# HPSG から有限状態オートマタへ

鳥澤 健太郎　　辻井 潤一
東京大学大学院 理学系研究科 情報科学専攻
{torisawa,tsujii}@is.s.u-tokyo.ac.jp

**要旨**　本論文では、主辞駆動句構造文法 (Head-driven Phrase Structure Grammar;HPSG[3, 4]) を用いた高速なパーズのための一技法を提案する。我々の技法では、恐らくもっとも洗練された文法の枠組の一つである HPSG を、もっともプリミティブな文法の枠組である有限状態オートマタにコンパイルする。実験を通し劇的な効率の改善が見られたことを示す。

## 1　はじめに

従来、HPSG は理論的には美しいが、現実的でないという評価を得てきた。おそらく、その様な現状をもたらした要因は次の三点に集約されるであろう。

1. 単一化にかかる高いコストのため高速なパーザーが実現できなかった。
2. lexicalize されていることは、例外的な言語現象を書く上で利点であるとされる一方で、全ての単語に精緻な辞書項目をあたえることは実際的ではなかった。
3. いわゆる「文法的でない文」を扱えるのかどうかが不明であった。

本論文のテーマは、1 に対して解答を示すことにあるが、我々の最終的な目標は上の三点全てを解決することをふくむ。2 に関しては実験に関する報告の中で触れることとなろう。3 に関しては、計算の枠組としての HPSG は一般の logic program に比せる程強力であり、他の単一化文法と同じことができないとする根拠はないと主張するにとどめる。

我々のパーザー高速化における主眼は、「パーズ時に可能な限り単一化を行なわない。」のひとことに尽きる。これを実現するため、A) Off-line Raising、B) Dependency Analysis 及び C) Partial Unification の 3 つのテクニックを開発した。

## 2　パージングアルゴリズムの概要

HPSG においては sign(構文木のノード) は単一化により生成されるが、sign の素性構造のある部分は辞書項目 (sign の一つ) の素性構造からコピーされている、あるいは辞書項目の素性構造と共有されている。このような素性構造をある辞書項目から raise された素性構造とよぶ。

off-line raising は、sign 中の素性構造で辞書項目から raise されているもの、例えば head feature をパーズに先立ち計算してしまうテクニックである。このようにして計算された sign あるいはその素性構造の部分を、親子の関係で結ぶと有限状態オートマタができる。これにより、入力文にたいする可能な構文木を高速に数え上げることができる。また、off-line raising により生成された素性構造のことを **core structure** と呼ぶ。

off-line raising で計算される sign は、一つの辞書項目から raise される素性構造しか含んでいない。つまり、ある sign の子 sign 同士の間で生じる構造共有を介した相互作用は無視されている。したがって、パーズ時にそれらの相互作用の結果生じる素性構造を計算する必要がある。そのような動的に求めなければならない素性構造を **sub structure** と呼ぶ。sub structure を効率良くされるために開発されたのが、Dependency Analysis と Partial Unification である。

パージングは 2 フェーズに分けて行なわれる。

**Phase 1** off-line raising の結果生成された有限状態オートマタを使って構文木を組み上げる。

**Phase 2** Phase 1 で生成された構文木に対して sub structure を計算する。

Phase 1 では、core structure 間の単一化可能性チェックのみで処理がすすみ、Phase 2 で計算される sub structure は、「本物」の sign と core structure との間の差分であり単一化は最小限に押えられる。

## 3　Off-line raising と有限状態オートマタ

### 3.1　sign の数え上げ

例えば、例文 "John hit Mary." をパーズすることを考える。(図 1参照)。素性構造に逐一プリンシプルなどをボトムアップに適用するナイーブなアルゴリズムによりパーズが行なわれたと考えると、phrasal-sign $S_1, S_2$ はパーズ時に単一化によってそのつど生成されることになる。

しかしながら、$S_1, S_2$ を注意深く見てみると素性構造の大部分は何ら変更が加わることなく主辞, つまり "hit" の辞書項目から raise されていることが分かる。また、パーズの経過に大きな影響をおよぼす **subcat** や **head**

