自 然 言 語 処 理 146－11
（２００１．１１．２１）

素性に基づく文法のための辞書記述ツール

宮田 高志
科学技術振興事業団 CREST
miyata.t@carc.aist.go.jp

大谷 朗
奈良先端科学技術大学院大学/大阪学院大学
akira-o@is.aist-nara.ac.jp

素性に基づく文法の開発環境の一機能として、辞書記述を支援するためのシステムを試作した。本システムの特徴は、実際の解析で使う複雑な素性構造をそのままユーザにみせるのではなく、データベースのビュー機能を使って文法との整合性を保ちつつ柔軟に構造を変更できる点にある。システムは「素性・型定義エディタ」「素性構造エディタ」「辞書エディタ」の三つのモジュールからなり、素性の多重継承や適切性のチェック・語彙項目の編集・XML 形式などの各種フォーマット出力などの機能を持つ。

[キーワード]　型付き素性構造、単一化文法、辞書記述、グラフィカルユーザインターフェース

A Lexical Description Tool for Feature-Based Grammar

Miyata Takashi[†] and Ohtani Akira[‡*]
CREST, Japan Science and Technology Corporation[†]
Nara Institute of Science and Technology[‡] / Osaka Gakuin University[*]

A lexical description tool for feature-based grammar has been constructed as part of feature-based grammar development environment system. The main characteristics of this tool is a view function which is commonly used in database field. A view function simplifies a complex feature structure and allows user to modify the structure maintaining the consistency of its grammar. The tool consists of three modules – feature-type definition editor, feature structure editor, and lexicon editor. They manage checking multiple inheritance and appropriateness in feature-type definition, editing of lexical entries, and formatting output in LaTeX, PostScript, and XML.

[Keyword]
Typed Feature Structure, Unification Grammar, Lexical Description, Graphical User Interface

1　はじめに

近年、自然言語処理の分野において、解析を中心に統計的手法を用いたアルゴリズムが盛んに研究されており、多くの成果をあげている。統計的手法を適用するには大量の学習データが必要となるため、そこで使われる情報としては表層的なものにならざるを得ない。一方で表層的な情報の統計だけでは正しく解析できない文が存在することは事実であり、単純な統計的手法では解析精度だけを考えても限界がある。実際、最近の統語解析に関する研究では、解析精度 80%後半から 90%前後にかけて 1%程度の精度向上を競うという状況を呈しており ([4, 3, 6])、現在の方法論では飛躍的な進歩の望みは薄いと思われる。

現在のところ、統計的手法を用いる解析システムは木構造を出力するのみであるが、応用において本当に必要なのは文の命題内容や発語内行為のタイプといった「意味」である。Web からの情報検索・情報抽出などをみると、木構造さえ不要で形態素解析や固有名詞の認識といったいわゆる浅い解析だけで十分ではないか、という反論もあるかもしれない。しかしこれはむしろ話が逆で、意味解析の技術が成熟・普及していないから浅い解析しか行なわれていない

とみるべきである。木構造から「意味」構造への変換は必ずしも自明ではないが、統語解析ほど盛んに研究されているとはいい難い。

以上の二つの点から、我々は Head-driven Phrase Structure Grammar (HPSG) に基づく統語解析、とくに日本語文法に関して研究を続けている。HPSG では語や句が担う統語的な情報と意味的な情報が一つの型付き素性構造 (typed feature structure) というデータ構造で表現され、解析は二種類の情報を同時に処理することで進行する。どのような意味論を採用するかは別として、適当な意味論に基づいて各単語の意味記述を与え、統語構造に沿った意味構造の構成規則を与えれば、解析終了時には統語構造と同時に意味構造も得られる。

