FB-LTAG から HPSG への文法変換

吉永 直樹† 宮尾 祐介‡ 建石 由佳‡ 鳥澤 健太郎‡* 辻井 潤一‡
†東京大学理学部情報科学科 ‡東京大学大学院理学系研究科情報科学専攻 *さきがけ研究 21
{yoshinag, yusuke, yucca, torisawa, tsujii}@is.s.u-tokyo.ac.jp

1 はじめに

本稿では, Lexicalized Tree Adjoining Grammar (LTAG) [1] から Head-driven Phrase Structure Grammar (HPSG) [2]-like な文法への文法変換アルゴリズムを提案する. ここで HPSG-like な文法とは, 素性構造 [3] で記述された語彙項目と文法規則から成る文法を指す. 本研究の最終目的は, 任意の LTAG 文法を Sag らにより提案された HPSG の枠組に変換することであり, 本稿ではその第一段階について報告をする.

LTAG 文法から HPSG 文法への変換手法は, 建石ら [4] によりすでに提案されている. しかしそのアルゴリズムは変換対象である XTAG 英文法 [5] に依存しており任意の LTAG 文法に対して適用することはできない. これに対し本研究では, 変換可能な文法の条件を明確に与えた上で, その条件を満たす任意の LTAG 文法に対する形式的な文法変換のアルゴリズムを提案する.

LTAG 文法を HPSG 文法に変換することによる利点の一つとして次のような点が挙げられる. 現在 HPSG 文法に対しては高速なパーザが開発されており, それが LTAG パーザより高速であることが報告されている [6]. 従って HPSG 文法が既存の大規模 LTAG 文法から自動的に得られるのであれば, 様々な最適化が施された高速な HPSG パーザ [7, 8, 9] を LTAG 文法を用いたパージングに利用することができるようになる.

2 背景

2.1 Lexicalized Tree Adjoining Grammar (LTAG)

TAG [10] は基本となる木 (*elementary tree* : 図1) を組み合わせて構文木を生成する文法枠組である. elementary tree の各ノードには終端記号または非終端記号が付与されている. elementary tree は *initial tree* と *auxiliary tree* に分類され, 前者は非終端記号が付与された中間ノードと, 非終端記号または終端記号が付与された葉ノードからなる. 後者は根ノードと同じ非終端記号が付与された *foot node* と呼ばれる葉ノードを一つ持つ. elementary tree 中で非終端記号の付与された葉ノードを *substitution node* と言う. 全ての elementary tree に対し, 終端記号の付与されたノードが必ず一つ以上存在する TAG を LTAG と呼び, そのノードのことを anchor と呼ぶ.

TAG の構文解析は以下の 2 種類の操作によって木同士を結合していくことにより行われる (図1).

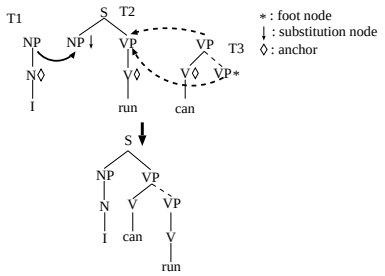


図 1: TAG の elementary tree と結合操作

substitution

substitution node を, そのノードの非終端記号と同一の非終端記号を付与された根ノードを持つ initial tree によって置き換える操作 (図1 の T1 と T2 の結合を参照).

adjunction

auxiliary tree の根ノード, foot node と同一の非終端記号を付与されたノードにその auxiliary tree を割り込ませる操作 (図1 の T2 と T3 の結合を参照).

TAG における構文解析の操作は以上の 2 種類のみであるため, これらを HPSG-like な文法における文法規則で模倣出来るように変換を行う (3.3 節参照).

2.2 Head-driven Phrase Structure Grammar (HPSG)

HPSG [2] は, 型付き素性構造 [3] で語彙項目と文法規則を記述する単一化ベースの文法枠組である. 本稿が変換の目標としている HPSG-like な文法は以下の条件を満たすものとする.

