2003－NL－155 (4)
2003／5／26

実用的な文法を開発するためのデバッグツール

薬師寺 あかね[†] 建 石 由 佳[†,††] 宮 尾 祐 介[†]
吉 永 直 樹[†] 辻 井 潤 一[†,††]

自然言語処理の分野において、汎用かつカヴァレッジの広い文法を情報抽出などの実用に適用したいという要求がある。しかし汎用かつカヴァレッジの広い文法を人手で開発することは難しい。我々は、正解コーパスの参照と構文解析誤りの蓄積とによって、文法を効率的に改良することを助けるツール *willex* を開発した。*willex* は構文解析結果を正解コーパスと比較し、誤った部分結果を自動的にあるいは手動で取り除くことで結果の曖昧性を減らす。また文法の欠陥部分をデータとして蓄積することで、文法開発者が統計的に文法の欠陥を明らかにすることを助ける。実際の適用例として、HPSG 文法に *willex* を適用し明らかになった文法の欠陥を挙げる。

Debug Tool for Practical Grammar Development

Akane Yakushiji,[†] Yuka Tateisi,[†,††] Yusuke Miyao,[†]
Naoki Yoshinaga[†] and Jun'ichi Tsujii[†,††]

In the field of natural language processing, there have been requirements for the application of general-purpose and wide-coverage grammars to practical information extraction. However, general-purpose and wide-coverage grammars are hard to develop manually. We have developed a tool, *willex*, that helps grammar developers to work efficiently, by consulting annotated corpora and recording parsing errors. First, *willex* decreases ambiguity of the parsing results by comparing them to an annotated corpus and removing wrong partial results automatically and manually. Second, *willex* accumulates parsing errors as data for the developers to clarify the defects of the grammar statistically. We applied *willex* to an HPSG grammar for an example.

1. はじめに

本稿では、我々の開発したデバッグツール *willex* について述べる。

情報抽出などの実用的なアプリケーションに構文解析器を使いたいという要求が、自然言語分野においてある。例えば、HPSG 文法の１つである XHPSG[1] に基づいた構文解析器を用いた、生医学論文からの項構造の抽出という研究[2] がある。ここで、構文解析器に必要なある程度確立された汎用な大規模文法は例えば XHPSG などがすでに存在しているが、しかしカヴァレッジが低いという問題を持っている。

カヴァレッジが低い原因としては、文法に不足があるということが考えられる。例えば XHPSG の場合、動詞の並列構造が扱えない（ex. “Molybdate **slowed but did not prevent** the conversion.”）、reduced relative が扱えない（ex. “Rb mutants **derived** from patients with retinoblastoma.”）といった不足が挙げられる。さらに、欠陥を見つけて文法を改良するには、人間の労力が多く必要になる。

そこで今回、汎用文法の改良を支援するためのツールとして以下のような機能を持つツール *willex* を開発した。

- 構文情報がタグ付けされたコーパスに含まれている言語直感を取り入れることで、文法を改良する人間の労力を削減する。
- 文法の欠陥のデータを収集しまとめることで、コーパス全体を視野に入れたデバッグを提供し、デバッグ経験ログを保存する。それにより見通しの良いデバッグを行うことができる。

† 東京大学大学院情報理工学系研究科コンピュータ科学専攻
Department of Computer Science, Graduate School of Information Science and Technology, the University of Tokyo
†† CREST, 科学技術振興事業団
CREST, Japan Science and Technology Corporation

2. 理想的な文法デバッグとは

既存の文法開発ツールには XTAG の grammar writer[3]、ALEP[4]、ConTroll[5]、宮田らによるツール[6]、[incr tsdb()][7] などがあるが、これらのツールの問題点としてはデバッグする人間の直感に頼っている、構文解析結果の曖昧性が大きいため文法欠陥を探すのが困難である、１つ１つの文法欠陥に対して１つ１つ探索・対処しているといったことがある。それに対し我々は、タグ付きコーパスの言語直感を取り入れ結果の曖昧性を減らす、また文法の欠陥を収集しまとめ全体を視野に入れたデバッグを行うという、より人間の労力が少なく見通しが良い手法を提案する。

