ICOT Technical Memorandum: TM-0756

TM-0756

LTB操作説明書

田中裕一，奥西稔幸，久保幸弘，
赤坂宏二，幡野浩司，杉村領一(松下)

July, 1989

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

LTB 操作説明書

– LTB 概要編 –

(LTB1.1 版)

ICOT 第二研究室

平成元年 3 月 31 日

本説明書は汎用日本語処理系 LTB(Language Tool Box) の概要を述べている。

汎用日本語処理系 LTB(Language Tool Box) の説明書は次のような構成になっている。

- LTB 概要編
- LTB マスタ辞書編
- CIL 編
- LAX 編
- SAX 編
- 文生成編

目次

第 1 章

LTB 概要

ICOT における自然言語処理の研究開発は、知的ソフトウェア・インタフェースを目的とすることから、談話理解についての技術開発に重点が置かれいる。具体的な研究開発としては、談話理解実験システム DUALS を逐次論理型推論マシン PSI 上に現在までに 3 回試作しており、自然言語理解に必要な種々の技術が、この実験試作と評価を通じて整理されてきた。

これらの各技術は、エキスパート・システムをはじめとする種々の応用分野において知的なインタフェースを具現化する道具となることを目指しているので、種々の開発環境を具備したソフトウェアとして整備されてきている。これが汎用日本語処理系（Language Tool Box 略して LTB）である。

LTB に用意された技術には、日本語についての種々の情報が計算機処理できる形で蓄積された辞書、日本語の文字列を有意味な単語などの並びに変換する形態素処理系 (略称 LAX)、この並びを更に関連づけて文の意味を得る構文解析系 (略称 SAX)、日本語の文字列を生成する生成系、そして、辞書や種々の処理系が必要とする日本語の意味を記述するための種々の基本機能を提供する意味記述言語 (略称 CIL) がある。

これら各処理系の中には、LTB としてソフトウェアをまとめることが計画される以前から、開発環境などの研究開発が開始されていたものがあるが、汎用日本語処理系のツールとしての各処理系の研究開発は、1987 年から本格的に開始された。そして、1988 年 6 月に開かれた第五世代コンピュータに関するシンポジウムにおいてプロトタイプのデモが行なわれた。以上の経緯から、1989 年 3 月時点で研究開発開始から約 2 年しか経過しておらず、まだまだ使い勝手や提供されるべきデータに至らぬ点が残っており、また、開発期間の相違などから各処理系の完成度にもばらつきがある。

LTB を使用する際に必要となる情報は、意味記述言語 CIL、辞書、形態素解析処理系 LAX、構文解析処理系 SAX、生成処理系各々に用意されたマニュアルに記述されている。マニュアルは、プログラムの詳細な処理内容に関する情報が記述されたソフトウェア仕様書と、処理系を種々の目的に応じて利用する際に必要な情報を取りまとめた操作説明書からなる。

また、技術背景等、マニュアルに記述しきれない点もあるので、現在までに発表された論文を補足資料として添付した。

LTB 操作説明書

— マスタ辞書編 —

(LTB1.1 版)

ICOT 第二研究室

平成元年 3 月 31 日

本説明書は LTB マスタ辞書の構成および辞書のアクセス・ツールの操作方法について記述している。

- LTB マスタ辞書の記載内容
- LTB マスタ辞書の構造とフォーマット
- LTB マスタ辞書のアクセスメソッド

汎用日本語処理系 LTB(Language Tool Box) の説明書は次のような構成になっている。

- LTB 概要編
- LTB マスタ辞書編
- CIL 編
- LAX 編
- SAX 編
- 文生成編

目次

第1章

LTB マスタ辞書の記載内容

1.1 概要

1.1.1 はじめに

LTB において、LTBマスタ辞書は他のツールから独立している。現在の版では、LTB の各ツールが、必要とする情報をあらかじめマスタ辞書の情報に追加し、その上で利用するという形式をとっている。

LTB マスタ辞書には、LTB の形態素解析部、構文解析部、文生成部が使用する単語の形態的情報および構文的情報と、若干の意味的情報が記載されている。本章では、マスタ辞書に記載されている構文的情報を中心に、その文法と記載内容を説明する。

