自然言語処理 129－13
（1999. 1. 21）

単一化文法用 GUI デバッガの実装

宮田 高志, 高岡 一馬, 松本 裕治
奈良先端科学技術大学院大学 情報科学研究科
{takashi,kazuma-t,matsu}@is.aist-nara.ac.jp

素性構造単一化に基づく文法の開発を容易にするための GUI システムを作成した。このシステムは Chart 法に基づく構文解析において、すべての解析結果を CKY 表の形で表示したり、生成された部分木のもつ素性構造を AVM 表示したりできるほか、指定した部分単語列に対して (場合によっては一時的に一部の素性の値を変更して) 詳細な再実行を行なうことで文法のどこに不具合いがあっていつ失敗したのかを知ることができる。再実行時には Chart 法における拡張・補完および補強項の実行を一つのステップとして各状態が木構造で表示され、ユーザは任意の状態に対して変更を加えて実行することができる。本システムは構文解析器 SAX に対するインターフェースとして、Prolog, Tcl/Tk および Perl/Tk を用いて実装した。

[キーワード] 文法開発, デバッガ, 単一化文法, 型付き素性構造, グラフィカルユーザインターフェース

Implementation of GUI Debugger for Unification-based Grammar

MIYATA Takashi, TAKAOKA Kazuma, MATSUMOTO Yuji
Graduate School of Information Science, Nara Institute of Science and Technology

A GUI system that supports development and debugging of unification-based grammars is developed. The system presents all parses which are output from Chart algorithm in a CKY tabular form, displays the feature structure accompanied with each partial parse in an AVM form, and allows a mechanism for detailed re-parsing on a specified partial input sequence. These mechanisms enable the user to know where his grammar has a bug and when the parsing process was failed. When the system re-parses on a specific partial input sequence, expansion/completion operations in Chart algorithm and execution of auxiliary terms are illustrated as transitions among states and they forms as a tree structure. The user can modify and retry to process any state. The system is built as an interface to the syntactic parser SAX using Prolog, Tcl/Tk, and Perl/Tk.

[Keyword] grammar development, debugger, unification-based grammar, typed-feature structure, graphical user interface

1 はじめに

HPSG [9] や LFG [1] のような単一化に基づく文法体系では、理論の中で処理の順序を仮定しないという特徴がある。すなわち、ある句に対していくつかの制約 (constraint) が課されていた場合、それらがすべて満たされた時のみその句が認められるのであって、それらの制約を特定の順序で評価することが仮定されたり評価すべき制約が増減したりすることは、少なくとも理論の上では起こらない。しかし逐次的なアルゴリズムによって構文解析を行なう場合、なんらかの順序に従って制約を評価しなければならず、記述した文法と実際の処理の間にある程度の乖離が生じる。

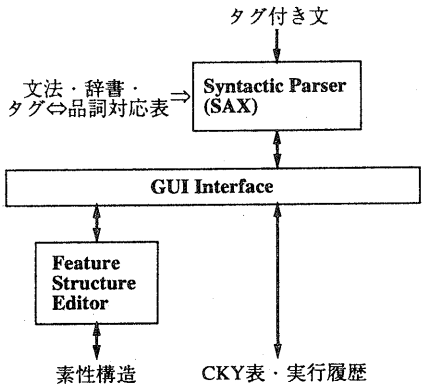

図 1: システム構成図

また、完成した文法ではなくバグを含んだ文法を修正する場合、本来平等な制約の間に「期待された評価結果を返した制約」と「期待された評価結果を返さなかった制約」という区別/順序がつくので、これを分かりやすくユーザに伝えることが重要である。

このように、理論の上では処理の順序を仮定しないように記述されている文法でも、デバッグにおいては順序をかなり意識する必要があり、このことが文法を開発する際の困難の一つとなっている。