この「統語的な情報と意味的な情報が同時に処理される」という特徴は、逆に HPSG に基づく文法開発を難しくしている。「普遍的な」意味論というのは少なくとも現時点では存在しないので、当面は応用分野ごとに (少しずつ) 異なる意味論を採用することになると思われるが、一度作った文法の意味的な部分だけを変更するというのは手作業では困難である。これは、統語構造における局所的な枝分かれが意味構造の構成規則と対応するように配慮しながら二種類の構造を決めなければならないからである。本研究では、そのような場合の作業負担を軽減するためのシステムを提案する。我々は過去の研究で、[12, 8] を改良したシステムを使って日本語 HPSG の開発およびその拡張を行なってきたが ([13, 14, 9, 15])、その時の経験から文法開発において次のような支援の必要を感じた。

1. 型付き素性構造の定義とそのインスタンスである辞書の自動的な一貫性保持
2. 着目部分だけを表示するビュー機能と出力形式の柔軟な制御
3. テストセットコーパスの策定と自動検査

1. は、例えば「syn 素性の下にあった subcat 素性を syn|val の下に移動したい」といった時に、定義の変更と同時に辞書も自動的に変更してほしいということである。さらにこのような変更が単なる位置の変更にすぎないのか、解析に影響を及ぼすような「本質的な」変更なのかについてもある程度チェックすることが望ましい。

2. は、句が担っている素性構造が一画面に収まらないほど大きな場合、離れた位置に表示されている二つの素性の値を見比べるのが困難であるという問題である。現在のシステムにも素性を畳み込む機能はあるのだが、二つの素性がそれぞれ別の深い埋め込みの中にあるような時は畳み込むだけでは不十分であり見づらい。素性構造そのものは変更せずに、見たい部分だけを簡単に並べて表示するような機能が必要である。

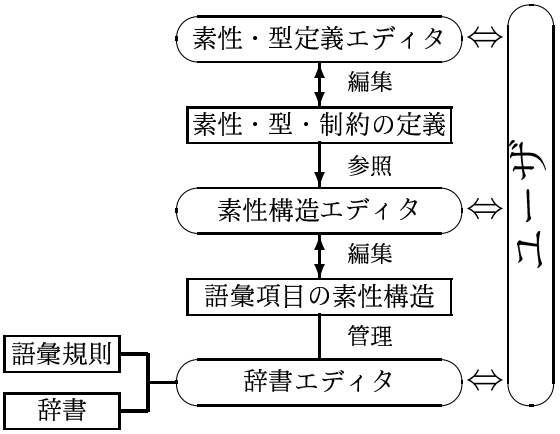

図 1: システム構成

3. は、辞書や文法の変更が本当に改良だったのか、バグを作り込んでいないか、などをチェックするためのテストデータを決め、そのチェックを自動化したいということである。これはソフトウェア開発全般に言われることであるが [1, 5]、本論文では扱わない。

2 システムの概要

本システムは図 1 の三つのモジュールからなる。これらのモジュールは次のような機能をもつ。

- 素性・型定義エディタ
  型階層を中心に、各型が持つ素性を表示・編集する。素性の多重継承や適切性のチェック [2] なども扱う。
- 素性構造エディタ
  「素性・型定義エディタ」で作成した定義に従って、素性構造のインスタンスを表示・編集する。ポップアップメニューによる型の選択やドラッグアンドドロップ/カットアンドペーストによる部分構造間の単一化、指定した素性のズームイン/ズームアウト、ビューによる表示の制御などの機能を持つ。
- 辞書エディタ
  語彙項目の一覧表示・検索・複製を行なう。「素

性構造エディタ」と連携して各語彙項目を編集することもできる。また、語彙規則を選択的に適用した結果を閲覧することもできる。

また、機能としては次の三つの特徴を持つ。

- 素性・型定義における適切性検査機能
- 素性構造表示における「ビュー」機能
- 各種出力フォーマットのサポート (XML, LaTeX, PostScript)