マスタ辞書には、名詞、動詞、形容詞、副詞、連体詞の5つの品詞の単語に関する情報が登録されている。これらの記載は、あらかじめ定められた辞書フォーマットに従って行われている。マスタ辞書のフォーマットは、品詞ごとにそれぞれ違った記載項目をもっている。このフォーマットの詳細は、第2章で説明する。

マスタ辞書の内容は、あらかじめ用意されたアクセスメソッドによって参照することができる。このアクセスメソッドの内容と使用法は、第3章で説明する。

現在の版において、マスタ辞書に蓄積されている語の総数はおよそ 3600 語である。その内訳は、次の表に示す通りである。

品詞	語数
名詞	約 2200 語
動詞	約 1100 語
形容詞	約 100 語
副詞	約 200 語
連体詞	約 30 語

以下、本節ではマスタ辞書の品詞分類、用言の活用形を説明する。また、次節以降で各品詞ごとの辞書の記載内容を、適宜マスタ辞書の文法に言及しつつ説明する。

1.1.2 マスタ辞書の品詞分類

LTB では、助詞、助動詞、および補助動詞に関する情報は、各ツールにあらかじめ用意されている。したがって、マスタ辞書に記載するのは、これら以外の品詞の単語である。マスタ辞書に登録する単語は、次のような基準で、前述の5つの品詞に分類される。

1.2. 動詞に関する記載内容

動詞 ： 活用し、終止形(現在形)がウ段音で終わる単語
形容詞: 活用し、終止形(現在形)がイ段音で終わる単語
連体詞: 活用せず、文末に来ず、体言のみを修飾し、他から修飾されない単語
副詞 ： 活用せず、文末に来ず、用言を修飾することのできる単語
ただし、形容動詞的活用をもつ単語や、『な』または『の』を伴って連体修飾する単語は名詞に入れる
名詞 ： 動詞、形容詞、連体詞、副詞のいずれにも属さない単語

このように、マスタ辞書の文法では、従来の文法よりも名詞の範囲がかなり拡大されている。従来の形容動詞や、『する』を伴ってサ変動詞化する単語、および副詞としての働きをもつ時や数量を表す単語などは、すべて名詞として扱われる。

名詞は、さらに次の3つに分類される。

形容動詞性名詞: 形容動詞型の活用をする名詞
サ変動詞性名詞: 『する』を伴ってサ変動詞化する名詞
普通名詞 ： 形容動詞性名詞、サ変動詞性名詞以外の名詞

従来の文法における接続詞はマスタ辞書では副詞に含めており、副詞に対して結合に関する情報を付与することによって処理を行う。

1.1.3 用言の活用形

動詞の活用において五段活用を採用すると、終止形以外の活用形において、音便形のようにひとつの活用形が複数の活用語尾をもつ場合がありうることになる。これは、助動詞や接続助詞に前接する活用語尾を判定するときに不都合が多い。そこで、マスタ辞書の文法では、ひとつの活用形がひとつの活用語尾になるように、以下のように用言の活用形を細分している。

(1) 『ない』が後接する未然形
(2) 『ぬ』が後接する未然形
(3) 『た』が後接しない連用形（動詞と助動詞）
(4) 『た』や『ございます』が後接しない連用形（形容詞と形容動詞）
(5) 終止形（連体形と合わせる）
(6) 「～な」となる連体形（形容動詞とナ型形容詞）
(7) 『よ』が後接する命令形
(8) 『よ』が後接しない命令形（一段活用動詞）

さらに、マスタ辞書の文法では、従来の文法で助動詞とされてきた『た』の活用形『(9)たろ(う)、(10)た、(11)たら』、および接続助詞とされてきた『(12)たり、(13)て』を活用語尾とみなす。これにより、接続による音便処理は不要となる。また、活用しない助動詞である『(14)う(よう)、(15)まい』、および接続助詞の『(16)ば』も、活用語尾とみなす。

上記の活用形にこれらを加えて、マスタ辞書の文法では、用言に対して16段の活用形を設定している。

1.2 動詞に関する記載内容

マスタ辞書では、動詞に関して以下のような項目の構文的情報が記載されている。

(1) 活用型
(2) 態

（3）相
（4）自他
（5）可能動詞化
（6）意志性
（7）派生名詞

1. 活用型

マスタ辞書の文法では、従来の五段動詞を強変化動詞と呼んでいる。また、一音の上一段動詞、下一段動詞で語幹がなくなることを避けるため、両者を合わせて弱変化動詞とし、活用する語尾だけを活用表に記載している。