文法開発におけるもう一つの問題点は、**素性構造** (feature structure) の複雑さである。素性構造とは素性と値の対の集合で表される構造であるが、値として再帰的に素性構造をとることが許されるので、埋め込み構造を作ることができる。さらに**構造共有** (structure sharing) が起こると一見しただけでは容易に把握できないような複雑な構造が頻繁に発生する。

以上の二つの問題を解決するために、既存の構文解析過程表示システム VisIPS [14] を基に次の点を拡張した文法開発支援システムを作成した。

- 入力単語列の任意の部分列に対して、実行の順序を明示した、詳細なトレースを行なう。
- 任意の (型付き) 素性構造に対してグラフィカルなインターフェースを用いて編集を行ない、編集結果に基づいて再解析を行なう。

システムの構成を図 1 に示す。

本論文では Prolog および Definite Clause Grammar (DCG) に関する知識は前提とする。これらに関する詳細は [12, 7, 5] などを参照されたい。

2 構文解析アルゴリズム

この節では、本論文のシステムが用いている構文解析器 SAX [16] の動作原理について簡単に説明し、デバッグのための情報をどのように取得するかを述べる。

2.1 Chart 法について

SAX は Chart 法とよばれる構文解析アルゴリズム (スキーム) を、ストリームを使った制御によって実現したシステムである。Chart 法では**活性弧** (active edges) と**不活性弧** (inactive edges) とよばれる二種類のデータ構造を用いる。活性弧は $[[\,]\beta_1,\ldots,[\,]\beta_i,[?]\beta_{i+1},\ldots,[?]\beta_n]\alpha$ $(1 \leq i < n)$ というもので、これは $\alpha \rightarrow \beta_1,\beta_2,\ldots,\beta_n.$ という句構造規則の右辺が i 番目の要素まで部分的に具体化されており、残りの $\beta_{i+1},\ldots,\beta_n$ が作られると統語カテゴリ α が出来上がる、という状態を表す。不活性弧は $[[\,]\beta_1,\ldots,[\,]\beta_n]\alpha$ というもので、統語カテゴリ α がすでに作られている状態を表す。これは $[\,]\alpha$ と略記することもある。さらに不活性弧にはその統語カテゴリが覆う句の開始位置と終了位置の二つの情報が付与されているとする。

1. $E = \{[\,]w_1,[\,]w_2,\ldots,[\,]w_N\}$ とする。ここで $[\,]w_i$ は入力単語列の各単語から作られた不活性弧で、E は活性弧および不活性弧のプールを表す。
2. E から不活性弧 $[\,]\beta'_1$ を取り出す。もし文法中に規則 $\alpha \rightarrow \beta_1,\beta_2,\ldots,\beta_n.$ が存在し、β'_1 と β_1 が単一化可能であれば、新しい活性弧 $[[]\beta''_1,[?]\beta'_2,\ldots,[?]\beta'_n]\alpha'$ を E に加える ($n = k = 1$ の時は、活性弧ではなく不活性弧 $[\,]\alpha'$ を E に加える)。この操作を**展開** (expansion) という。もとの二つの弧はそのまま E に残しておく。
3. E から活性弧 $A = [[\,]\beta_1,[\,]\beta_2,\ldots,[\,]\beta_{k-1},[?]\beta_k,\ldots,[?]\beta_n]\alpha$ を取り出す。もし E 中に不活性弧 $[\,]\beta'_k$ があり、かつ A 中でもっとも左の具体化されていない要素 $[?]\beta_k$ と β'_k が単一化可能であるならば、E に活性弧 $[[\,]\beta'_1,[\,]\beta'_2,\ldots,[\,]\beta'_{k-1},[\,]\beta''_k,[?]_{k+1}\beta'_{k+1},\ldots,[?]\beta'_n]\alpha'$ を加える (この結果、すべての $[?]\beta_i$ が具体化された時は、活性弧ではなく不活性弧 $[\,]\alpha'$ を E に加える)。ただし、$[\,]\beta'_{k-1}$ の終了位置と $[\,]\beta''_k$ の開始位置はそれぞれ隣合っていなければならない。この操作を**補完** (completion) という。もとの二つの