以下の節ではこれらの項目に関して説明する。

2.1 各モジュールにおけるユーザインターフェース

2.1.1 素性・型定義エディタ

図 2 に素性・型定義エディタの GUI ウィンドウを示す。図 2 左のウィンドウでは、ウィンドウ左半分の木構造が型の階層を表している[1]。マウスで型を選択すると、その型が持つ素性が右上のリストボックスに表示される。リストボックス中では親から継承した素性とその型固有の素性が色で区別して表示される。右下のリストボックスは「語もしくは句が担うべき全ての情報」を列挙しており、現在選択している型で表現している情報・その先祖で表現している情報が色で区別して表示される。また、左の木構造の上で型名をドラッグアンドドロップまたはカットアンドペーストすることで、型の間の階層を変更することができる。さらに右上もしくは右下のリストボックス中の素性名や情報名を左の型名へドラッグアンドドロップまたはカットアンドペーストすることで、それらの出現可能な位置を変更することができる。

図 2 右のウィンドウでは同じ定義が Attribute-Value Matrix (AVM) 形式で表示されている。各素性の型はポップアップメニューを使って切替えることができ、型を切替えるとその型で許容される素性が (やはり継承したかどうかを色で区別して) 表示される。さらにマウスで素性名を他の素性名へドラッグアンドドロップまたはカットアンドペーストすることで、その素性が出現可能となる位置を変更することができる。また、「語もしくは句が担うべき全ての情報」のリストボックスから各項目をドラッグアンドドロップまたはカットアンドペーストすることで、

[1] 多重継承も扱うので実際には束 (lattice) になるが、多重継承が起こっている型は、その親のうちで最初に出現した型との関係だけを表示することで木構造に埋め込んで表示している。

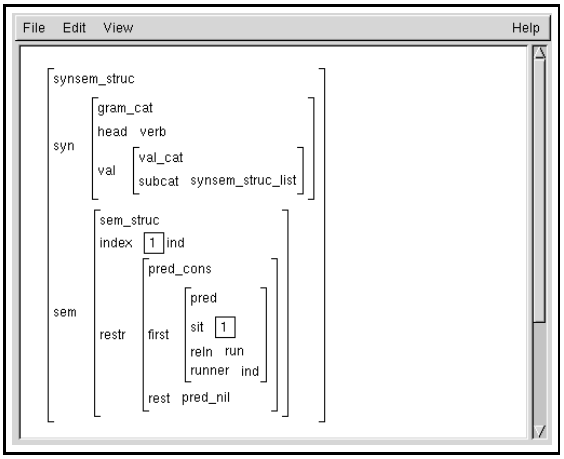

図 3: 素性構造エディタ

それらの情報がどの素性に格納されているのかを定義することができる。さらに、素性を指定して一時的にその値だけをウィンドウ全体で表示するズームインも可能である。

素性・型の定義に対して木構造と AVM 形式の二通りの表示形式を提供しているが、それぞれ一長一短がある。型の間の階層関係を木構造で表示する表示形式は素性の継承関係を確認するのに便利である。例えば pos 型の下位型として noun, verb などの型を設けた場合、「品詞全般がもつべき素性は pos 型で定義し、名詞が固有にもつ素性は noun 型で定義する」といったことを行なう (あるいは確認する) ためには木構造表示が優れている。一方、具体的な語または句においてどのような素性が出現し得るのかをみるには AVM 形式がよい。

2.1.2 素性構造エディタ

図 3 は素性構造エディタの GUI ウィンドウである。これは素性・型定義エディタの AVM 形式とほとんど同じだが、ドラッグアンドドロップおよびカットアンドペースト操作の意味付けが異なる。すなわち、素性構造エディタでは素性名を他の素性名へドラッグアンドドロップまたはカットアンドペーストすると単一化 (unification) が試みられ、成功すれば構造共有が起こる。その他の機能は AVM 形式と同じで、型名をクリックすると出てくるポップアップメニューで型を変更できる点や、ズームイン/ズームアウトの機能は共通である。さらに 2.2.2 節で説明するビューをメニューから選択して適用することで、埋め込みの深い所にある「語もしくは句が担うべき情報」を簡潔に表示させることができる。

