Vol.2015-NL-221 No.5
Vol.2015-SLP-106 No.5
2015/5/25

並列構造解析に向けた依存構造解析アルゴリズムの拡張

吉本 暁文[1,a)]　新保　仁[1,b)]　原 一夫[2,c)]　松本 裕治[1,d)]

概要：一般的に依存構造解析のアルゴリズムでは，句構造を扱わないために並列構造を考慮することが難しい．そこで，依存構造解析のための Eisner アルゴリズムを，並列構造解析ができるように拡張した．その規則の導出木は，既存の依存構造のアノテーションから導出することができる．

1. はじめに

構文解析において扱いが難しい構造の一つに並列構造が挙げられる．並列構造とは，例えば英語の場合 “and” や “or” のような等位接続詞によって単語や句が列挙される構造を指す．並列構造の扱いの難しさは複数あり，主に以下のものが挙げられる．

(1) 列挙される単語や句の範囲同定
(2) 複数の句が並列されているか否かの判定
(3) 並列構造のアノテーション

これらのうち，(1) と (2) は構文解析器で解くことの困難さであるのに対し，(3) は構文解析器で解きやすい問題を設定することの困難さである．

他方，係り受け解析では，近距離にある単語間での係り受けに比べて，遠距離の係り受けの精度が低くなってしまうことが知られており，並列構造のように比較的長い単語列と遠距離の係り受けを生じやすい状況では通常の係り受けより解析が困難になると考えられる．

並列構造のうち，頻出する種類の一つに名詞並列がある．名詞並列では，意味やカテゴリに共通点のある事柄が列挙されると考えられ，列挙されたそれぞれの項目間で類似度をはかることで，それらの項目が並列されていることを判定できる． 黒橋らの手法 [1] では日本語解析において列挙された句の構造的な類似度を系列アライメントを用いて捉える手法を提案しており，日本語係り受け解析器 KNP [*1] [2], [3] の処理の一部として組み込まれている．原ら [4] は，英語解析において，並列句の文字列上の類似度を系列アライメントに機械学習を組み合わせて高精度に捉えるとともに，単純な文法規則を導入し並列句の階層構造を同定する手法を提案している．

構文解析に系列アライメントによる並列構造解析を組み込む手法は，日本語のものでは黒橋らの手法が挙げられるが，英語では Hanamoto らの手法 [5] が挙げられる．Hanamoto らの手法は英語の構文解析器 Enju [*2] [6] と原らの並列句同定手法を双対分解を用いて組み合わせたもので，並列構造を含む文で解析精度が向上したことが示されている．

しかし通常の係り受け解析では句を扱わないため，係り受け解析で直接的に句の構造的な類似度を考慮することはできなかった．また，系列アライメントに基づく並列句同定手法でも，並列句の構造や種類によって異なる関数を導入することはできていない．例えば，名詞並列ならば並列項目同士の品詞やカテゴリの共通点が有効な素性となるが，動詞句や文の並列となると，品詞や単語から表層的に得られる系列の類似度が有効でないことも多い．

そこで本稿では，依存構造解析でも並列構造における句の構造を考慮することができるアルゴリズムとして，Eisner アルゴリズム [7] をベースに拡張した手法を提案する．このアルゴリズムを用いることによって，例えば並列項目の Head や部分木などの構造に応じて系列アライメントだけでなく，他の類似度を考慮する手法も使うことができるようになると期待される．

統計的依存構造解析器の学習には，正解の依存構造を用意する必要があるが，英語の依存構造データとしては一般的に Penn Treebank (PTB) [8] の句構造でアノテーションされたデータを機械的に変換したものが用いられている．

句構造から依存構造への変換は一意に定まるものではなく，並列構造を含め，様々な依存構造への変換手法が考え

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[2] 国立遺伝学研究所
National Institute of Genetics
[a)] akifumi-y@is.naist.jp
[b)] shimbo@is.naist.jp
[c)] kahara@nig.ac.jp
[d)] matsu@is.naist.jp

[*1] http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP

[*2] http://www.nactem.ac.uk/enju/index.ja.html

© 2015 Information Processing Society of Japan

られているが，本稿では一般的に利用されている変換手法を利用する．

2. 並列構造に対応する依存構造

PTB のアノテーションを句構造から依存構造に変換する手法は複数あるが，本稿では Stanford parser [*3] [9] で用いられている変換手法（以下，Stanford アノテーション）を用いる．