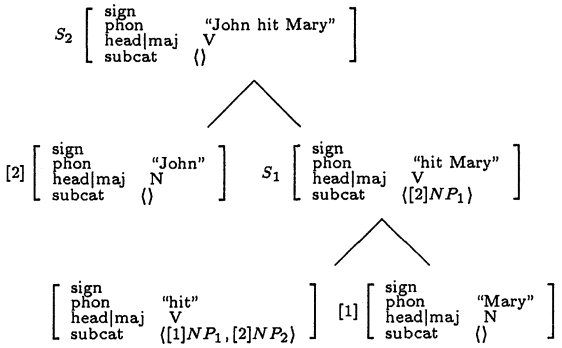


図 1: John hit Mary

などは主辞から raise されていることに注意せよ。このことから次のように推論できよう。

- 主辞の素性構造が決まれば、可能なパーズはかなり限定される。

我々が開発した off-line raising はこのような推論にもとづき、パージングを高速化するテクニックである。具体的には、

- パーズに先立ち、一つの辞書項目からそれを語彙的主辞とする可能な sign を全て数え上げる。
- 数え上げられた sign を状態、それらの間の親子の関係を遷移弧とみなすオートマトンを構成する。

ここで問題になるのは、1) 「通常複数の子 sign から親 sign が生成されるのに、一つの子から可能な親を数え上げるなどということが可能か？」 2) 「オートマトンが有限状態になるのか。」である。

まず、1) に関してであるが、ここでプリンシプル, ID-スキーマなどの sign 間の制約を表現している素性構造の内の一つを $R$ で表し、子 sign を $D_0, \cdots, D_n$, 親を $M$ で表現するとナイーブなアルゴリズムで子から親を作る操作は次のように表される。

$$M = \begin{bmatrix} \texttt{sign} \\ \texttt{head-dtr} & D_0 \\ \texttt{non-head-dtr-1} & D_1 \\ \cdots \\ \texttt{non-head-dtr-n} & D_n \end{bmatrix} \sqcup R \qquad (1)$$

ここで、off-line raising での $M$ に対応する core structure $Core(M)$ の生成を、

$$Core(M) = \begin{bmatrix} \texttt{sign} \\ \texttt{head-dtr} & D_0 \\ \texttt{non-head-dtr-1} & \bot \\ \cdots \\ \texttt{non-head-dtr-n} & \bot \end{bmatrix} \sqcup R \qquad (2)$$

とすると、単一化が成功し $M$ が生成される時には、必ず $Core(M)$ も生成される。これは素性構造間の順序関係である subsumption $\sqsubseteq$ に関する単一化の単調性と、任意の素性構造 $F$ に対し、$[\ \bot\ ] \sqsubseteq F$ であることから

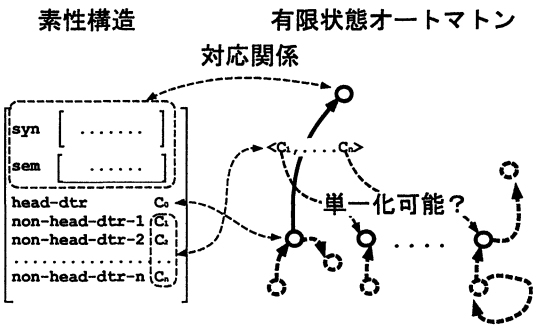


図 2: 素性構造と有限状態オートマトン

示せる。また、次の式を $Core(M)$ に再帰的に適用することにより、可能な sign が数え上げられる。

$$Core(M') = \begin{bmatrix} \texttt{sign} \\ \texttt{head-dtr} & Core(M) \\ \texttt{non-head-dtr-1} & \bot \\ \cdots \\ \texttt{non-head-dtr-n} & \bot \end{bmatrix} \sqcup R \qquad (2')$$

また、任意の $M, Core(M)$ に対し次が成立する。

$$M = Core(M) \sqcup \begin{bmatrix} \texttt{sign} \\ \texttt{head-dtr} & D_0 \\ \texttt{non-head-dtr-1} & D_1 \\ \cdots \\ \texttt{non-head-dtr-n} & D_n \end{bmatrix} \qquad (3)$$

これにより、$Core(M)$ から $M$ がパーズ時に「復元」できることになる。

次に 2) の問題に関してであるが、一般に、上で示した方法で off-line raising を行なった場合、可能な sign が無限に生成されることはありうる。この問題は、i) 同値な素性構造をもつ sign はオートマトンの同一の状態とみなし、ii) restriction[5] と呼ばれるテクニックにより素性構造の無限の variation を避けることをおこなうことによって解決できる。(詳細は、[7] 参照)

## 3.2 Phase 1

off-line raising により生成された有限状態オートマトンは実際のパージングでの Phase 1 において使用されることになる。先に off-line raising 時に生成された sign 間の主辞と親の間の関係がオートマトンの遷移弧になると述べた。この遷移弧の入力は、主辞でない子 sign に相当する素性構造 (core structure 中での `non-head-dtr-`$i$ の値) と単一化可能な sign あるいは他のオートマトンの状態である。(図 2参照) 特に、図 1での $S_1, S_2$ のように、subcat principle を使って生成された親子関係の場合は、主辞でない子 sign は `subcat` の値と構造共有 (structure sharing) されている。

