文法でも可能な構文木の候補を出力する．

3.1 構文解析の流れ

フェーズ 1 は，N 個の文脈自由文法を使用する場合，それぞれの文脈自由文法を使用し N 個の

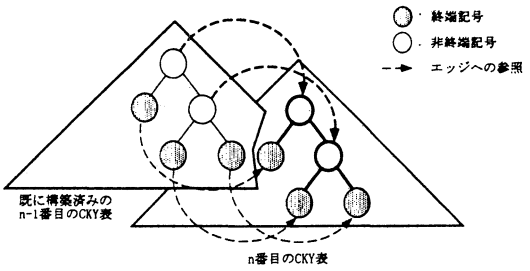

図 3: $n(n \neq 1)$ 番目の CKY 表の作成

CKY 表を構築する．文脈自由文法を使用する順番はユーザーが指定する．

1 番目の CKY 表は，従来のフェーズ 1 と同様にボトムアップ CKY パーザーで CKY 表を構築する．$n(n \neq 1)$ 番目の CKY 表の構築は，$n-1$ 番目の CKY 表の結果を参照しつつ構築していく．図 2 に示すように，構文解析では合計で $N+1$ 個の CKY 表を使用する．

フェーズ 1 において $n(n \neq 1)$ 番目の CKY 表は $n-1$ 番目の CKY 表をもとに構成するため，$n-1$ 番目の CKY 表のエッジは n 番目の CKY 表のエッジを参照するためのリンクを持つ必要がある．そのため本パーザーで用いるエッジは

- 記号又は素性構造
- 娘と適用されたルールスキーマの組の集合
- 次の CKY 表でリンクされるエッジへの参照の集合

の 3 つ組により構成される．

構文解析の流れは以下の通りになる．

- フェーズ 1 で使用する N 個の CKY 表とフェーズ 2 で使用する CKY 表のそれぞれに終端記号及び語彙項目をセットし，同じ語彙項目に対応する終端記号をフェーズ 1 で使用する 1 番目の CKY 表からフェーズ 2 で使用する CKY 表まで順番にリンクして繋ぐ．
- 1 つめ CKY 表を従来のフェーズ 1 と同様にボトムアップに構築していく．
- n 番目 $(n \neq 1)$ の CKY 表を，$n-1$ 番目の CKY 表での結果をもとに以下の通りに構築する．
  - $n-1$ 番目の CKY 表の構文木をトップダウンに探索していき，到達可能なエッジにマークをつける．

- マークされたエッジから，n番目のCKY表のエッジを参照し，ボトムアップにn番目のCKY表を構築する．以下の順番でn番目のCKY表のエッジを以下の要領でボトムアップに構築していく(図3)．
  1. $n-1$番目のCKY表の子からn番目のCKY表の子を参照する．
  2. $n-1$番目のCKY表のエッジ構築に使用されたルールスキーマ名が使用されている書き換え規則をn番目のCKY表の子に適用する．
  3. 適用に成功した場合，$n-1$番目のCKY表の親からn番目の親にリンクを張って参照できるようにする．

- フェーズ2のCKY表をN番目のCKY表の結果をもとにn番目$(n \neq 1)$のCKY表と同様に構築していく．違いは実際に素性構造の単一化を用いて構文木を組み立てていくことである．

3.2 いくつかの注意点

3.2.1 エッジのリンク方法とファクタリング

1つのフェーズ2で使用されるCKY表のエッジに対して，フェーズ1の最初のCKY表のエッジからフェーズ2のCKY表のエッジまでがリンクによって繋がれる．このとき同一のCKY表において同一の非終端記号をもつエッジをファクタリングによって1つにまとめることができる．しかしながら，$n-1$番目と$n+1$番目のCKY表において非終端記号が異なっていて，n番目のCKY表において非終端記号が同一でファクタリングされている場合，$n+1$番目のCKY表の構築においてはn番目のエッジからたどれるすべてのエッジに関して計算しなくてはならない．

このような無駄を避けるために，2フェーズパーザーにおいては，$1 \ldots n-1$番目のCKY表において同一の非終端記号をもつ場合に限り，n番目のCKY表のエッジをファクタリングしている．

3.2.2 1つの非終端記号にまとめた場合

異なる複数の文脈自由文法の非終端記号をまとめて，1つの非終端記号として扱う方法も考えられる．これにより解析速度の向上する文脈自由文法も存在しうるが，我々の実験ではかえって速度が低下した．この理由は，

ファクタリングでまとめられるエッジ数が減少し，フェーズ1で生成されるエッジ数が増大する．このため構文解析の効率はむしろ悪化してしまう．

	(a)	(b)	(c)
非終端記号の数	1682	4821	7694
書き換え規則の数	47516	1537495	2035616

表 1: 実験に使用した文脈自由文法

文脈自由文法	フェーズ1	フェーズ2	全体
(a)	46	770	816
(b)	81	520	601
(a)+(b)	64	110	174
(c)	62	109	171