従来のサ変動詞とカ変動詞はそのままであるが、これらに加え、不規則な活用をするものとして次の3つの活用型を設けている。

ある動詞 ： 『ない』が後接しない動詞
(例) ある
いく動詞 ： カ行五段活用で、連用形がイ音でなく、促音便となる動詞
(例) 行く
なさる動詞: 命令形が「～い」の形になる動詞
(例) なさる　下さる

また、従来上一段動詞とサ変動詞のそれぞれに登録されていた、「～じる」および「～ずる」の形をもつ『感』、『信』、『存』などの漢語系サ変動詞は、まとめて弱サ変動詞としている。これらをまとめているのは、「～じ」となる活用形において上一段動詞とサ変動詞の形態的区別ができないからである。

サ変動詞は、語幹が単独では他の品詞(名詞や副詞)にならない『関する』、『有する』などの動詞である。マスタ辞書では、『愛する』、『恋する』などの動詞は『愛』、『恋』をサ変動詞性名詞に登録することで処理し、『しんみりする』、『ゆっくりする』などの動詞は『しんみり』、『ゆっくり』を副詞に登録し、サ変動詞用法があることを明記することで処理している。ただし、『感ずる』、『損ずる』などは語幹が名詞となるが、弱サ変動詞として動詞にも登録している。

以上、動詞の活用型は次の8つである。

強変化｜弱変化｜カ変｜サ変｜弱サ変｜ある｜いく｜なさる

2. 態

動詞の態は、次の4つの態の表現の有無を、それぞれ**あり／なし** の2値の属性値によって登録している。

直接受動: 通常の受身表現
(例) 「旗が振られる」「ディスクが破壊される」
間接受動: 主格が被われる被害の受身表現
(例) 「彼女が雨に降られる」「親が子に死なれる」
使役 ： 使役表現
(例) 「彼に船を漕がせる」「トラックに荷物を運ばせる」

1.2. 動詞に関する記載内容

授受表現: 授受表現
(例) 「〜てもらう」「〜てくれる」「〜ていただく」

3. 相

動詞の相は、それぞれの動詞のもつ基本アスペクトを、下記に示す4つの属性値からひとつ選んで登録している。

状態 : テイル形がない動詞
(例) ある いる 値する
準状態: テシマウ形やカケル形がない動詞
(例) そびえる 優れる 劣る
瞬間 : 単数主語文に、ダス形がない動詞
(例) 死ぬ 知る 結婚する 寝る 座る 立つ
継続 : 上記以外の動詞
(例) 太る 増える 枯れる 読む 話す 走る

相は、補助用言との接続可能性の検査や、複合アスペクトの合成・分解に用いられている。また、時を表す副詞の一部との共起性の検査にも使用されている。

4. 自他

動詞の自他は、それぞれの動詞の自他性を、下記に示す5つの属性値からひとつ選んで登録している。

絶対自動詞: 形態的に対立する他動詞のない動詞
(例) 死ぬ 泣く 歩く 走る
絶対他動詞: 形態的に対立する自動詞のない動詞
(例) 殺す 切る 飲む 買う
相対自動詞: 形態的に対立する他動詞がある動詞
(例) 壊れる あく 閉まる 回る
相対他動詞: 形態的に対立する自動詞がある動詞
(例) 壊す あける 閉める 回す
両用動詞 : 目的格をもつ他動詞形と、目的格を主格にした自動詞形の両方の用法をもつ動詞
(例) ひらく 閉じる

5. 可能動詞化

「-eru」がついて可能動詞となることができるか(可)できないか(不可)、あるいは当該の動詞自体が可能動詞であるかどうかの情報を、下記に示す3つの属性値からひとつ選んで登録している。

可 | 不可 | 可能動詞

6. 意志性

意志性のある動詞は、有意志物または準有意志物が行為格 (agent case) にくる、瞬間相や継続相の動詞でなければならない。

実際には判別の難しい動詞もあるが、「私も～する」、「～するのをやめる」などの表現の可能な動詞を意志性ありとし、そうでない動詞を意志性なしとする。マスタ辞書では、これをあり／なしの2値の属性値によって登録している。