1. 各単語に対して素性構造で記述された語彙項目が割り当てられる. 語彙項目にはその語の下位範疇化要素や語特有の振る舞いを決定する情報が記述される.
2. 娘から親を生成する文法規則 (スキーマ) は親と娘との満たすべき関係を定めた, 素性構造で記述された規則からなる. スキーマはあくまで親と娘の一般的関係を規定するもので, 語特有の情報は記述しない. すなわち, スキーマの数が少ない方が望ましい.

Sag らによって提唱された HPSG [2] は, 以上の条件を満たしているが, それらに加え様々な原理 (プリンシプル) を与えることにより, 言語学的理論を構築している.

3 アルゴリズム

3.1 等価性の条件及び入力と出力

LTAG では木構造を一括して与えるのに対し, HPSG ではスキーマにより木構造を一段ずつインクリメンタルに再現していかなければならない. このインクリメンタルな木構造の再現において文法の等価性を保証するためには以下の条件を満たすことが必要である.

条件 **1** LTAG において各 anchor から根ノードに至るパス (これを *trunk* と呼ぶことにする) に沿ってボトムアップにスキーマを適用していくことで elementary tree と等価な木構造が得られること. ここで言う等価とは LTAG 文法を使って構文解析した結果の構文木と本手法で変換した HPSG-like な文法を使って構文解析した結果の構文木との間に 1 対 1 の対応関係があるということである.

条件 **2** 条件 1 で再現された elementary tree の根ノードは elementary tree の substitution node と単一化できること (これは substitution を模倣するための要件である).

条件 **3** 条件 1 の再現の途中で他の elementary tree による adjunction が模倣できること.

本変換アルゴリズムの入力は LTAG の elementary tree, 出力は HPSG-like な文法の語彙項目及びスキーマである. ただし入力に対しては以下の仮定を満たす elementary tree を今回の変換の対象とする.

仮定 **1** trunk 上のノード以外は全て葉ノードである.

仮定 **2** anchor は 1 つしか存在しない.

LTAG 文法の elementary tree は, イディオムに対する木を除けばその多くが 1 つしか anchor を持たない. 従ってこの仮定を置くことで変換の結果得られる文法の記述力が大きく損なわれることはない.

これらの仮定を満たす elementary tree を変換の対象とすることで次節のように定式化が容易になる.

3.2 elementary tree の変換

任意の elementary tree は二分木に 1 対 1 に変換することができるので以下では二分木のみを考えることとする.

まず根ノードの深さを 0, anchor の深さを d として木の深さ i の trunk 上のノードに付与された非終端記号を n_i と記述する. 次に深さ i の葉ノードに付与された非終端記号を l_i と記述する. 葉ノードの trunk から見た位置を D_i を以下のように定義する.

$$D_i = \begin{cases} left & (\text{葉ノードが } trunk \text{ から見て左にある場合}) \\ right & (\text{葉ノードが } trunk \text{ から見て右にある場合}) \\ null & (\text{葉ノードがない場合}) \end{cases}$$

また深さ i の葉ノードに対し, T_i を以下のように定義する.

$$T_i = \begin{cases} - & (\text{葉ノードが } substitution\ node \text{ の時}) \\ + & (\text{葉ノードが } foot\ node \text{ の時}) \end{cases}$$

HPSG 文法では trunk に沿ってボトムアップにスキーマを適用していくことで構文木を構成することから, HPSG-like な文法の語彙項目は, 以下のような素性構造として与えられる (図 2).

$$\begin{bmatrix} Sym : n_d \\ Arg : \left\langle \begin{bmatrix} Sym & : n_{d-1} \\ Leaf & : l_d \\ Dir & : D_d \\ Foot? & : T_d \end{bmatrix}, \ldots, \begin{bmatrix} Sym & : n_0 \\ Leaf & : l_1 \\ Dir & : D_1 \\ Foot? & : T_1 \end{bmatrix} \right\rangle \end{bmatrix}$$

この語彙項目は *Arg* 素性として elementary tree の組み上げに必要な情報である下位範疇化要素と, trunk 上にある親ノードの情報を, 組み上げる順番にスタックに保存した形で持つ. elementary tree を trunk に沿って anchor から再帰的に一段ずつ組み上げる際には, *Arg* 素性のスタックの先頭から要素を一つずつポップして葉ノードと単一化していくことになる. 従って葉ノードの種類毎に次節で説明するようなスキーマを定義する.