Stanford アノテーションの依存構造にはラベルが付与されており，その種類や役割は Stanford typed dependencies manual [10] に記されている．

これらのラベルの中で，並列構造がある場合しか使われ方が定義されていないものに *cc*, *conj* がある．また，並列構造がある場合にしか使われないとは限らないが，並列構造がある場合に頻出するものに *punct* がある．以下に [10] を参考に *cc*, *conj* のラベルの使われ方を記す．

***cc*: coordination**

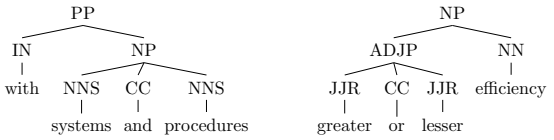
並列構造の一つの項目の中の単語と，その並列構造をなす等位接続詞との間の関係を指す．この項目には，通常，並列構造の中で最初に現れる項目が選ばれ，並列構造の Head となる．等位接続詞は文頭に現れる場合もあるが，これも *cc* と呼び，文の述語の子となる．（この後者の *cc* については，本稿のアルゴリズムでは一般の依存構造として扱う）

***conj*: conjunct**

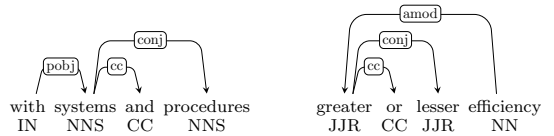
“and” や “or” などの等位接続詞によってつなげられた 2 つの要素の間の関係を指す．並列項目は非対称に扱うこととし，並列構造の Head は最初の並列項目であり，他の並列項目はその Head に対する *conj* 関係の子になる．

2.1 英語の代表的な並列構造

英語の並列構造で最も基本的なものに，A & B の形が挙げられる．PTB においてこの構造が出現する例を以下に示す．



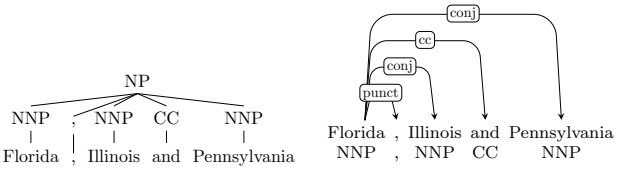
これらを Stanford アノテーションの依存構造に変換すると，それぞれ以下の構造に変換される．



[*3] http://nlp.stanford.edu/software/lex-parser.shtml

並列構造の範囲のアノテーションは一貫して，並列される最初の項目が Head になり，等位接続詞はラベル *cc* で最初の項目の子となり，他方の項目はラベル *conj* で同じく最初の項目の子となる．このように，同じ並列構造に属する項目はその並列構造の最初の項目を親として共有する．

3 つ以上の項目が並列される場合の最も基本的なものには A, B & C の形が挙げられる．PTB における例を以下に示す．



この例のように，3 つ以上の項目が並列される場合であっても，Stanford アノテーションでは同じ並列構造に属する項目や *punct*, *cc* は一貫して最初の項目の子となる．これは A, B, and C のように *punct* と *cc* が連続する場合や A, B and, he says, C のように挿入句が並列構造の内部に含まれる場合も同様である．この挿入句も，並列構造の最初の項目の子となる．

3. 既存の依存構造解析アルゴリズム

主要な依存構造解析アルゴリズムには Transition-based の手法や Graph-based の手法がある [11]．Transition-based の手法は基本的には線形時間で解析できる特徴があるが，一般に大域最適解を求めることができない．これに対し，Graph-based の手法はモデルの複雑さに応じて計算量が増加するが，大域最適解を求めることができる．本稿では，依存構造解析の過程で並列関係を捉えることに焦点を当てるため，Graph-based の手法を用いることとする．

並列構造解析では並列関係にある句の範囲の同定が重要なため，Graph-based の手法の中でも Span-based の手法に着目し，その中で代表的な手法である Eisner アルゴリズムを出発点とする．この手法は注目する部分単語列の範囲に Span を連結してより大きな Span を作りながらその範囲の最適な部分木を見つけていくボトムアップの解析法で，CKY 法 [12] をベースとしている．

3.1 Eisner アルゴリズム

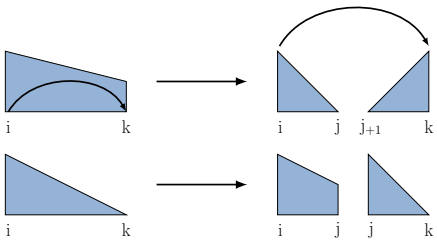
Eisner アルゴリズムを拡張した様々な解析アルゴリズムの元となり，そして本稿でもベースとしている Eisner アルゴリズムは，単語の範囲を示す Span をボトムアップに組み上げて最終的に文全体の Span を計算する．