既存の文法開発と今回提案する文法開発ではワークフローが次のように異なってくる。既存の文法開発では、文を１文ずつ見ていって文法の欠陥を探し、その上で見つけた文法の欠陥を１つずつローカルに直す。今回提案する文法開発では、タグで構造が付けられた文を１文ずつ見ていって文法の欠陥を探して記録し、その上で文法の欠陥全体をまとめる。それからまとまったデータを直す。こうすることで、コーパス中での出現頻度がより高い文法に関する欠陥から直すといったことや、構文解析の目的にとって重大な文法欠陥から直すということが可能になる。もちろん、既存の文法開発でもユーザ側で文法欠陥をまとめて直すこともあるが、今回提案する手法ではシステムの機能として文法の欠陥全体をまとめる。

3. *Willex* の機能

Willex の機能は以下のようなものがある。

(1) 語のまとまりと品詞・ラベルの情報を含んだ XML タグ付きコーパスを利用することで、結果の曖昧性（人が探す労力）を削減する。

(2) 文法欠陥の情報をファイルに出力することで、文法の欠陥データの収集と全体を視野に入れた解析を行う。また、行われたデバッグのログを保存する。

3.1 XML タグ付きコーパスの利用

XML タグ付きコーパスからは、語のまとまり情報と品詞・ラベル情報を利用する。

まず、語のまとまり情報を構文解析時に与えることにより文の曖昧性が減るため、構文解析時の曖昧性が減り解析が速くなり、結果の曖昧性も減る。具体例を示すと、タグ付きコーパスに

<su> I saw <np> a girl with a telescope </np></su>

のような文があった場合、パーザに対して

(I saw (a girl with a telescope))

という入力を *willex* が与える。ここでパーザの機能として、このような入力を受け取った場合には図 1 の上図のように “a girl with a telescope”をひとまとまりとしたような構文木のみが出力され、下図のような “with a telescope” が “saw”にかかる構文木は出力されないということを仮定する。

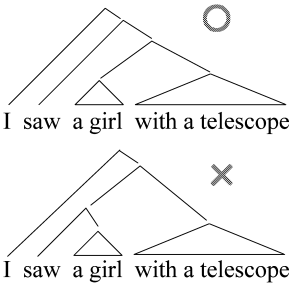

図 1 語のまとまりに沿った構文木の出力例

次に、*willex* は XML タグで表されている品詞・ラベルと構文解析結果を比較し、不適切な構文木を削除する。これにより、人間がチェックするべき構文解析の途中経過の部分構文木が減る。なお、部分構文木の削除は後から手動でも行える。具体例を示すと、図 2 に示したように NP タグで囲まれた “A cat”に対応する部分構文木はラベルが NP であるもののみが正解であり、他のものは削除される。

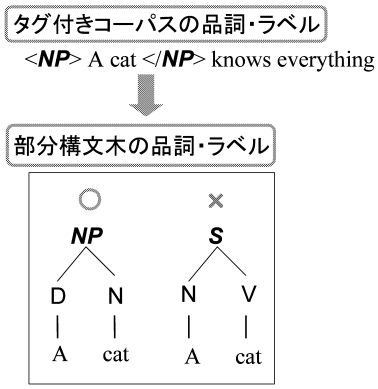

図 2 品詞・ラベルを用いた部分構文木の削除例

品詞・ラベルによる部分構文木の削除の仕組みを詳しく説明すると、以下のようになる。また例を図 3 に示す。

Step 1：タグと対応する部分構文木とで品詞・ラベルを比較し、異なる木を削除する。

Step 2；削除された木から組み上げられる木も、順にさかのぼって削除していく。

例外：タグと一致したノードから unary rule だけでたどれるノードは、削除しない☆。

3.2 文法欠陥情報の出力

また *willex* は、文法の欠陥情報をファイルに出力する機能を持つ。これにより、文法の欠陥データを収集し、統計的に文法欠陥を処理できるようになる。また、どのような文法欠陥が発見されたかというデバッグ経験ログを保存することができる。