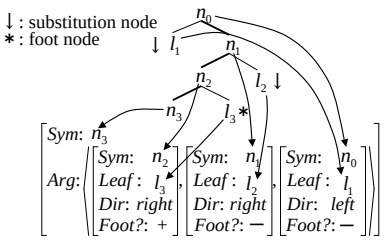


図 2: HPSG の語彙項目への変換

3.3 スキーマ

葉ノードの種類によって木構造の分岐を分類し, それぞれ 3.1 節で述べた 3 つの条件を満たすようにその関係をスキーマとして記述すると以下の 5 通りとなる.

- unary branch の親ノードを生成するスキーマ (スキーマ 1). すなわち TAG における unary branch を組み上げる.
- substitution node を下位範疇化するスキーマ (スキーマ 2, 3). すなわち substitution を模倣する.
- foot node を下位範疇化するスキーマ (スキーマ 4, 5). すなわち adjunction を模倣する.

以下では, これらのスキーマの詳細を説明する.

スキーマ 1

Arg 素性の先頭要素が unary branch を示している場合 (*Dir* 素性が *null* である場合) に適用される.

$$\begin{bmatrix} Sym : [1] \\ Arg : [2] \end{bmatrix}$$
$$\Big|$$
$$\left[Arg : \left\langle \begin{bmatrix} Sym : [1] \\ Leaf : null \\ Dir : null \end{bmatrix} \mid [2] \right\rangle \right]$$

スキーマ 2

左側の substitution node を下位範疇化する際に適用される. substitution node の *Arg* 素性が空になっていることにより, 3.1 節の条件 2 が満たされる.

$$\begin{bmatrix} Sym : [1] \\ Arg : [2] \end{bmatrix}$$

$$\begin{bmatrix} Sym : [3] \\ Arg : \langle\ \rangle \end{bmatrix} \qquad \left[Arg : \left\langle \begin{bmatrix} Sym : [1] \\ Leaf : [3] \\ Dir : left \\ Foot? : - \end{bmatrix} \mid [2] \right\rangle \right]$$

他の *initial tree* の根ノード　　*trunk*

スキーマ 3

右側の substitution node を下位範疇化する場合に適用される. スキーマ 2 の娘を左右入れ替えたの形になる.

スキーマ 4

左側のノードが foot node である場合に適用される. auxiliary tree が根ノードまで組み上げられた後 adjunction された木の組み上げをできるように親ノードの *Arg* 素性は trunk 上のノードの *Arg* 素性を foot node 側の *Arg* 素性の上に積み上げたものとなる. これにより, 3.1 節の条件 3 が満たされる. ただしここで $A \oplus B$ は A と B の *append* を表す.

図 3 はこのスキーマが適用される様子を示している. 図中で太線が adjunction された木 (木 1 とする) の trunk, 破線が adjunction した木 (木 2 とする) の trunk を示す. 図中 4 段目に対してスキーマ 4 が適用されている. 木 1 が根ノードまで組み上がるための情報が, adjunction されたノードの *Arg* 素性の値に入っている (図中 [6] 及び [2] の構造共有で表された素性構造). 木 2 を根ノードまで組み上げた後に, 木 1 の組み上げを再開できるように, adjunction したノードの *Arg* 素性の値を adjunction されたノードの *Arg* 素性の値にプッシュしたものを親ノードの *Arg* 素性の値としている (図中 3 段目の $\oplus$ によりこの操作が行われている).