1st order の Eisner アルゴリズムでは，Span は 2 種類用意される．一つは complete span と呼ばれるもので，Head とその右側だけまたは左側だけの子を持ち，かつそれ以上その右側または左側に子を持たないことを表す部分木を示す．もう一つは incomplete span と呼ばれるもので，Head

© 2015 Information Processing Society of Japan

の位置の異なる 2 つの complete span を，片方の Head がもう片方の Head の子となるように係り受けを作る．この Span も Head とその右側だけまたは左側だけの子を持つが，そのさらに右側または左側に同じ側に部分木を足す complete span が後から結合でき，complete span になれることを示す．

以下の図で三角形で示されるのが complete span であり，台形で示されるのが incomplete span である．Head から見た部分木が右側か左側かで，三角形や台形の右下がりと左下がりを区別している．

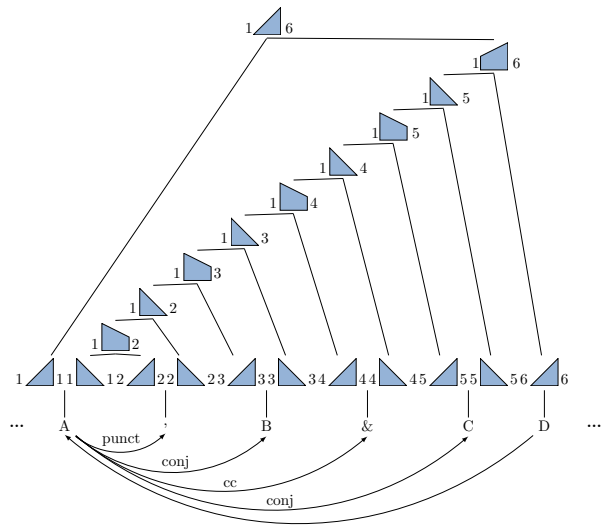


図の i, j, k で表される添字は文中の何番目の単語かを示すインデックスであり，j_{+1} のように表すときは j の直接右隣であることを意味する．通常，文の最も左側に，右下がりの三角形のノードとしてルートノードを配置し，それを含む全体で右下がりの三角形になる Span を作り，文全体の最終 Span とする．

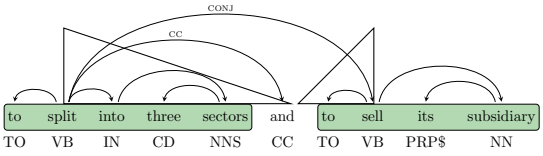
このアルゴリズムでは Span の組み合わせを考慮する際に i, j, k それぞれ単語数 n に比例する探索が必要となり，時間計算量は $O(n^3)$ となる．

4. 並列構造解析アルゴリズム

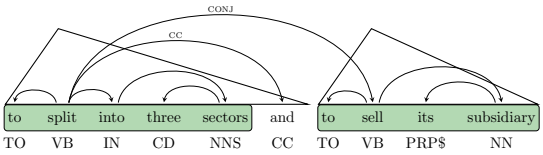
1st order の Eisner アルゴリズムで Stanford アノテーションによる並列構造の導出を示すと以下のようになる．



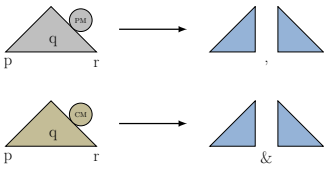
A, B や A & B の構造があるときの A と B の係り受けを作るときの Span の組み合わせは，いずれも A の Head から右側の部分木と B の Head から左側の部分木の組み合わせとなる．この係り受けを作る際，並列構造のそれぞれの項目の並列らしさ，例えば単語列の系列的な類似度を考慮したくても，それに対応する Span と係り受けがそのままでは対応しない問題がある．



そこで並列構造の範囲だけ Span の種類を変えて，Head の両側の部分木をまとめて一つの Span とすることを考える．



並列項目を区切る文字の出現を表す *punct* や *cc* については，左側の部分木と右側の部分木を分けた場合でも，左側の部分木がこれらの並列項目の子になった直後に必ず右側の部分木を合わせることになる．このため，これらの *punct* や *cc* についても同様に両側の部分木をまとめて一つの Span とすることを考える．これらの Span をそれぞれ PunctMark, CCMark と定義する．