弧はそのまま E に残しておく。

4. 上のステップで E に新たな弧が全く加えられなかった時は手続きを終了し、文の統語カテゴリをもつ不活性弧を E から取り出して出力する。そうでないときは、ステップ 2, 3 を繰り返す。

上のアルゴリズムではどのような順序で不活性弧・活性弧を選択すべきかについてはなにも述べてはいないが、SAX では弧のプール E を終了位置によって分割して管理することで、重複なく不活性弧を生成している。[1]

2.2 デバッグのための情報とその提示方法

我々は、デバッグにおいて前節のアルゴリズムの各ステップにおいてどのような弧が生成されたかを追跡するという方針をとることにした。とくに、

- 必要な不活性弧がきちんと生成されているかどうか、
- その内容は正しいか

のチェックが容易に行なえるように、本システムでは弧のプール E の状態をノードとし、各ステップの実行をエッジとする木構造で実行の履歴を提示することとした。次の文法を用いて入力「きた 時」という二単語の列を解析する例をもとに、図 2 に E の状態遷移と実行の履歴との対応関係を示す。

$$N \rightarrow [きた]. \qquad N \rightarrow [時].$$
$$V \rightarrow [きた]. \qquad N \rightarrow V, N.$$

図 2 では、E が初期値である状態が ⟨Start⟩ というノードに対応している。同図上において E に不活性弧 [[ ] きた]N が追加されているが、これは同図下において [[ ] きた]N/inactive というラベルをもつノードに対応している。

厳密にいえば、E の状態のグラフは束 (lattice) とするのが自然である。つまり e_1, e_2 という二つの弧が独立に生成できる場合、e_1, e_2 の順で追加された状態と e_2, e_1 の順で追加された状態は同じであるからである。その場合、実行履歴のグラフはノードではなくエッジにラベルをふることになる。現在は実装の都合上、弧の生成された順序の違いを区別しているため、実行履歴は木構造としている。

設計に際しては、他にも次の点に留意した。

[1] 活性弧は中間的なデータで、必ずしもすべての可能な活性弧を生成する必要はないのに対し、不活性弧は (すべての解析結果を求める場合) すべての可能な不活性弧を (過不足なく) 生成する必要がある。

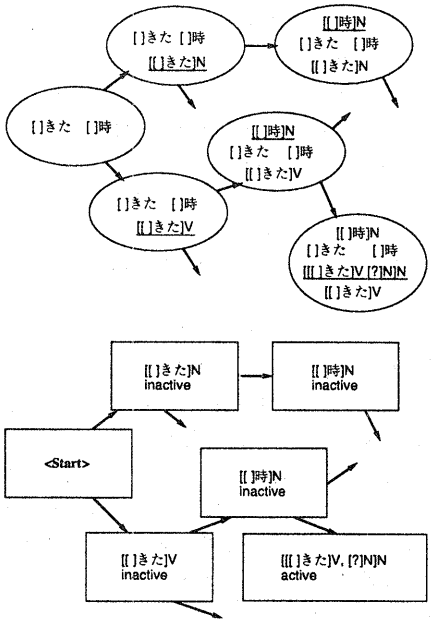

図 2: 弧のプール E の状態遷移 (上) と実行履歴 (下) との対応関係

- 曖昧性が非常に多い時でも、内容をできるだけ容易に確認できるようにする。
- 長い文に対しても、適当に省略したりまとめたりすることで結果を一目で把握できるようにする。
- ちょっとした変更ならば元の文法を修正・再読み込みを行なわなくても、その場で再実行できるようにする。文法が大きくなってくると再読み込みに手間がかかるようになるので、この機能は不可欠である。

3 設計方針

この節では、2.2 節で述べた問題点に対する本研究での解決策を示す。

3.1 曖昧性の抑制

曖昧性が多い時の表示の繁雑さは、文法作成初期の、制約が不足している状態ではとくに深刻な問題である。経験上、不要な句が排除されているかどうかよりも、必要な句が作られているかを調べることの方が多いので、あきらかに無関係な句は表示しないようなフィルターを入れることが望ましい。この