7. 派生名詞

当該の動詞に派生名詞が存在する場合、それらを並べて記載している。連用形がそのままの形で名詞的に使われ、かつ意味も変化しないときには記載していない。ただし、名詞化するときに送り仮名がなくなる場合などには記載している。

1.3 形容詞に関する記載内容

マスタ辞書では、形容詞に関して以下のような項目の構文的情報が記載されている。

(1) 活用型
(2) 形容動詞用法
(3) 接尾辞
(4) 骨語用法

1. 活用型

マスタ辞書では、形容詞の活用を次の3種類に分類し、そのいずれかを記載している。

ナ型形容詞: 連体詞的な活用をもつ形容詞
(例) 大きい
サ型形容詞: 『そう』の前接形が「～さ」の形になる形容詞
(例) よい ない
普通形容詞: 上記以外の形容詞

2. 形容動詞用法

『暖かい』や『細かい』のように形容動詞的活用 (「～に」の形) をもつことができるかどうかの情報を、形容動詞用法の有無として、あり／なしの2値の属性値によって登録している。

3. 接尾辞

形容詞は、いろいろな接尾辞がついて他の品詞に派生することがある。下記の接尾辞に対しては、あらかじめ用意された規則により構文的な情報を合成して処理を行っている。

1.4. 連体詞に関する記載内容

まる: 強変化動詞を派生する。対象が、形容された状態へ自動的に変貌する。
める: 弱変化動詞を派生する。対象が、形容された状態へ他動的に変貌する。
がる: 強変化動詞を派生する。感情をもった主体が、形容された状態を感知する。(客観)
む : 強変化動詞を派生する。対象が、形容された状態へ自然に変貌する。
あるいは、感情をもった主体が、形容された状態を感知する。(主観)
げ : 名詞を派生する。形容された状態を感知している様子。
み : 名詞を派生する。形容された状態自体。
あるいは形容された状態をもつもの。
らか: 名詞を派生する。形容された状態自体。
め : 名詞を派生する。形容された状態に近い状態。

マスタ辞書では、これらの接尾辞のうち、当該の形容詞に接続が可能なものすべてを並べて記載している。

4. 畳語用法

形容詞には、語幹を重ねて、意味を強めた副詞や繰り返しを表す副詞を作ることのできるものがある。

(例) 丸い→丸々　　近い→近々(チカヂカ、キンキン)

マスタ辞書には、このような畳語用法の可能な形容詞について、副詞の記述に準じて以下の項目を記載している。(第5節を参照。)

(1) 読み
(2) にと
(3) サ変動詞用法

1.4　連体詞に関する記載内容

形容詞にナ型活用を設定したことにより、『大きな』とか『小さな』が連体詞ではなくなった。マスタ辞書の連体詞には、『同じ』や『名誉ある』などの通常の連体詞に加えて、『この』や『その』などの「こそあど」の類いと、『あらゆる』や『ある』などの限量的な単語が含まれている。

『この』や『その』は指示連体詞として知られているが、指示体系は連体詞だけではなく、次の表のように他の品詞にも及んでいる。

品詞	指示	近称	中称	遠称	不定称
名詞	物	これ	それ	あれ	どれ
	時(から／まで)	これ	それ	あれ	
	複数物	これら	それら	あれら	
	場所	ここ ここら	そこ そこら	あそこ あそこら	どこ どこら
	場所・人	こちら こっち こなた	そちら そっち そなた	あちら あっち あなた あんた	どちら どっち どなた
	人	こいつ こやつ	そいつ そやつ	あいつ あやつ	どいつ
連体詞	事物	この	その	あの	どの
	事象	こんな	そんな	あんな	どんな
副詞	事象	こう	そう	ああ	どう

これらの単語は、マスタ辞書とは別に、各ツールによって用意されている指示体系の辞書に登録されている。また、この辞書には「こそあど」系のほか、「か」系の『かれ、かのじょ、かれら、かのじょら、(ここ)かしこ、かなた、かの』や、「いづ」系の『いづれ、いづこ』、さらに人称代名詞の『私、僕、俺、君、お前、誰』、不定称の『いつ、なぜ、何、如何』なども登録されている。