具体的なファイル出力内容は、表 1 のようである。まず文番号と、文法欠陥が存在したために構文解析に失敗した

☆ unary rule だけでたどれるノードの全ては同じタグに対応することになるので、この例外処理が必要となる。

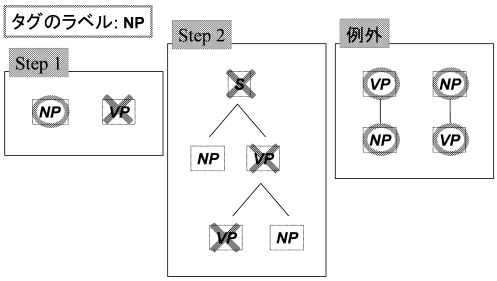

図 3 品詞・ラベルを用いた部分構文木の削除例（詳細）

個所の語の範囲を語番号で出力する。そして、ユーザが入力したコメントをそのまま出力する。

表 1 ファイル出力内容

文番号	語番号	ユーザが入力したコメント
0	3–12	動詞の並列構造
1	–	OK
2	4	辞書エントリなし

4. *Willex* の動作

willex の実装としては、素性構造を用いた文法のパーズ結果の表示ツールである *will* [8] の拡張とした。使用したプログラミング言語は JAVA である。

以下で、具体的な *willex* の動作の様子について述べる。

実際の動作においては部分構文木中のノードの親子関係を取得するのに時間がかかるため、不適切な部分構文木の削除までをオフラインで行って結果をファイルに保存し、その後ファイルから削除結果を取り出して処理の続きを行うというシステムを構築した。

図 4 がオフライン実行を指定するウィンドウである。ここでユーザは処理する XML タグ付きコーパスファイルと、括弧付き文（図中では `Brck. Sentence`）の出力先ファイル、部分構文木の削除結果を保存するファイルを指定する。

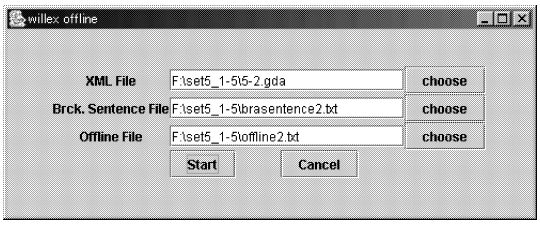

図 4 オフライン実行指定ウィンドウ

部分構文木の削除結果を保存するファイルの内容は、図 5 のようになる。ここで、括弧付き文の保存されているファイル（`Brck.file`）の名前も保存している。`S#: 0` は文番号 0 を表し、その次の行の番号列が削除されるべきノードに割り振られている番号列となっている。

図 6 がオフライン実行の結果を元にして文法欠陥を探していく作業を行う際の最初のウィンドウである。ここでユーザは部分構文木の削除結果ファイルと、見つかった文法欠陥の情報を出力するファイル、また文法欠陥の候補を挙げてあるファイルを指定する。

```
Brck.file:  F:\set5_1-5\brasentence2.txt
S#:  0
318 317 314 313 312 311 310 309 308 307 …
```

図 5 部分構文木の削除結果ファイル

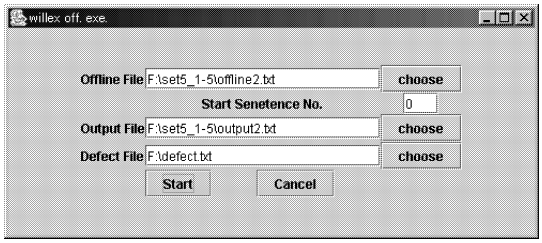

図 6 オンライン実行指定ウィンドウ

図 6 のウィンドウの Start ボタンを押すと、図 7 のウィンドウが現れる。ユーザはこのウィンドウでコーパス中の文を 1 文ずつ処理していく。Start ボタンを押すと、文がパーズされ☆オフライン処理結果に沿った部分構文木のみが CKY テーブルの形で表示される（図 8）。ある文を処理中に他の短い典型的な文をパーズするといったこともでき、元の文に復帰するには ReStart ボタンを押す。