ここで，p, q, r で表される添字は文中の何番目の単語かを示すインデックスである点は i, j, k のときと同じ扱いだが，探索数としては並列項目の区切り文字にしか依存せず，文中の全単語数と比べて非常に少ないと考えられるものであることを示す．時間計算量を考えるとき，これらに依存する計算量は文中の区切り文字の個数 m を用いて表す．

並列構造に関係する部分木の組み合わせを考える際，その順序は依存構造に従う．Stanford アノテーションの場合 A , B & C ならば，以下の順番で組み合わされなければならない．

(1) P = A + ,
(2) Q = P + B
(3) R = Q + &
(4) S = R + C

このとき，A, B, C が全て対等に並べられているとすると，以下に挙げた項目のペアについて並列らしさを計算することが考えられる．

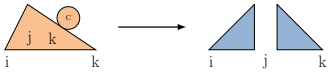
(a) A と B　(b) A と C　(c) B と C

© 2015 Information Processing Society of Japan

ここで，(b) や (c) は，Span の始点，終点，Head の位置が分かっていてもそのままでは A や B の範囲を得ることができない．前述の Q を作るとき，(a) の範囲がまとめられてしまうため，これらの範囲を得るためには A か B の範囲が分かるように追加の変数を用意する必要がある．

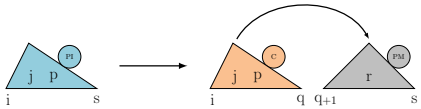
(b) と (c) はいずれかを実現できれば良いが，ここでは (b) を選ぶこととする．この場合，並列項目を区切る文字が現れたときに A の終点を Span 上の追加の変数に保存することで，後に Span が組み合わせられるときに A の範囲を復元することができ，(b) を実現できる．

この追加の変数を扱えるようにした，左右の部分木をもつ Span を以降 Conj と呼ぶ．Conj は，並列構造の項目を最初に捉える規則で，任意の単語列から作ることができると定義する．



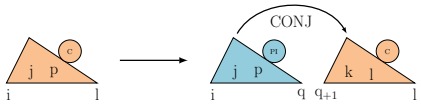
任意の単語列から Conj を作るときは，まだ A の範囲に相当するものがない．このように A の範囲を扱わない場合には，追加の変数は Span 終点の位置と一致させ，追加の変数がない Span と同様に扱うこととする．

Conj が並列構造の区切り文字に隣接したとき，そこで追加の変数を変更する規則を導入し，これを PunctIncomplete と定義する．



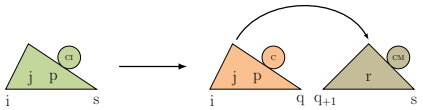
ここで，図の p が最初の並列項目の終点位置を記憶する変数を表している．最初に Conj が PunctMark と連結するとき，図の p と q は同じ位置となる．既に PunctMark と連結した範囲を Conj が含んでいるときには p と q は異なる位置となり，同じ p の位置を連結後の Span に保持させる．

PunctIncomplete は区切り文字が連結されているため，再び Conj と連結して並列項目同士を繋げることができる．このための規則を以下のように導入する．

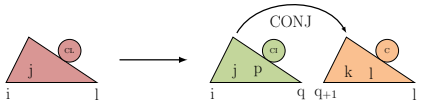


このとき，係り受けのラベルは conj にならなければならない．これは正解の係り受けアノテーションから得られるべき導出を曖昧性のないように得るための制約として用いることができる．

並列項目が最後の区切り記号として考えられる CCMark と連結されたとき，次の項目で並列構造を完成させるための規則を以下のように導入し，これを CCIncomplete と定義する．

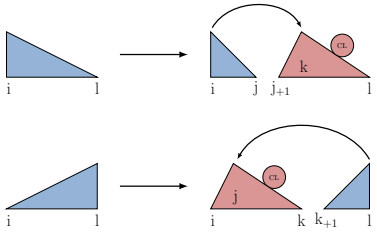


CCIncomplete が最後の並列項目と連結されたとき，元の Eisner 規則による部分木と連結させるための規則を以下のように導入し，これを Complete と定義する．



これ以上並列項目が対等な関係で連結しないため，Complete ではそれまで並列項目の範囲を記憶していた追加変数を保持しない．また，この場合も係り受けのラベルは conj にならなければならない．