ために、予め部分的に括弧とタグをつけた文を入力とし、括弧やタグに違反する句は解析の段階で捨てるものとする。

逆に、ある括弧とタグに整合的な句が一つも作られなかったときは、単に解析を失敗させるのではなく、そのタグから予想される句を補ってとりあえず解析を続けさせれば、どんな制約を追加すべきかの見通しもたてやすい。さらにこの機構をある程度賢くすれば、半自動で文法的制約を獲得することも期待できる。

入力に括弧とタグを施すもう一つの理由として、統語解析の入力としては形態素は細かすぎることが挙げられる。この問題に関しては、形態素の列を正規表現で定義した文節にまとめたり [4]、形態素解析を係り受け解析と統合したり [15] する解決方法が考えられるが、いずれにしても統語解析の前処理という形になるので、本研究では細かすぎる部分は括弧付けしてタグをふってあるものとした。[2]

3.2 長文の扱い

長文の解析結果が見づらいことは、結果の一部を隠しておけるようにすれば解決する。実際、解析がうまくいったかどうかの確認はトップダウンに行なうことが多いので、構文木の階層構造に従って任意の部分木を畳んでおけるようにしておけばよい。

ただし、この畳込みによって一部の情報が失われる。例えば図 3 上の状態で p_1 を畳むと同図下の状態になるが、p_2 が見かけ上消えてしまう。それでも「単語 1」から「単語 3」までの全体を覆う句として左上に p_0^2 が表示されているので、このことはそれほど深刻な欠点ではないと判断した。

3.3 GUI からの文法変更・再実行

解析が終わった後でちょっとしたタイプミスや見落としなどに気がついて、もう一度やり直させたいことはよくある。あるいは「この句とこの句に対して、このスキーマが適用されるはずなのに、どこで失敗しているのか？」というように解析が終わった後で部分的に再実行したいこともある。本研究では、型付き素性構造で表現された不活性弧の内容を編集するためのシステムを用意し、その編集をあたかも弧のプール E に対する状態遷移であるかのように見なすことで、再実行の機構を提供することとした。図