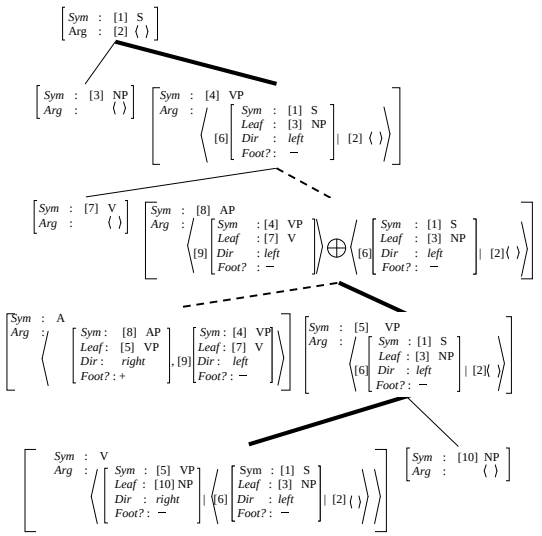
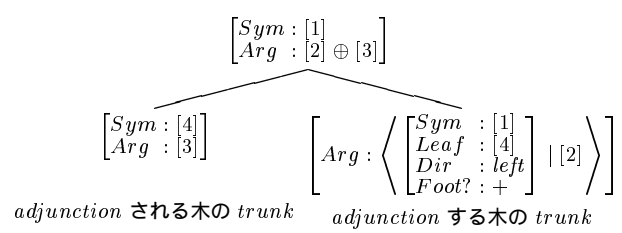


図 3: スキーマによる adjunction の模倣



adjunction される木の *trunk*　　*adjunction* する木の *trunk*

スキーマ 5

右側のノードが foot node である場合に適用される. スキーマ 4 の子ノードを左右入れかえた形になる.

以上のスキーマを適用して, 根ノードまで組み上げられたときの素性構造が以下の形であるとき 3.1 節の条件 1 が満たされる.

$$\begin{bmatrix} Sym & : & S \\ Arg & : & \langle\ \rangle \end{bmatrix}$$

3.4 素性構造への拡張

FB-LTAG は elementary tree の各ノードに素性構造を付与したものである. 素性構造は親ノードとの素性の構造共有の情報を記述した部分 (*top* と呼ぶ) と子ノードとの素性の構造共有の情報を記述した部分 (*bottom* と呼ぶ) に分かれる. 上で述べた手法を FB-LTAG に拡張するには, *Sym* 素性の非終端記号の代りに素性構造を記述し, スキーマの中で top と bottom を単一化することで実現する.

4 実装

本アルゴリズムを素性構造記述言語 LiLFeS [11] で実装し, 小規模な LTAG 文法及び XTAG 英文法 [5] に対して適用した.