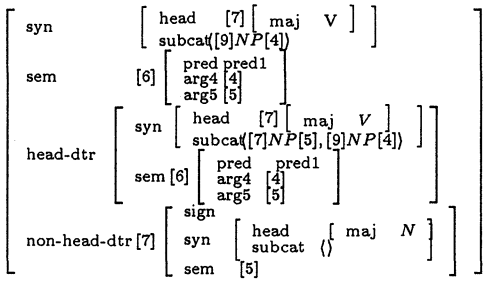


図 3: $Core(M)$

この遷移弧の入力に関する条件を、$\langle C_1, \cdots, C_n \rangle$ と表す。ただし、$C_i$ は $Core(M)$ 中の属性 (attribute) **non-head-dtr-$i$** の値である素性構造である。

Phase 1 parsing が chart 法に準拠したアルゴリズムによって行なわれると仮定する。任意の edge $e$ には、ある有限状態オートマトンの状態 $s$ が割り当てられており、これを $s = state(e)$ と表すことにする。

edge $e_0, e_1, \cdots, e_n$ が与えられたとして、それらの親に相当する新たな edge で、$e_0$ を主辞とするようなもの $e_m$ を生成する操作は、

1. $e_0$ の状態 $state(e_0)$ から出る遷移弧で遷移先が $s$, 入力条件が $\langle C_1, \cdots, C_n \rangle$ であるものを探す。
2. 次の二つの条件が成立したときに、$s = state(e_m)$ となるような $e_m$ を生成し、chart に書き込む。
   - chart 上の $e_m$ と同じ vertex 間に $s = state(e')$ となるような edge $e'$ がない。かつ
   - 任意の $i (1 \leq i \leq n)$ で $C_i$ が $state(e_i)$ と単一化可能である。

ここで注意すべきは、1) 単一化可能性のテストしか行なっていない。2) 冗長な edge の検出が、CFG の場合のように単純な identity check $state(e_1) = state(e_2)$ によって行なわれていることである。従来のナイーブなアルゴリズムでは、これらの操作が大量のメモリを消費する単一化と、素性構造の走査を必要とする subsumption check によって行なわれていたので、以上の手続きにより Phase 1 parsing はかなり高速になる。

しかしながら、先に述べたように Phase 1 parsing で用いられる有限状態オートマトンは、元の文法の近似にしかなっていない。その結果として、Phase 1 パージングは過剰生成 (overgeneration) を起こすが、生成された冗長な構文木は次の節で説明する Phase 2 で除去されることになり、やはりパーズの正しさは保証される。

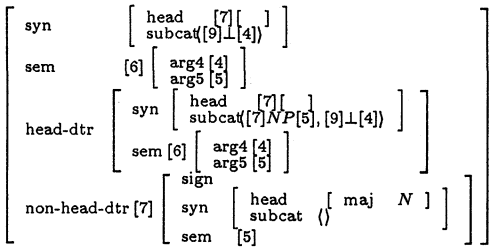


図 4: $dep(Core(M))$

## 4 Dependency Analysis と Partial Unification

本節では、Phase 2 で用いられているテクニックについて述べる。前にも述べたように Phase 1 parsing で使われるオートマトンは、兄弟 sign 間の構造共有を解した相互作用を無視しているので、「近似」に過ぎない。[1]

Phase 2 ではこれらの近似された部分を「復元」する。これを行なうためのもっとも単純な方法は、前節での (3) 式を用いる方法であるが、これでは単一化のコストがかかりすぎてそれほどの高速化は期待できない。ここでは、「本物」の sign と core structure との間の差分を core structure とはなるべく独立に計算することにより効率を上げるテクニックについて述べる。この差分を sub structure と呼び、任意の sign $S$ の sub structure は $Sub(S)$ と表すことにする。

dependency analysis は、$Core(M)$ から冗長なものを取り除く操作である。より具体的には、構造共有により $Core(M)$ の **syn/sem** に子 sign から raise されているものを取り除く。(ただし、構造共有自体はさらなる raise に備えて、残される。) これにより単一化ルーチンに入力に与えられる素性構造が小さくなり、単一化のコストが減る。例えば、図 3 にあるのは、$Core(M)$ の例であるが、これに dependency analysis を行なった結果である $dep(Core(M))$ は図 4 にある。head feature あるいは semantic feature のある部分が取り除かれたことがわかる。これを用いて、sign $M$ の sub structure $Sub(M)$ を子 sign $D_0, D_1, \cdots, D_n$ から求めるには、次を用いる。

$$Sub(M) = p\_unify(dep(Core(M)), \begin{bmatrix} \textbf{sign} & \\ \textbf{head-dtr} & Sub(D_0) \\ \textbf{non-head-dtr} & Core(D_1) \sqcup Sub(D_1) \\ \cdots & \\ \textbf{non-head-dtr} & Core(D_n) \sqcup Sub(D_n) \end{bmatrix})$$