[2] 現在の実装では与えられた括弧をヒントにして、括弧がない部分に対しては自動的に形態素解析を施すようにしている。

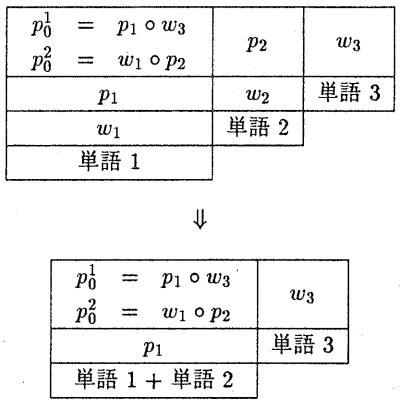

図 3: 畳込みによって一部の情報が失われる例

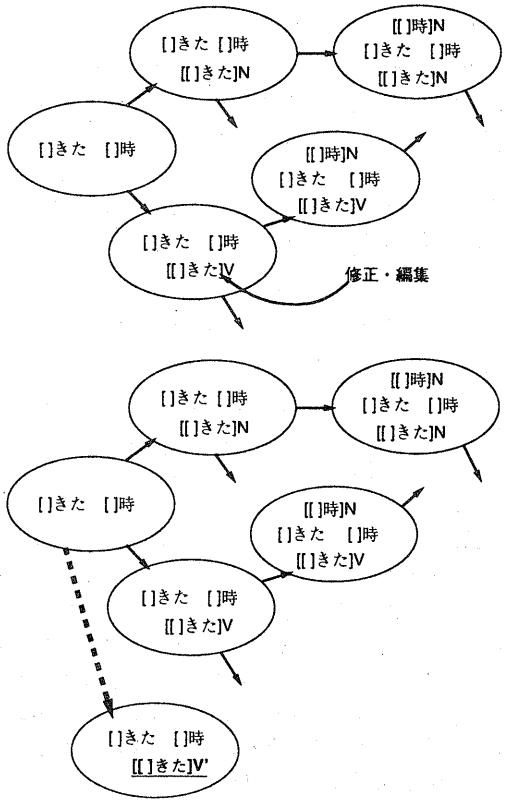

図 4: 不活性弧の編集と再実行

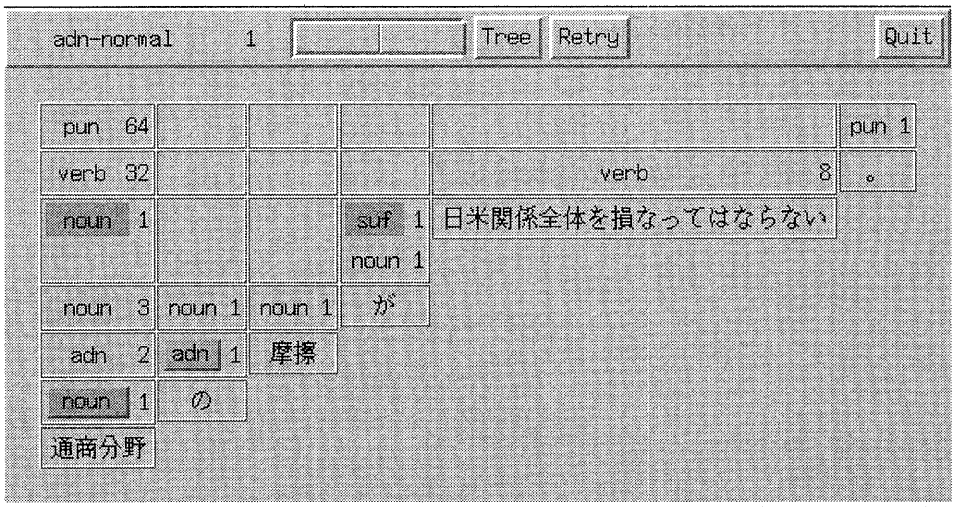

図 5: 解析結果を表す CKY 表 (表の各欄にはその真下の単語から右の単語までを覆う不活性弧が記述される。なお「日米関係全体を損なってはならない」の部分の解析結果は折り畳んである)

4 の上の図において、初期状態から追加された [[ ] きた]V という弧の内容を修正・変更したとする。修正・変更した弧以外はもとの状態と同じ弧からなる状態 {[ ] きた, [ ] 時, [[ ] きた]V'} を、もとの状態の兄弟として、破線で表されている仮想的な状態遷移によって追加したのが同図下である。

4 部分的再実行・トレースの実行例

部分的再実行とは、取り敢えず得られた結果に関して挙動が不審な部分だけをもう一度 (今度は実行過程を確認しながら) 実行させることで文法中の誤りを効率的に発見するための機能である。例えば、図 5 では「通商分野の」という句に対して二つの adn という統語カテゴリをもった不活性弧が作られていることが示されているが、その不活性弧の内容 (図 6) が不適切であることが分かったとする。図 6 において、sem 素性の arg1 は *content* 型ではなく、「通商分野」が担っているはずの意味表現 $_{obj}$[name 通商分野] でなければならない。[3]

そこで、「通商分野」と「の」を覆う不活性弧を選択して再実行機能を起動する。図 7 は修正せずに実行したときの履歴の一部を表す。各ノードのラベルの上の一行はその時生成された弧の中で、具体化された要素が覆う部分単語列を表し、下の一行は生成された弧の種類を表す。弧の種類は活性弧・不活性弧の他に**補強項** (auxiliary term) の実行が残っているかどうかの区別があるので、下の四種類となる。

completed	補強項が未チェックの不活性弧
inactive	補強項がチェック済みの不活性弧
expanded	補強項が未チェックの活性弧
active	補強項がチェック済みの活性弧