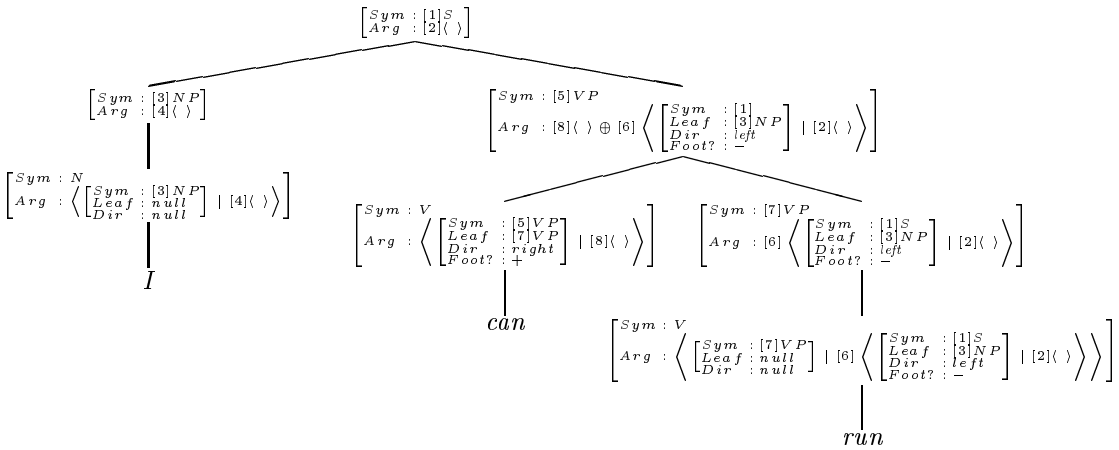


図 4: “I can run” に対する変換された文法による構文木

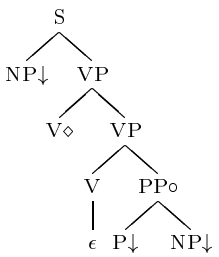


図 5: 変換されない elementary tree

小規模な LTAG 文法の変換　図 1 の elementary tree からなる小規模な LTAG 英文法を変換した．得られた HPSG-like な文法による “*I can run*” の構文解析結果は図 4 のようになり, 図 1 の LTAG における構文木と等価な構文木が生成されている.

XTAG 英文法の変換　さらに XTAG 英文法 [5] を HPSG-like な英文法に変換した．anchor が 1 つの elementary tree 438 個のうち, 293 個を変換することができた．変換されなかった elementary tree は It による強調構文や特定の前置詞句を補語に取る動詞句などに対応する．例えば図 5 は “John ventured into the cave.” という文における “ventured” に対する elementary tree である．図 5 の ◦ で示されたノードのように trunk 上になく, かつ葉ノードでもないノードが存在するため変換されない．これらの木への対処は今後の課題となっている.

5　まとめと今後の課題

本稿のアルゴリズムにより, 3.1 節で加えられた仮定を満たす elementary tree からなる LTAG 文法を等価な HPSG-like な文法に変換できる．同様に FB-LTAG 文法についての変換も本稿のアルゴリズムの拡張として得ることができる.

今後の課題としては, 本アルゴリズムで変換できなかった木への対処, 及び Sag の HPSG に近い文法への変換があげられる．さらに高速 HPSG パーザを XTAG 英文法から変換された文法に適用して XTAG 英文法の処理速度を向上させることも試みたい.

参考文献

[1] Yves Schabes, Anne Abeille, and Aravind K. Joshi. Parsing strategies with ‘lexicalized’ grammars : Application to tree adjoining grammar. In *Proc. ACL87*, pages 578–583, 1987.

[2] Carl J. Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

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

[4] Yuka Tateisi, Kentaro Torisawa, Yusuke Miyao, and Jun’ichi Tsujii. Translating the XTAG english grammar to HPSG, 1998. in Proc. TAG+4 workshop.

[5] The XTAG Research Group. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS Research Report 98-18, IRCS, University of Pennsylvania, 1998.

[6] Minoru Yoshida, Takashi Ninomiya, Kentaro Torisawa, Takaki Makino, and Jun’ichi Tsujii. Efficient FB-LTAG parser and its parallelization. In *PACLING 99*, pages 90–103, 1999.

[7] Yusuke Miyao, Takaki Makino, Kentaro Torisawa, and Jun’ichi Tsujii. The LiLFeS abstract machine and its evaluation with the LinGO grammar. *to appear in journal of Natural Language Engineering Special Issue*, 2000.

[8] Kenji Nishida, Kentaro Torisawa, and Jun’ichi Tsujii. Efficient HPSG parsing algorithm with array unification. In *NLPRS 99*, pages 144–149, 1999.

[9] Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun’ichi Tsujii. An HPSG parser with CFG filtering. *to appear in journal of Natural Language Engineering Special Issue*, 2000.

[10] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree adjunct grammars. *Computer and System Science*, 10:136–163, 1975.

[11] Takaki Makino, Minoru Yoshida, Kentaro Torisawa, and Jun’ichi Tsujii. LiLFeS — towards a practical HPSG parser. In *Proc. COLING–ACL ’98*, pages 807–811, 1998.