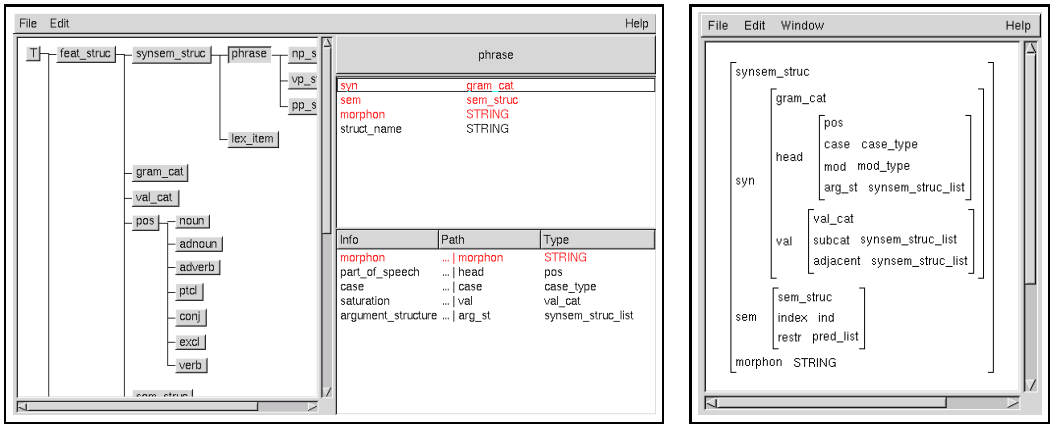

図 2: 素性・型定義エディタ

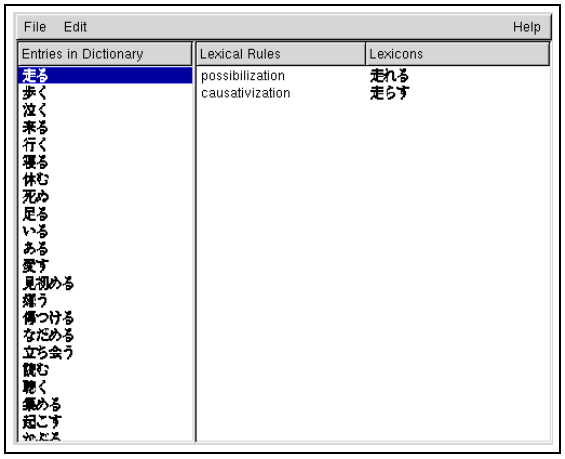

図 4: 辞書エディタ

2.1.3 辞書エディタ

図 4 は辞書エディタの GUI ウィンドウである。ウィンドウ左のリストボックスには適当な条件、例えば “syn|head|pos=verb” などで検索した辞書中の語彙項目が並べられる。ウィンドウ右のリストボックスは二つのカラムをもつが、左側のカラムには語彙規則が並べられており、それぞれの語彙規則を適用した結果が右のカラムに表示される。左のリストボックスもしくは右カラムの語彙項目をダブルクリックすると素性構造エディタが起動され、具体的な素性構造を閲覧したり編集したりすることができる[2]。

[2]システムを簡単にするため、語彙規則を適用した結果の語彙項目に対する編集は辞書には反映されない。

2.2 各機能の特徴

2.2.1 適切性検査

任意の型に任意の素性を配置することを許してしまうと、単一化の結果、常に型が一意に決定できることを保証できなくなってしまう。[2] では単一化の結果の型が常に一意に決定できるための条件として、素性の定義に対する次のような制限を挙げている。

1. 全ての素性 f について、$\mathrm{Approp}(f,\sigma)$ が定義されるような型 σ のうち最も一般的なものが一意に定まる。
2. $\mathrm{Approp}(f,\sigma)$ が定義されていて、τ が σ より特殊ならば、$\mathrm{Approp}(f,\tau)$ も定義されてかつ $\mathrm{Approp}(f,\sigma)$ より特殊である。