今の場合、実行履歴の最初の「通商分野/inactive」の内容が不十分であることが分かっているので、これを図 8 のように修正する。ここでは core 素性に sem 素性を追加した後、sem 素性の型を *content* から *obj* へ変更し、その name 素性の値を「通商分野」という文字列とした。

すると、修正した不活性弧を含むような新しい状態「通商分野'/inactive」が作られ (図 9)、この状態に対する処理をすすめると、望み通りの結果が得られる。(図 10)

このように本システムでは実行履歴が木構造で表示されているので、任意の時点に戻って素性の値を適宜変更しながら再実行することができる。一般的なデバッグの戦略としては、実行履歴の木に対して [11] で提案されている矛盾点追跡アルゴリズムを手動で行なうことになる。

[3] 「通商分野」という複合語が辞書に登録されていなかったのが原因である。

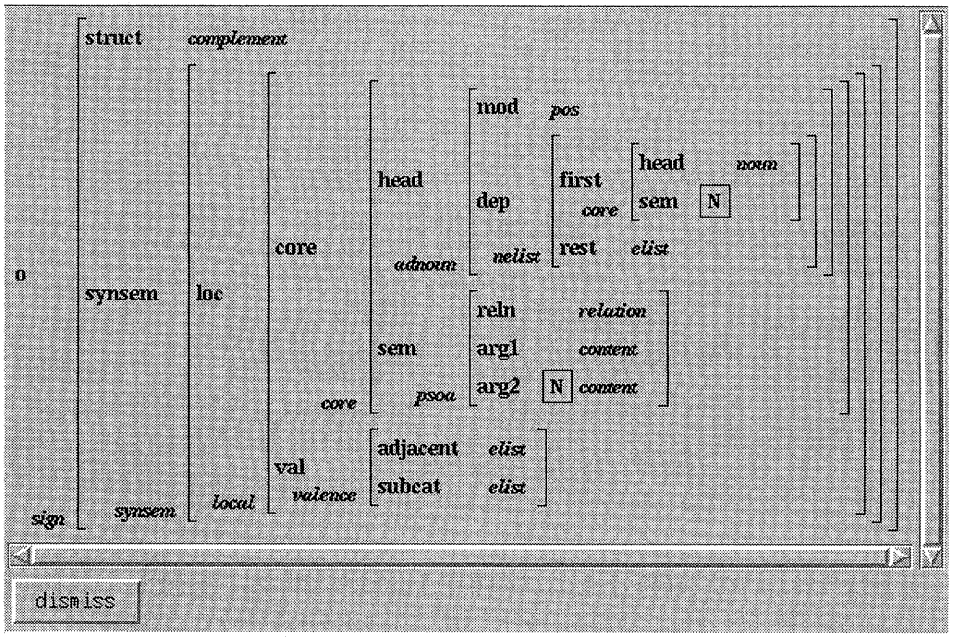

図 6: 「通商分野の」に対する不適切な解析結果の一つ (素性 arg1 の値は「通商分野」の意味表現である必要がある。もう一方の解析結果については省略した。)