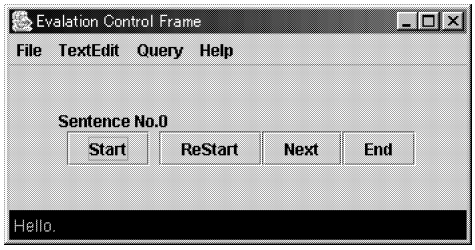

図 7 文処理ウィンドウ

CKY テーブルウィンドウ（図 8）では、左側に CKY テーブルがボタン（中の数字は対応するノードの数を示す）の山の形で、右側に左側のボタンのうち押されたものに対応する部分構文木が表示される。

ユーザが構文解析に失敗した個所を見つけたら、まず必要に応じて「そこまでは正解である」ような娘部分構文木あるいは語の範囲（これは CKY テーブルのボタンで表される）を指定する（図 9）。その上で、文法欠陥が存在したため構文解析に失敗した語の範囲に対応するボタンから、文法欠陥情報の出力ウィンドウ（図 10）を呼び出す。

図 10 のウィンドウには、文法欠陥が存在したため構文解析に失敗した個所の語の範囲と（指定されていれば）娘である語の範囲が表示されている。このウィンドウで、ユー

☆ 既にオフライン処理の際に一度パーズしているので二度手間になるが、パーズ結果の全てのデータを保存するのはデータ量的に負担が大きすぎると判断した。

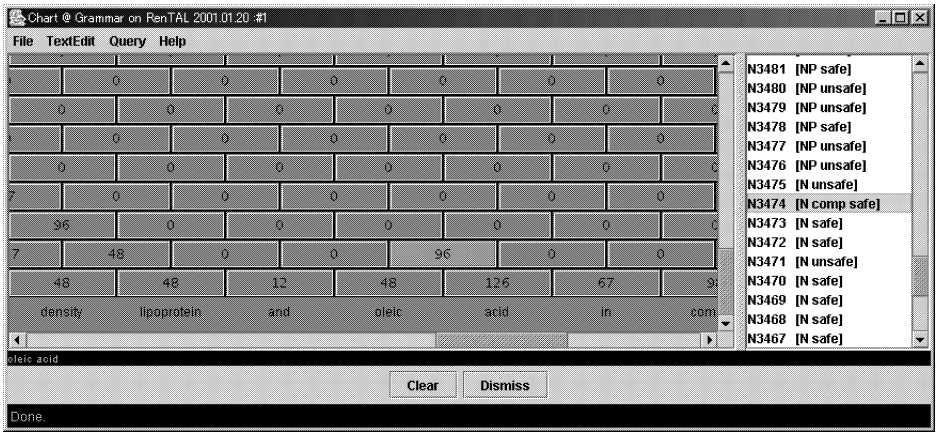

図 8 CKY テーブルウィンドウ

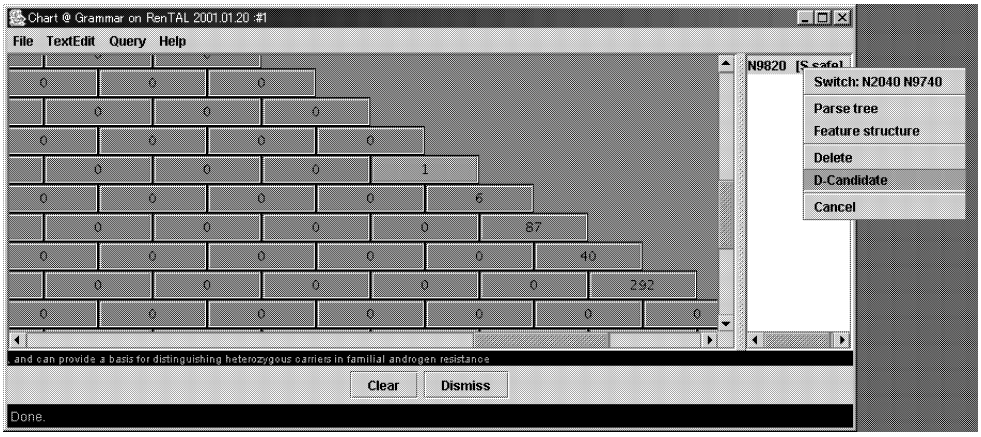

図 9 文法欠陥個所の娘部分構文木の指定

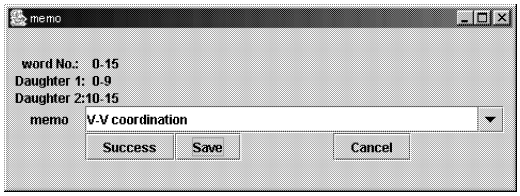

図 10 文法欠陥情報出力ウィンドウ

ザは既出の文法欠陥をプルダウンメニューから選ぶか、新しく入力するかで文法欠陥の種類を指定し、Save ボタンを押す。Success ボタンは、文の解析に成功し正解を得た場合のためのショートカットボタンである。