最後に，元の Eisner 規則による部分木と連結して一般的な部分木に取り込むための規則を以下のように定義する．



この他，再帰的な並列構造や並列構造同士の係り受け，*punct* と *cc* が隣接する場合に対応した規則の一覧を付録 A.1 に載せている．

5. 並列構造規則のカバー率

PTB の WSJ パートを表 1 に示すように分割し，Stanford アノテーションに変換した．この依存構造アノテーションから，並列構造規則を用いて導出を得られるか確認したところ，表 2 に示す導出数となった．依存構造アノテーションは一文につき一通りだが，同じ依存構造に到達できる導出が複数ある場合，一文あたりの導出数が複数となる．

5.1 導出数が複数となる場合の内訳

付録 A.1 に示す規則のうち，(4) の規則は，PunctMark が使われた場合と使われなかった場合の 2 通りの導出を許す曖昧性の原因となる．実際，(4) の規則を除くと，wsj 22 の場合で一意の導出が 1656，導出なしが 44 となり，同じ依存構造に到達できる複数の導出が存在するような曖昧性が生じない．導出なしとなる事例数が増えるのは，3 並列以上かつ “, and” が出現するような場合に導出できなくなるからである．

5.2 導出なしの内訳

導出なしとなる事例のうち，wsj 22 についての内訳を表 3 に示す．*cc* が並列構造の Head から見て左側の部分木としてアノテーションされる例が最も多い．詳しいアノテー

© 2015 Information Processing Society of Japan

表 1 Penn Treebank の分割と内訳

	wsj 2-21	wsj 22	wsj 23
文の数	39832	1700	2416
cc/*conj* タグを持つ文の数	17901	771	1035
cc タグの数	24004	1003	1384
conj タグの数	24367	1007	1348

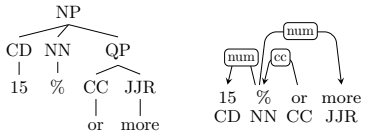
表 2 並列構造規則による導出数

文あたり導出数	wsj 2-21	wsj 22	wsj 23
1	35188	1480	2140
2	3220	165	186
4	307	23	23
8	8	0	0
導出なし	1109	32	76

表 3 導出なしとなる場合の内訳（wsj 22）

事例数	エラーの種類
13	*cc* が並列構造の Head の左側の子
7	複合語 *cc*
4	*conj* アノテーションの挿入句
3	交差
2	*conj* になるべきラベルが *conj* になっていない
1	*conj* になるべきでないラベルが *conj* になっている
1	*parataxis* アノテーションの挿入句
1	未登録 *cc*

ションの内訳はさらに細かくなるため省略するが，そのような事例の一つとして以下のような部分木を持つ事例がある．



左側が PTB で右側が変換後の依存構造であるが，本来 *cc* の関係は逆になるべきだと考えられる．PTB で並列構造にとって一般的でない句構造が存在した場合に，依存構造への自動変換に誤りが生じている可能性が考えられる．

次に多いのは複合語 *cc* で，wsj 22 では “rather than”, “along with”, “as well as”, “instead of” の 4 種類があった．本稿では CCMark に一単語のものだけを考慮したため，このような結果となったが，複数単語の間に別の単語が入る場合を除けば，一単語の場合と同様に対応することができると考えられる．

6. おわりに

本稿では依存構造解析と同時に並列構造の構造的な類似度を考慮するための第一歩として，依存構造解析で並列項目の句を考慮するためのアルゴリズムを提案した．

本稿のアルゴリズムにより，依存構造の解析中に注目している句の部分木構造や，より簡単には Head の種類に応じた様々な並列構造らしさのスコアを計算するために利用することができると考えられる．

さらに，句の種類に応じて並列構造に関するスコア関数の重みを学習し，これまでのように名詞句だけでなく動詞句やより複雑な構造のときにはそれらに適した関数が重視されるような解析器を作ることもできると考えられる．