ここで $\mathrm{Approp}(f,\sigma)$ とは、型 σ における素性 f の値で、最も一般的な型を表す。型 σ が素性 f を持たない時は未定義とする。これらから、素性を多重継承する時は二つ以上の祖先たちが同じ素性名を持っていてはいけないこと、継承した素性のとり得る型の再定義はより特殊な型への変更しか許されないことなどが導かれる。

2.2.2 「ビュー」機能

一般に文法が大きくなればなるほど、語や句が担う素性構造は複雑で巨大なものになってゆく。しかし、品詞や格フレームなど多くの情報は単独ではそれほど複雑な構造はもっていない。語や句が担う素性構造が複雑にみえるのは、いろいろな場合に備えて用意した素性を一度に閲覧するからである。例えば接続助詞は主辞としては動詞句を受けるので subcat 素性を持ち、付加語としても動詞句を修飾するので

mod 素性を持つ。さらに二つの動詞句が表す意味構造を何らかの関係で結びつけるので、sem 素性は空でない内容をもつ。もちろんこれら三つの素性の間ではいくつかの素性を構造共有しており、全体として接続助詞の機能を表現しているのだが、常に全体をみなければ機能がわからないわけではない。

そこでまず、語もしくは句が担うべき全ての情報をあらかじめ列挙して一ヶ所に記述しておく。型付き素性構造の設計とはこれらの情報を型階層中にどう配置するかを決めることに他ならないが、これとは別に「表示する時に」どう配置するかを決めるのがビューである。本システムではまず、図 5 のような形式で「語もしくは句が担うべき情報」を指定する。この文は、語や句は feat_struc 型 (もしくはその下位型) の素性構造で表現され、「語もしくは句が担うべき情報」には morphon, inflection, part_of_speech, case, saturation, role の 6 つがあるということを宣言している。それぞれの行で at の右側はその情報が配置されるパスを意味する。{infl,form} や /[^sit,reln]/ はパスに対する正規表現であり、それぞれ infl と form 素性、および sit と reln 素性以外の素性を表している。

このような宣言をふまえて、次のようにビューを定義する。

```
define view myview
  case at CASE
  saturation at SUBCAT
  roles at SEM|‘/*|(*)/‘
end
```

この文は myview という名前で新しいビューを定義しており、そのビューでは (もとの素性構造でどこに配置されていても) case に関する情報は CASE という (仮想的な) 素性に、saturation に関する情報は SUBCAT という素性に、thematic_role に関する情報は SEM|～ という素性に配置して表示せよ、ということを指定している。ここで、‘/*|(*)/‘ はパスに対するパターンマッチおよびその評価を表しており、() 内のパターンにマッチした部分がこの部分と置き換わる。

このように定義した myview というビューを使うと、(1) のような素性構造が (2) のように出力される。素性構造エディタでは、予め定義しておいた複数のビューを、必要に応じてメニューで切り替えることができる。

(1)
$$\begin{bmatrix} \text{syn} & \begin{bmatrix} \text{head} & \begin{bmatrix} \textit{verb} \\ \text{case} \quad \text{nil} \end{bmatrix} \\ \text{val} & \begin{bmatrix} \text{subcat} & \langle \boxed{1} \rangle \\ \text{adjcnt} & \langle\,\rangle \end{bmatrix} \end{bmatrix} \\ \text{sem} & \begin{bmatrix} \text{index} & \boxed{2} \\ \text{restr} & \left\langle \begin{bmatrix} \text{sit} & \boxed{2} \\ \text{reln} & \text{run} \\ \text{runner} & \boxed{1} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}$$

(2)
$$\begin{bmatrix} \text{CASE} & \text{nil} \\ \text{SUBCAT} & \langle \boxed{1} \rangle \\ \text{SEM} & \begin{bmatrix} \text{runner} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