5. 適用例；rental-XTAG

5.1 設　　定

実際の *willex* の適用例として、XTAG 英文法 [9] を文法変換器 [10] で変換した大規模な HPSG 文法（rental-XTAG）に対して *willex* を適用した。用いたコーパスは、GDA-DTD[11] を修正した DTD☆に基づいたタグが付与された MEDLINE アブストラクトである。例を図 11 に挙げる。

☆ GDA には英語の前置詞句を直接指定するタグが存在しないため、新たに <prep>、<prepp> を追加した。

```
<su><v><np><n>The initial <n><n>infec-
tion </n><prepp>with human immunodeficiency
virus type 1 (HIV-1) </prepp></n></n>
<prepp>in most individuals </prepp></np>
<v>usually <v>results </v><prepp>in <np>
<n>the establishment </n><prepp> of a la-
tent or chronic infection </prepp></np>
</prepp></v></v><prepp>before <np><n>
eventual progression </n><prepp>toward ac-
quired immunodeficiency syndrome </prepp>
</np></prepp>.</su>
```

図 11 タグ付けされた MEDLINE アブストラクトの例

またコーパス中のタグ付き文は rental-XTAG に基づいた正解を指定しているわけではない（タグが仮定している文法が rental-XTAG と異なっている）ため、表 2 のような多対多の品詞・ラベル変換表を用意した。例えば、修正 GDA で <n> と <np> のいずれかのタグで囲まれている句に対応する rental-XTAG の構文木は、N と NP のいずれかのラベルが振られていれば正解とする。

26 アブストラクト中の 208 文（1 文当たりの語数 : 最大 68 語、平均 24.2 語）を調査した結果、73 文で構文解析に成功し正解を得た。よってカヴァレッジは 35.1%である。逆に、135 文になんらかの問題が含まれることになる。

表 2　品詞・ラベル変換表

修正 GDA	rental-XTAG
<n>, <np>	N, NP
<v>, <vp>	V, VP, S
<aj>, <ajp>	A, AP
<ad>, <adp>	Ad, AdP, P, PP, D, DP, Comp, S
<prep>, <prepp>	P, PP
<ij>	I
<su>	S

5.2　*willex* の定性的評価

実際に *willex* を使ったユーザから得られた評価としては、

- チェックする部分結果が減り見やすくなるため、部分結果を削除する機能は有効である。
- コーパスが品詞・ラベルを指定していない場合があるので、手動で部分結果を削除できる機能は有効である。
- 文法欠陥ログをファイルに取る機能は、一時的なメモを書き、後で熟考して書き直せるので便利である。

というものが得られた。しかし逆に否定的な評価としては、

- 文法欠陥のログを取るときに、構文解析が失敗した文中の場所を確定するのが面倒である。
- デバッグする文法を熟知した人間でないと、文法欠陥を厳密に判定できない☆。

というものが得られた。

これらの評価から、*willex* の主な機能は有効に活用されたと言える。しかしまだ人手に頼る部分も多く、より一層の自動化が求められていることが分かる。

5.3　*willex* の定量的評価

タグ付きコーパスを利用することによる部分構文木の数の減少を調べた結果が図 12 である。このグラフから、タグ付きコーパスを使用したことにより人間の手間が省けたと言える。

5.4　Rental-XTAG の欠陥

Willex の適用により見つかった rental-XTAG の欠陥を、表 3 にあげる。この表から、rental-XTAG の欠陥のうち重要なのは

(1)　lexical entry の不足（ex. “upstream” : 名詞、“productively” : 副詞）
(2)　reduced relative が扱えない
(3)　動詞の並列構造が扱えない