様々な種類の並列構造を捉えるために系列アライメントだけでなく，単語や句の分散表現やそれらを利用した演算も取り入れることができると期待している．

今後，実際に複数種の並列構造らしさのスコアを導入し，多様な並列構造に対して柔軟に解析できる解析器の実現に取り組む予定である．

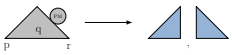
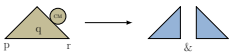
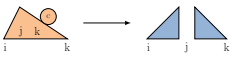
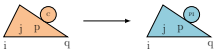
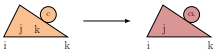
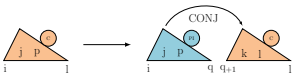
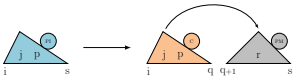
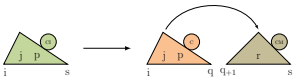
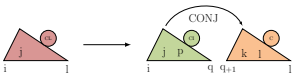
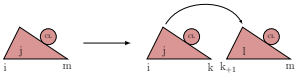
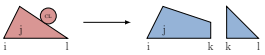
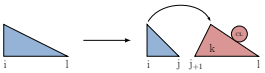
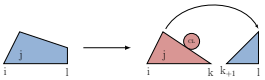
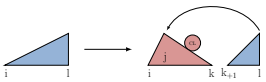
参考文献

[1] Kurohashi, S. and Nagao, M.: A syntactic analysis method of long Japanese sentences based on the detection of conjunctive structures, *Computational Linguistics*, Vol. 20, No. 4, pp. 507–534 (1994).
[2] 黒橋禎夫：日本語構文解析システム KNP version 2.0 b6 使用説明書，京都大学大学院情報学研究科，Vol. 6 (1998).
[3] 黒橋禎夫：結構やるな, KNP，情報処理，Vol. 41, No. 11, pp. 1215–1220 (2000).
[4] Hara, K., Shimbo, M., Okuma, H. and Matsumoto, Y.: Coordinate structure analysis with global structural constraints and alignment-based local features, *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP (ACL-IJCNLP '09)*, Singapore, pp. 967–975 (2009).
[5] Hanamoto, A., Matsuzaki, T. and Tsujii, J.: Coordination structure analysis using dual decomposition, *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, pp. 430–438 (2012).
[6] Miyao, Y. and Tsujii, J.: Deep linguistic analysis for the accurate identification of predicate-argument relations, *Proceedings of the 20th international conference on Computational Linguistics*, Association for Computational Linguistics, p. 1392 (2004).
[7] Eisner, J. M.: Three new probabilistic models for dependency parsing: An exploration, *Proceedings of the 16th conference on Computational linguistics-Volume 1*, Association for Computational Linguistics, pp. 340–345 (1996).
[8] Marcus, M. P., Marcinkiewicz, M. A. and Santorini, B.: Building a large annotated corpus of English: The Penn Treebank, *Computational linguistics*, Vol. 19, No. 2, pp. 313–330 (1993).
[9] Klein, D. and Manning, C. D.: Fast exact inference with a factored model for natural language parsing, *Advances in neural information processing systems*, pp. 3–10 (2002).
[10] De Marneffe, M.-C. and Manning, C. D.: Stanford typed dependencies manual, *URL http://nlp. stanford. edu/software/dependencies manual. pdf* (2008).
[11] McDonald, R. T. and Nivre, J.: Characterizing the Errors of Data-Driven Dependency Parsing Models., *EMNLP-CoNLL*, pp. 122–131 (2007).
[12] Younger, D. H.: Recognition and parsing of context-free languages in time n^3, *Information and control*, Vol. 10, No. 2, pp. 189–208 (1967).

© 2015 Information Processing Society of Japan

付　　録

A.1　並列構造規則のリスト

(1)	$O(m)$	PunctMark			
(2)	$O(m)$	CCMark			
(3)	$O(n^3)$	Conj			Make general conjunct
(4)	$O(m^2n^2)$	Conj			Case (A ,) and B
(5)	$O(n^3)$	Conj			Case (A & B) and C
(6)	$O(m^2n^4)$	Conj		係り受け素性 j → k 並列素性 (i-p)(q_{+1}-l)	Make ((A ,) B)
(7)	$O(m^2n^2)$	PunctIncomplete		係り受け素性 j → r	Make (A ,)
(8)	$O(m^2n^2)$	CCIncomplete		係り受け素性 j → r	Make (A &)
(9)	$O(m^2n^4)$	Complete		係り受け素性 j → k 並列素性 (i-p)(q_{+1}-l)	Make ((A &) B)
(10)	$O(n^5)$	Complete		係り受け素性 j → l	Make ((V_1 & V_2)(N_1 & N_2))
(11)	$O(n^4)$	Complete			
(12)	$O(n^4)$	RightComplete		係り受け素性 i → k	
(13)	$O(n^4)$	RightIncomplete		係り受け素性 j → l	
(14)	$O(n^4)$	LeftComplete		係り受け素性 l → j	

© 2015 Information Processing Society of Japan