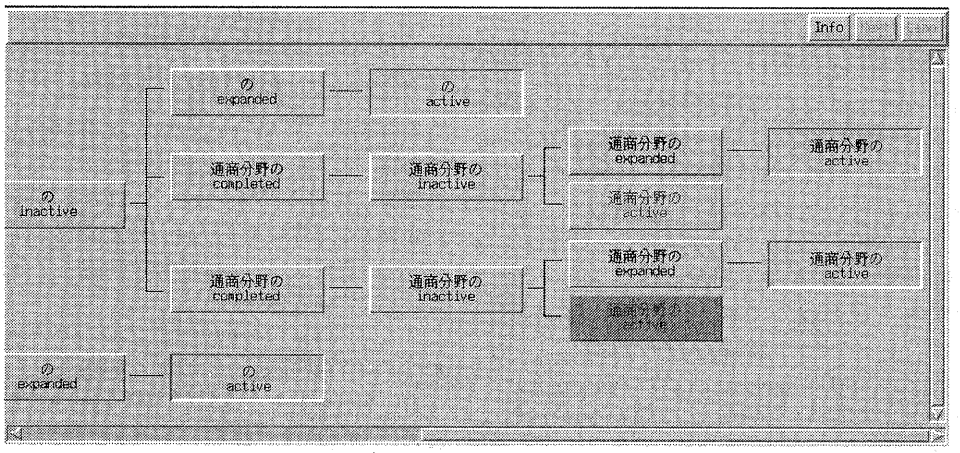

図 7: 再実行機能を使った時の履歴の様子 (指定した範囲を覆う不活性弧が一つ以上生成できた状態は赤字のラベルで、そうでない状態は青字の沈んだラベルで表示される。)

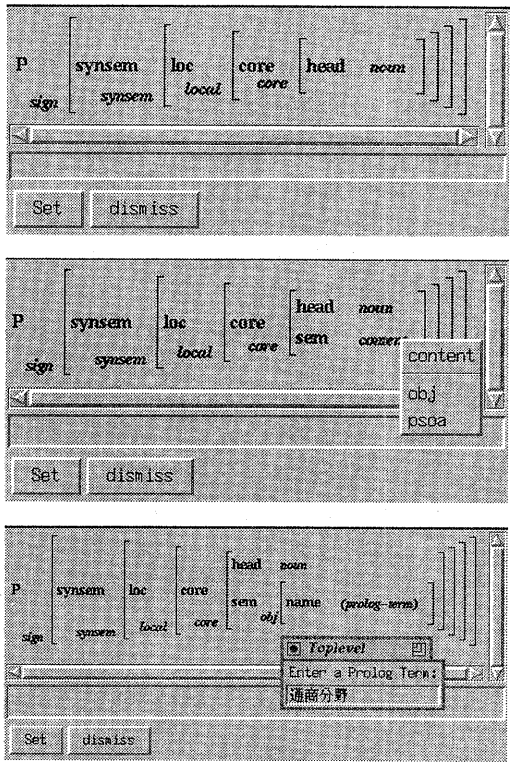

図 8:「通商分野/inactive」がもともと担っていた内容 (上) とその修正の様子 (中・下)

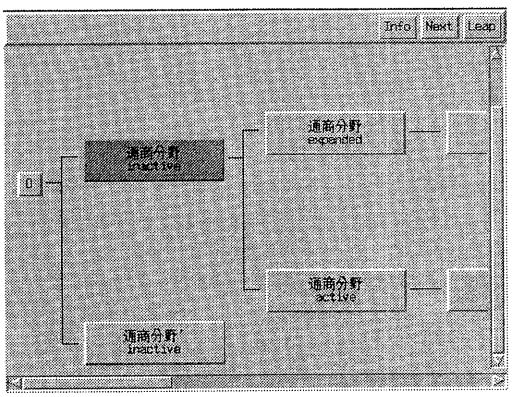

図 9: 不活性弧の内容を修正することによって生成された新たな状態

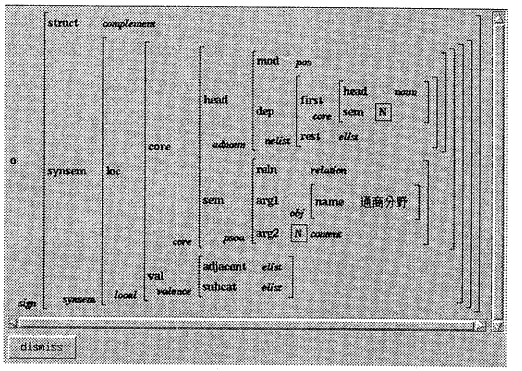

図 10: 修正結果が反映された、「通商分野の」に対する解析結果 (図 6 と比べて sem 素性の arg1 の値が $_{obj}$[name 通商分野] になっている。)