2.2.3 出力フォーマット

以前のバージョン [8] でも avm.sty スタイルファイルに対応した LaTeX ファイルや PostScript ファイルを出力することはできたが、今回からさらに XML フォーマットでも出力できるようにした。DTD は図 6 のように簡単なものに固定した。atomic, compound のような一般的なタグ名ではなく、型名・素性名をタグ名として素性・型定義をそのまま DTD として使うことも考慮したが、あとから自由に加工することを考えてできるだけ単純な出力になるような方法を採用した。素性構造エディタの出力した XML ファイルと図 6 の DTD、出力した XML ファイルを HTML に変換するための XSLT ファイルとスタイルシートを合わせることで、Web ブラウザを素性構造ブラウザとして使うことができる。

3 関連システム

[11] の最新バージョン (version 2.0) は LiLFeS [7] を GUI で操作するための Java で書かれたラッパーツールであり、解析結果を木構造で表示したり素性構造やその型階層を表示・編集したりすることができる。特に、二つの木構造や素性構造の差分を表示する機能を備え、解析器が大量の曖昧性を出力した時にそれらの違いを手早くみつけるのに役に立つ。

[10] では単一化文法を XML で記述することで、各文法における細かい記法上の違いを XSLT やスタイルファイルで吸収することを提案している。具体的には、サンプルとして作った文法を PATR と HPSG の二通りの記法で出力している。構造をもったデータを複数のアプリケーションで交換する際の共通フォーマットとして XML を使うというアイデアは今後一般的になると思われる。

```
define information (feat_struc)
  morphon at morphon|morph
  inflection at syn|head|{infl,form}
  part_of_speech at syn|head
  case at syn|head|case
  saturation at syn|val|subcat
  roles at sem|restr|first|[^sit,reln]
end
```

$$\begin{bmatrix} \text{morphon} & \begin{bmatrix} \text{morph} & \dots \end{bmatrix} \\ \text{syn} & \begin{bmatrix} \text{head} & \begin{bmatrix} \textit{pos} \\ \text{infl} & \dots \\ \text{form} & \dots \\ \text{case} & \dots \end{bmatrix} \\ \text{val} & \begin{bmatrix} \text{adjcnt} & \dots \\ \text{subcat} & \dots \end{bmatrix} \end{bmatrix} \\ \text{sem} & \begin{bmatrix} \text{index} & \dots \\ \text{restr} & \left\langle \begin{bmatrix} \text{sit} & \dots \\ \text{reln} & \dots \\ \dots \end{bmatrix}, \dots \right\rangle \end{bmatrix} \end{bmatrix}$$

図 5: 「語もしくは句が担うべき情報」の定義 (左) と想定している素性構造 (右)

```
<!ELEMENT doc ANY>

<!ELEMENT feature (name,(compound|atomic))>
<!ELEMENT compound (feature+)>
<!ELEMENT atomic EMPTY>
<!ELEMENT ref EMPTY>
<!ELEMENT name (#PCDATA)>

<!ATTLIST compound type CDATA #IMPLIED
                   id ID #IMPLIED
                   color CDATA #IMPLIED>
<!ATTLIST atomic type CDATA #IMPLIED
                 id ID #IMPLIED
                 color CDATA #IMPLIED>
<!ATTLIST ref id IDREF #REQUIRED
              color CDATA #IMPLIED>
<!ATTLIST name color CDATA #IMPLIED>
```