照応処理や代名詞化、あるいは質問事項の解釈などの処理を行うときには、この指示体系辞書を用いて行う。

以上のような処理を行っているため、マスタ辞書では、連体詞に関して特に構文的情報を与えていない。

1.5 副詞に関する記載内容

副詞は活用しないとされているが、『ドキ』や『バタ』などの象徴語から派生した副詞では、『ドキリ、ドキン、ドキット、ドキドキ、ドッキリ、ドッキシ、ドッキン』など多くの変形をもち、ある程度類型化することが可能である。しかし、これらの変形による意味の変化は明確ではないので、現在の仕様ではこれらを別の副詞として登録している。

マスタ辞書では、副詞に関して以下のような項目の構文的情報が記載されている。

(1) 細分類コード
(2) にと
(3) サ変動詞用法

1. 細分類コード

情態、程度、量、概括量、主体めあて、時間関係、陳述など副詞の細分類名を記載している。

2. にと

当該の副詞が、「～に」または「～と」を後接してほぼ同じ意味の副詞となるかどうかを、次の6つの属性値からひとつ選んで記載してある。

にと: 「～に」と「～と」の両方の形が可能な副詞　(例)　段々　次々
に : 「～に」の形が可能な副詞　(例)　すぐ　大変

1.6. 名詞に関する記載内容

と ：「～と」の形が可能な副詞	(例) さっぱり
に要: 「～に」の形に必ずなる副詞	(例) 一既 無性
と要: 「～と」の形に必ずなる副詞	(例) 赤々 何彼
単独: どちらの形にもならない副詞	(例) 暫く かなり

ひとつの副詞に語義によって異なる使い方がある場合には、それらの語義レコード(次章を参照)を別に用意している。

(例) あまり（1．度を過ぎて）→ に
あまり（2．さほど） → 単独

3. サ変動詞用法

副詞には、「～する」を後接してサ変動詞を作ることのできるものがある。

(例) 終始 すっきり ぺこぺこ ゆっくり

マスタ辞書には、このようなサ変動詞用法の可能な副詞について、下記の項目が記載されている。ただし、『ぱっと(しない)』、『暫く(して)』、『忽然と(して)』など、終止形で終わる文が作りにくい副詞については、サ変動詞用法を認めない。

活用形: 次の属性値のいずれかを記載する。

ト可サ変 ：「～する」と「～とする」の両方の形が可能な副詞
ト不可サ変: 「～する」の形のみ可能な副詞

態 ：動詞と同じ。
相 ：動詞と同じ。
自他: 動詞と同じ。
可能動詞化: 動詞と同じ。
意志性: 動詞と同じ。
派生名詞: 動詞と同じ。

1.6 名詞に関する記載内容

マスタ辞書では、名詞に関して以下のような項目の構文的情報が記載されている。

(1) 副詞的用法
(2) にと
(3) 連体法
(4) 格助詞
(5) 複合性
(6) 接頭辞
(7) 接尾辞
(8) 助数詞
(9) 畳語用法
(10) 可算性
(11) 形容動詞用法
(12) サ変動詞用法

1. 副詞的用法

当該の名詞が副詞として用いられることがあるかどうか、また用いられるときには、その名詞がどのような連体修飾を受けるかが、次の4つの属性値からひとつ選んで記載されている。

なし	：副詞とならない名詞。	
単独	：必ず単独で副詞となる名詞。	(例) 当然 がさがさ 本日
被修飾	: 必ず修飾を受けて副詞となる名詞。	(例) とき あいだ 程 早々
両方	：単独でも、修飾を受けても副詞となる名詞。	(例) 昔 将来 早朝

2. にと

名詞の中にタル活用の形容動詞や、名詞の働きをもつ副詞を含めているので、『する』や『なる』に前接するときの語尾を明示する必要がある。また、ダ型形容動詞の中にも、『有能な』や『無名な』のように「～に」とならない単語があるので、これらも明示する必要がある。そのため、当該の名詞に連用修飾の用法がある場合、次の3つの属性値のひとつを選んで記載している。

にと	: 「～に」と「～と」の両方の形が可能な名詞。	(例) 早々 自然
に	：「～に」の形のみ可能な名詞。	(例) 奇麗 早朝
と	：「～と」の形のみ可能な名詞。	(例) 歴然 堂々