表 2: 1文あたりの平均構文解析時間(msec)の比較

4 実験

本研究で提案した方法の有効性を示すため，XHPSG[7]をコンパイルして3つの異なる文脈自由文法(a),(b),(c)を作成した．文脈自由文法(a),(b)は，それぞれ異なる素性をルールスキーマ及び語彙項目から取り除いてコンパイルしたものである．文脈自由文法(c)は，文脈自由文法(a),(b)で取り除かれている素性を取り除かずにコンパイルしたものであり，複数の文脈自由文法を使った場合のオーバーヘッドを検証するために用いる．文脈自由文法(a),(b),(c)の非終端記号数と書き換え規則の数は，表1に示す通りである．

2フェーズパーザーを使用した構文解析実験は以下の4つの場合で行った．

1. 文脈自由文法(a)を使用
2. 文脈自由文法(b)を使用
3. 文脈自由文法(a)と(b)両方を使用
4. 文脈自由文法(c)を使用

構文解析に使用した文はATISコーパスから抜き出した500文の英文で，平均語長は7.4語である．素性構造の処理はLiLFeS[8][9]を使用している．実行環境は，Sun Ultra Enterprise(Ultra Sparc II 336MHz, Solaris 7, メモリ 6 GBytes)である．

1文当りの平均構文解析速度は，表2に示すとおりである．文脈自由文法(a),(b)両方を使用した場合，(a)のみの場合の4.7倍，(b)のみの3.5倍の速度をだすことに成功している．文脈自由文法(c)と比べた場合も，複数の文脈自由文法を使用することによるオーバーヘッドはほとんどみられなかった．

文脈自由文法(a),(b)を組み合わせた場合のフェーズ1にかかる時間が文脈自由文法(a),(b)を単独に使用した場合のフェーズ1にかかる時間の和と比べて短い理由は，表3を参照されたい．この

文脈自由文法	エッジ数
(a)	7461
(b)	19567
(b)((a) の適用後)	2947
(c)	4711

表 3: フェーズ 1 で生成されたエッジ数の比較

文脈自由文法	メモリ使用量 (MB)
(a)	171
(b)	431
(a)+(b)	481
(c)	845

表 4: システム全体のメモリ使用量の比較

表はフェーズ 1 において生成されるエッジ数[3] を比べたものであるが，文脈自由文法 (a) を用いて構築された CKY 表を利用した場合，文脈自由文法 (b) を用いて CKY 表を構築する際，生成しなければならないエッジの数が文脈自由文法 (b) を単独で使用した場合に比べて少なくて済む．このためフェーズ 1 にかかる時間がそれほど長くならないのである．

また使用メモリに関して，コンパイルされた文脈自由文法を格納するために非終端記号の 2 乗メモリ領域を要すると仮定すると[4]，文脈自由文法 (a),(b) のメモリ使用量は，文脈自由文法 (c) のメモリ使用量の 44% である．実際のシステムでのメモリ使用量の比較は表 4 で行っている．文脈自由文法 (a),(b) を使用した場合，文脈自由文法 (c) を使用した場合の 57% のメモリ使用量である．

5 結論

本論文で解説した手法を用いることにより，HPSG 文法から複数の互いに異なる文脈自由文法コンパイルし，それらの文脈自由文法を組み合わせて使用することにより，構文解析の高速化を達成することに成功した．

6 謝辞

本論文を執筆するにあたり，多くの助言を頂いた光石氏，二宮氏をはじめとする辻井研究室の方々に感謝致します．

[3] ファクタリングを無視した場合の数である
[4] 単純に 2 次元配列に書き換え規則を格納した場合である

参考文献

[1] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press and CSLI Publications, 1993.

[2] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, Cambridge, England, 1992.

[3] Kentaro Torisawa and Jun'ichi Tsujii. Computing phrasal-signs in HPSG prior to parsing. In *COLING 96*, pages 949–955, 1996.

[4] Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. An HPSG parser with CFG filtering. *To appear in the journal of Natural Language Engineering*, 2000.

[5] Kenji Nishida, Kentaro Torisawa, and Tsujii Jun'ichi. An efficient HPSG parsing algorithm with array unification. In *Proceedings of the Natural Language Processing Pacific Rim Symposium 1999 (NLPRS'99)*, pages 144–149, 1999.

[6] Bernd Kiefer and Hans-Ulrich Krieger. A context-free approximation of head-driven phrase structure grammar. In *To appear in the Proceedings of International Workshop on Parsing Technologies*, 2000.

[7] Yuka Tateisi, Kentaro Torisawa, Yusuke Miyao, and Jun'ichi Tsujii. Translating xtag english grammar to hpsg. In *Proceedings of Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, pages 172–175, 1998.

[8] Takaki Makino, Minoru Yoshida, Kentaro Torisawa, and Jun'ichi Tsujii. LiLFeS — towards a practical HPSG parser. In *COLING-ACL'98 Proceedings*, pages 807–811, August 1998.

[9] Yusuke Miyao, Takaki Makino, Kentaro Torisawa, and Jun'ichi Tsujii. The lilfes abstract machine and its evaluation with the lingo grammar. *To appear in the journal of Natural Language Engineering*, 2000.