と思われる。

これらの欠陥を直すと、今回パーズした 208 文のうち 159 文（約 76%）がパーズ可能となることが予測される☆☆。

5.5　修正 GDA と rental-XTAG の衝突

修正 GDA が仮定する文法と rental-XTAG が異なっているために衝突が起こり、パーズに失敗することもあった。

☆ このため、実際に文法欠陥を調べる過程を次のような 2 段階に分けた。まず文法をある程度分かっている人間が荒くパーズエラーを分類し、それから文法を熟知した人間の推測を参考に文法欠陥を判定した。

☆☆ ここでは、後述する「修正 GDA が仮定する文法と rental-XTAG が異なっているための衝突」を無視している。

表 3　Rental-XTAG の欠陥

lexical entry の不足	62
reduced relative が扱えない	35
過去分詞　26 現在分詞　9	
動詞の並列構造が扱えない	22
後置修飾をする形容詞が扱えない	9
“, but not”が解析できない	4
目的の to 不定詞が扱えない	3
“, which ...”が NP を後置修飾できない	3
as 関係詞節の reduced 形が扱えない	2
“greater than”（“>”）が解析できない	2
その他	17

衝突の統計を表 4 にあげる。

表 4　修正 GDA と rental-XTAG の衝突

修正 GDA	rental-XTAG	出現数
形容詞句	動詞句	36
“,”以外の括弧付け		10
“,”の括弧付け		8
語の欠如		2
動詞句	名詞句	1
動詞句	前置詞句	1
形容詞句	関係節	1
名詞句	補文	1
名詞句	形容詞（名詞の省略）	1

具体例として、修正 GDA では形容詞句であるのに対し rental-XTAG では動詞句である例を図 13 に、括弧付けの衝突の例を図 14 にあげる。

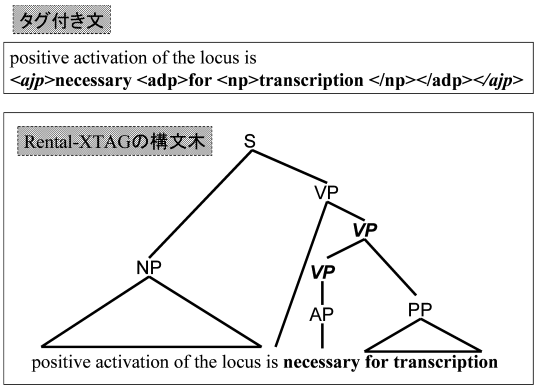

図 13　修正 GDA で形容詞句、rental-XTAG で動詞句の例

これらの衝突は、単純な品詞・ラベル変換表（前述した表 2）だけでは解消できない。解決法としては、不適切なタグを削除したりタグの位置を移動したりする前処理系を組み込むということが考えられる。

6.　まとめと今後の課題

我々は、XML タグ付きコーパスを利用し文法欠陥情報を出力するデバッグツール *willex* を開発した。XML タグ付きコーパスを利用することで、人間の作業の手間を削減することに成功した。また文法欠陥情報を記録することで、