なお、単独で副詞的に働けるかどうかは、副詞的用法の記載に依存している。

3. 連体法

当該の名詞が連体修飾に用いられる場合、その形態について、次の5つの属性値のうちからひとつを選んで記載している。

なの	: 「～な」と「～の」の両方の形が可能な名詞。	(例) 同様 当然 単独
な	：「～な」の形のみ可能な名詞。	(例) 完全 有能 満足
たる	: 「～たる」の形のみ可能な名詞。	(例) 歴然 堂々
の	：「～の」の形のみ可能な名詞。	(例) 私 本日 将来
なし	: 上記のような連体修飾がない名詞。	(例) 同じ 国際 学際

4. 格助詞

次にあげるような名詞は、格助詞を後接しない。

形容動詞的用法しかない名詞	:	(例) 静か 穏やか
連体詞的働きの強い名詞	:	(例) 具体
副詞的働きの強い名詞	:	(例) 当然
格助詞がつくと変化する名詞	:	(例) 新入 → 新入り

1.6. 名詞に関する記載内容

これらの名詞を他の名詞から識別するため、格助詞が後接するかしないかを、つく／つかないの２値の属性値によって登録している。

5. 複合性

連体詞的働きの強い名詞は、名詞連接形しかない。副詞的働きの強い名詞や、「的」が後接して形容動詞的用法をもつ名詞は、連接形がない。これらの情報を次の３つの属性値からひとつ選んで記載している。

必須: 名詞連接形しかない。
可　: 名詞連接形をつくることができる。
不可: 名詞連接形をつくらない。

6. 接頭辞

当該の名詞に前接することが可能な接頭辞を、次の中から選び、それらを並べて記載している。

お　ご　大(だい、おお)　小　非　不　無

7. 接尾辞

当該の名詞に後接することが可能な接尾辞を、次の中から選び、それらを並べて記載している。

さ　的　性　ら　達　供

8. 助数詞

当該の名詞に対して最も一般的に使用される助数詞を、ひとつだけ記載している。文生成において、助数詞を決める手がかりがない場合、ここに記載されているものを使用することができる。

(例)　個　本　匹　件

9. 畳語用法

日本語では、多くの名詞は、単一でも複数の要素をもつ集合を表すことができる。したがって、名詞の畳語表現は、通常その集合が大きいことを表している。名詞の畳語用法は、「多くの～」または「全部の～」と解釈することにする。

畳語名詞を格にとった用言は、反復相と解釈されやすい。しかし、反復の意味合いの強い『一筆一筆』なども「文字を書くときの全部の一筆」と解釈し、これに続く用言を反復相と解釈することにより処理を行うことができる。

マスタ辞書には、当該の名詞が上記の解釈に適合する畳語をもつとき、その読みを記載している。ただし、畳語用法が名詞以外の品詞となる『楽々』、『辱々(しい)』は、記載していない。

10. 可算性

日本語では、名詞が数的であるか量的であるかによって、語形が変化するなどの現象は少ない。しかし、量的な名詞に『数多くの』のような加算の意味をもつ連体修飾の単語がかかることはない。そのため、マスタ辞書では、当該名詞の加算性を、次の４つの属性値からひとつ選んで記載している。

数　：1個、2個と数えられるものを指示する名詞。

量　：量的な対象を指示する名詞。

唯一物：唯一物を指示する名詞。

不明　：上記以外の名詞。

11. 形容動詞用法

連体法の項に「なの」または「な」と記載されている名詞は、形容動詞用法が可能である。このように、当該の名詞に形容動詞用法がある場合、マスタ辞書には形容動詞用法にかかわる格情報およびシソーラスコードを記載している。