ここで、$p\_unify$ は partial unification とよばれる特殊な単一化である。$p\_unify$ は、第一引数の素性構造

[1] off-line raising で restriction を用いた結果生じた「近似」も以下の方法で処理できる。

| Parsing Algorithm | Type of sentences (# of sentences) | Avg Length (word) | Avg Time (sec) |
|---|---|---|---|
| Phase 1 only | all (70) | 19.2 | 1.25 (1.12) |
| Phase 1 & Phase 2 | all (70) | 19.2 | 3.00 (1.65) |
| Phase 1 & Phase 2 | only successful (43) | 18.8 | 3.37 (1.84) |
| Phase 1 & naive application of rule schemata | only successful (38) | 17.13 | 55.09 (9.27) |
| Phase 1 & naive application of rule schemata | only successful (5) | 31.4 | 1093.22 (82.12) |

図 5: 実験

の部分の内、第二引数の対応する部分との単一化により、型, 属性値が変更されたもののみを値として返す関数である。つまり、第一引数の instantiate された部分のみが値として返された素性構造に現れる。(上の式で $p\_unify$ の結果に現れない $dep(Core(M))$ の部分は既に $Core(M)$ に現れていることに注意せよ。)

これにより、素性構造を子 sign から複数回 raise することなく、sub structure が計算できる。

## 5 実験

本研究で提案した技法の有効性を示すため、新聞記事から取られた７０文をパーズすることを試みた。我々が実験で使用した文法は、通常の記法ではプリンシプル、ID- スキーマなどの役割を果たすルール・スキーマが５個、辞書項目が 44 個、それら辞書項目が割り当てられていない単語に対して与えられるデフォールト辞書項目 (default lexical entry templates) が５５個からなる。

デフォールト辞書項目は、ある品詞の単語の一般的な振舞いを記述した辞書項目であり、JUMAN の出力結果である品詞に従い各単語に割り当てられる。これにより、全ての単語に対し精緻な辞書項目を与えずともパーズが可能になる。また、パーズの結果、単一化によりデフォールト辞書項目に新たな素性構造が付け加えられるが、これを利用してコーパスからの知識獲得を行なうこともできる。

実験は Common Lisp Object System 上に実現されたパーザーを用い、SS20(128Mb RAM) 上で行なわれた。パーザーは入力された７０文のうち、４３文に対し、構文木を生成した。

実験では本論文で提案された技法を全て用いた場合と、Phase 1 と通常の単一化を用いたナイーブなルールスキーマの適用を行なった場合を比較した。文毎によりスピードアップの度合がことなるので、単純な比較は意味を意味をなさないが、平均で数十倍の高速化が示されている。

## 6 結論

本論文では高速な HPSG でのパージングアルゴリズムを提案し、実際に新聞記事をパーズすることによりその有効性を示した。また本論文では論理型プログラミング言語である Definite Clause Program[1] でプリンシプルの機能を補強した場合については述べていないが、実装されたパーザーではそのような拡張がなされており、その正当性も示されている。(詳細は [7] 参照。)

もとより我々の研究の最終的なゴールは日本語のための頑健かつ言語学的に妥当なパーザーおよび文法を提供することである。この実現のため、我々の研究室ではコーパスからの知識獲得の様々な技法の研究 [6, 2] を行なっているが、本研究で提案した技法、および実験に使用した文法は実際にそれらの研究において現在使用されている。

## 参考文献


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

[2] Keiko Horiguchi, Kentaro Torisawa, and Jun'ichi Tsujii. Automatic acquisition of content words using an HPSG-based parser. In *NLPRS'95*, 1995.

[3] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics Vol.1.* CSLI lecture notes no.13, 1987.

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

[5] Stuart C. Shieber. Using restriction to extend parsing algorithms for complex feature based formalisms. In *Proceedings 23rd Annual Meeting of the Association for Computational Linguistics*, pages 142–152, 1985.

[6] Kentaro Torisawa and Jun'ichi Tsujii. Compiling HPSG-style grammar to object-oriented language. In *NLPRS'95*, 1995.

[7] Kentaro Torisawa and Jun'ichi Tsujii. Computing phrasal-signs in HPSG prior to parsing, 1996. to be submitted.