5 先行研究との比較

本システムのもととなった VisIPS [14] においても解析の途中で不活性弧の内容を変更する機能は実装されているが、修正できるのはその不活性弧が使用される前だけなので、試行錯誤が不可欠なデバッグの機能としては不十分である。また、HPSG パーサのための GUI ツールである will [13] では構文木および型付き素性構造のグラフィカルな表示機能を提供しているが、素性構造に対して定義に整合的な編集のみを許すような機構[4] や解析の再実行といったデバッグのための支援機能は備えていない。

文法開発支援環境、とくに素性構造単一化に基づく文法の開発を支援するものとしては、ALEP [10] をはじめとして XTAG [8] や ConTroll [6] などがある。これらのシステムはいずれも解析木や素性構造の表示・編集機能を提供しているが、本研究で実装したシステムの特徴は、木構造で表示した実行履歴中の任意の状態から試行錯誤によって再実行できる点にある。

参考文献

[1] Joan Bresnan, editor. *The Mental Representation of Grammatical Relations.* MIT Press, 1982.

[4] 型付き素性構造の編集は ProFIT [3] や ALE [2] などの型付き素性構造の単一化エンジンさえあれば、単独のシステムとしても利用可能である。型階層や素性の定義の記法はこれらに準拠している。

[2] Bob Carpenter. *ALE: Attribute Logic Engine / User's Guide Ver 3.1.* Universität Tübingen, September 1998. `http://www.sfs.nphil.uni-tuebingen.de/~gpenn/ale.html`.

[3] Gregor Erbach. *ProFIT 1.54 User's Guide.* Universität des Saarlandes, December 1995. `http://www.coli.uni-sb.de/~erbach/formal/profit/profit.html`.

[4] Masakazu Fujio, Yuji Matsumoto. "Japanese Dependency Structure Analysis Based on Lexicalized Statistics". In *Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing*, pp. 88–96, Granada, Spain, June 1998.

[5] Gerald Gazdar and Chris Mellish. *Natural Language Processing in Prolog.* Addison-Wesley Publishing Company, 1989.

[6] Thilo Götz and Walt Detmar Meurers. "The ConTroll System as Large Grammar Development Platform". In *Proceedings of Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, pp. 38–45, Madrid, Spain, July 1997.

[7] Richard A. O'Keefe. *The Craft of Prolog.* MIT Press, 1990.

[8] Patrick Paroubek, Yves Schabes, and Aravind K. Joshi. "XTAG — A Graphical Workbench for Developing Tree-Adjoining Grammars". In *Proceedings of the 3rd Conference on Applied Natural Language Processing*, pp. 216–223, Trento, Italy, August 1992.

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

[10] Paul Schmidt, Axel Theofilidis, Sibylle Rieder, and Thierry Declerck. "Lean Formalisms, Linguistic Theory, and Applications. Grammar Development in ALEP". In *Proceedings of the 16th International Conference on Computational Linguistics (COLING '96)*, Vol. 1, pp. 286–291, Copenhagen, Denmark, August 1996.

[11] Ehud Y. Shapiro. "Inductive Inference of Theories from Facts". In J. L. Lassez and G. Plotkin, editors, *Computational Logic*, pp. 199–254. MIT Press, 1991.

[12] Leon Sterling and Ehud Shapiro. *The Art of Prolog — Advanced Programming Techniques —.* MIT Press, 1986.

[13] 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.

[14] 中山拓也, 松本裕治. 言語解析過程表示システム VisIPS (Tcl/Tk 版) 使用説明書. 奈良先端科学技術大学院大学, 第 version 1.0 版, 1996.

[15] 中山拓也, 松本裕治. "日本語係り受け解析の高速化手法". 第四回年次大会発表論文集, pp. 81–84, 九州大学, March 1998. 言語処理学会.

[16] 松本裕治, 伝康晴, 宇津呂武仁. 構文解析システム SAX 使用説明書. 奈良先端科学技術大学院大学, 第 version 2.0 版, 1993.