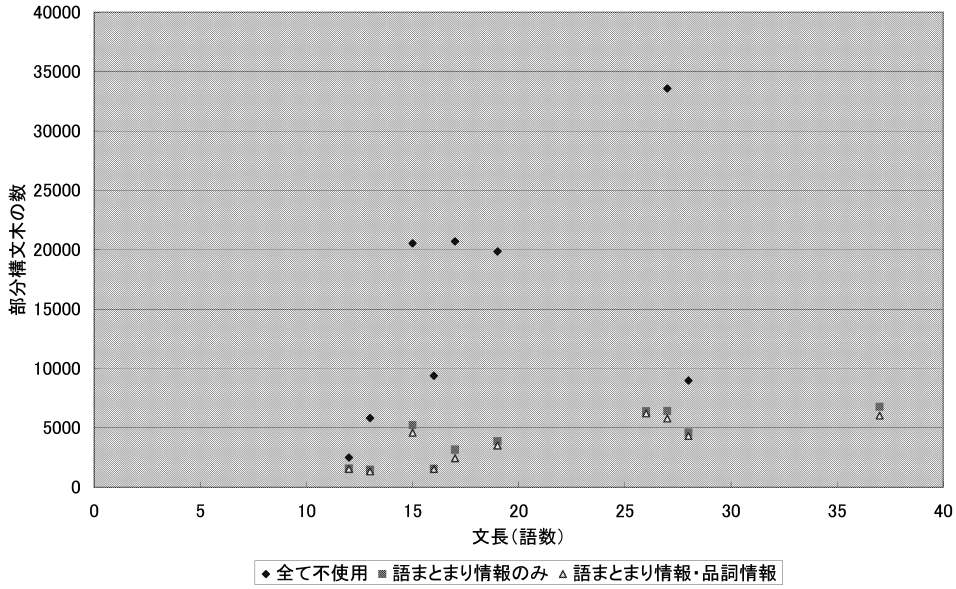

図 12 部分構文木の数の減少の様子

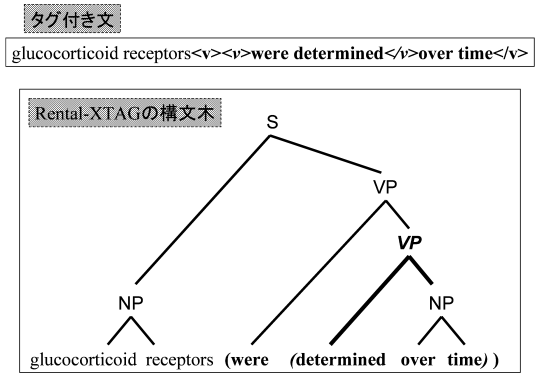

図 14 括弧付けの衝突の例

全体を視野に入れたデバッグ環境を提供した。

しかし、品詞・ラベル変換表だけではデバッグ対象の文法とタグが仮定する文法との不一致を吸収しきれていないという問題が残っている。

また今後は、構文エラー原因の（半）自動推測を行えるように *willex* を拡張したいと考えている。

参 考 文 献

1) Yuka Tateisi, Kentaro Torisawa, Yusuke Miyao, and Jun'ichi Tsujii. Translating the XTAG english grammar to HPSG. In *Proceedings of TAG+4 workshop*, 1998.

2) A. Yakushiji, Y. Tateisi, Y. Miyao, and J. Tsujii. Event extraction from biomedical papers using a full parser. In *Pacific Symposium on Biocomputing 2001*, pages 408–419, January 2001.

3) Patrick Paroubek, Yves Schabes, and Aravind K. Joshi. XTAG – a graphical workbench for developing Tree-Adjoining grammars. In *Proc. of the 3rd Conference on Applied Natural Language Processing*, pages 216–223, 1992.

4) Paul Schmidt, Axel Theofilidis, Sibylle Rieder, and Thierry Declerck. Lean formalisms, linguistic theory, and applications. Grammar development in ALEP. In *Proc. Coling '96*, volume 1, pages 286–291, 1996.

5) Thilo Goetz and Walt Detmar Meurers. The ConTroll system as large grammar development platform. In *Proc. of Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, pages 38–45, 1997.

6) 宮田高志, 高岡一馬, and 松本裕治. 単一化文法用 GUI デバッガの実装. In **情報処理学会研究報告** *NL-129*, pages 87–94, January 1999.

7) Stephan Oepen, Emily M. Bender, Uli Callmeier, Dan Flickinger, and Melanie Siegel. Parallel distributed grammar engineering for practical applications. In *Proc. of the Workshop on Grammar Engineering and Evaluation*, pages 15–21, 2002.

8) 今井久夫, 宮尾祐介, and 辻井潤一. HPSG パーザーの為の GUI. In **情報処理学会研究報告** *NL-127*, pages 173–178, September 1998.

9) The XTAG Research Group. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS Research Report 01-03, IRCS, University of Pennsylvania, 2001. available in `http://www.cis.upenn.edu/~xtag/`.

10) Naoki Yoshinaga and Yusuke Miyao. Grammar conversion from LTAG to HPSG. In *Proc. of the sixth ESSLLI Student Session*, pages 309–324, 2001.

11) HASIDA Koiti. Global document annotation (GDA). available in `http://www.i-content.org/GDA/`.