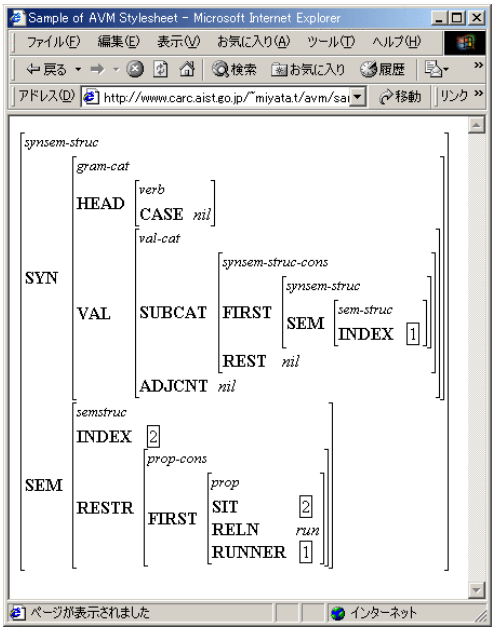

図 6: XML フォーマット用の DTD(左) と素性構造を Internet Explore で閲覧している様子 (右)

4 おわりに

本システムは C++ および GTK+ ライブラリを使って実装している。今年度中には公開予定である。

参考文献

[1] Norbert Bröker. The use of instrumentation in grammar engineering. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, pp. 118–124, Germany, July–August 2000.

[2] Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, 1992.

[3] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP) and the 1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. NAACL 132–139, USA, April–May 2000.

[4] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and the 8th Conference of the European Chapter of the Association for Computational Linguistics (ACL-EACL)*, pp. 16–23, Spain, July 1997.

[5] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley Longman, Inc., 1999.

[6] Taku Kudo and Yuji Matsumoto. Japanese dependency structure analysis based on support vector machines. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 18–25, Hong Kong, October 2000.

[7] Takaki Makino, Minoru Yoshida, Kentaro Torisawa, and Jun'ich Tsujii. LiLFeS — towards a practical HPSG parser. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL '98)*, Vol. 2, pp. 807–811, Canada, August 1998.

[8] Takashi Miyata and Yuji Matsumoto. Development system for feature-based unification grammar. In *Demonstration Abstracts at the 33rd Annual Meeting of the Association for Computational Linguistics*, pp. 31–32, Meryland, June 1999. Association for Computational Linguistics.

[9] Takashi Miyata, Akira Ohtani, and Yuji Matsumoto. An HPSG account of the hierarchical clause formation in Japanese — HPSG-based Japanese grammar for practical parsing —. In *Proceedings of the 15th Pacific Asia Conference on Language, Information, and Computation (PACLIC 15)*, pp. 305–316, Hong Kong, February 2001.

[10] The unification grammar markup language home page. Masters degree dissertation at School of Cognitive and Computing Science, Sussex University, September 2000. `http://www.cogs.susx.ac.uk/lab/nlp/ugml/`

[11] 今井久夫, 宮尾祐介, 辻井潤一. HPSG パーザーの為の GUI. 情報処理学会研究会資料, 第 98-82 巻 of *98-NL-127*, pp. 173–178. 情報処理学会, September 1998. `http://www-tsujii.is.s.u-tokyo.ac.jp/will/`.

[12] 宮田高志, 高岡一馬, 松本裕治. 単一化文法用 GUI デバッガの実装. 情報処理学会研究会資料, 第 99-2 巻 of *99-NL-129*, pp. 87–94. 情報処理学会, January 1999.

[13] 大谷朗, 宮田高志, 松本裕治. HPSG にもとづく日本語文法について — 実装に向けての精緻化・拡張—. 自然言語処理, Vol. 7, No. 5, pp. 19–49, November 2000.

[14] 大谷朗, 宮田高志, 松本裕治. 述語の隣接と述部の付加における意味的階層性. 情報処理学会研究会資料, No. 2000–107 in 2000-NL-140, pp. 103–110, 奈良先端科学技術大学院大学, November 2000. 情報処理学会.

[15] 大谷朗, 宮田高志, 松本裕治. HPSG に基づく述語タイプの記述 — 文法記述に求められる計算辞書の構成—. 第 18 回大会発表論文集, pp. 78–79, 公立はこだて未来大学, June 2001. 日本認知科学会.