12. サ変動詞用法

当該の名詞が「～する」の形でサ変動詞となる場合には、下記の項目が記載されている。

活用形：次の属性値のいずれかを記載する。

ヲ可サ変　：「～する」と「～をする」の両方の形が可能な名詞　(例)　恋

ヲ不可サ変：「～する」の形のみ可能な名詞　(例)　愛　がさがさ

可能用法：次の属性値のいずれかを記載する。

ガ可：「～できる」と「～ができる」の両方の形が可能な名詞　(例)　研究　運転

可　：「～できる」の形のみ可能な名詞　(例)　感動　結晶

不可：どちらの形も不可能な名詞　(例)　影響　成功

態　：動詞と同じ。

相　：動詞と同じ。

自他：動詞と同じ。

可能動詞化：動詞と同じ。

意志性：動詞と同じ。

派生名詞：動詞と同じ。

1.6. 名詞に関する記載内容

第 2 章

LTB マスタ辞書の構造とフォーマット

2.1 マスタ辞書の構造

LTB マスタ辞書は、見出し語ファイルと語義ファイルの 2 つのファイルから構成されている。前者は見出し語レコードから構成されており、後者は 語義レコードから構成されている。

ひとつの単語についての情報は、ひとつの見出し語レコードと、その見出し語レコードが参照するひとつ以上の語義レコードによって記述されている。見出し語に対応する語義がひとつだけのときは語義レコードもひとつだが、いわゆる多義語の場合には、複数の語義レコードが用意されている。

見出し語レコード、語義レコードは、いずれも、属性ラベルと属性値 の対が、リストに並んだ形をしている。品詞によっては、属性値が、さらに属性ラベルと属性値の対のリストになっている場合がある。また、いくつかの属性値に関してはデフォルト値 (既定値) が定められており、記載が省略されているときにはデフォルト値が処理の対象として仮定される。

見出し語レコードと語義レコードの構造と記載項目、およびそれぞれの属性値の定義は、2.2 節および 2.3 節において詳述する。このフォーマットに従ったマスタ辞書の記載例は、2.4 節において紹介する。

本節では、見出し語レコードの各項目の記載内容について、若干の解説を加える。見出し語レコードには、以下のような項目が記載されている。

(1) ID
(2) 品詞
(3) 活用型
(4) 表層表現
(5) 読み
(6) 分解
(7) 語義指標

1. ID

見出し語レコードを参照するための 8 桁の一連番号である。現在のバージョンでは、 2 桁の品詞識別子と、6 桁の通し番号をつないだものが記載されている。

2. 品詞

当該の単語の品詞が、マスタ辞書の文法に準拠して記載されている (第 1 章を参照)。

3. 活用型

当該の単語が用言の場合、マスタ辞書の文法に準拠した活用型が記載されている (第1章を参照)。

4. 表層表現

表層の記述形式が記載されている。当該の単語が用言の場合は、語幹と終止形語尾を「・」でつないだものが登録されている。読みが同じで送り仮名が異なる異表記がある場合には、一般的なものから順に並べて記載される。文生成部においては、選択の指示がなければ先頭の表記が用いられる。

(例) 表層表現／明る・い

5. 読み

読みは、すべてカタカナで表記されている。一つの表層表現に対して複数の読みが可能な場合には、それらが一般的なものから順に並べて記載される。

6. 分解

当該の見出し語が熟語や慣用句の場合、それを構成する単語に分解できることが多い。このような見出し語に対しては、修飾句の挿入可能性を示すために、その分解式が記載されている。

(例) 分解／［口 (名詞)+ を (助詞)+ 利く (動詞)］

7. 語義指標

当該の見出し語レコードが参照している語義レコードのIDが、リストの形で記載されている。これは、見出し語ファイルから語義ファイルへはられたポインタの役割を果たすものである。現在のバージョンでは、語義レコードのIDは、2桁の品詞識別子と6桁の通し番号、および1桁のアルファベットをつないだものである。

2.2 見出し語レコードのフォーマット

見出し語レコードのフォーマットは、以下に示す通りである。ただし、当該の項目が記述されていない場合のデフォルト値は、次のように定められる。

* ……当該スロットが存在しない
$ …… null リスト

〈見出し語レコード〉 : : =
〈識別番号／　〈見出し語レコードID〉,
品詞／　〈品詞〉,
* 活用型／　〈活用型〉,
表層表現／　〈表層表現〉,
読み／　〈読みリスト〉,
* 分解／　〈分解式リスト〉,
語義指標／　〈語義レコードIDリスト〉}.

フォーマット中のそれぞれの属性値の定義は、以下に示す通りである。ただし、ゴチック体は終端記号を示している。

〈見出しレコードID〉 :: = 〈アトム〉 (品詞識別子2桁+通し番号6桁)
〈品詞〉 :: = 動詞 | 名詞 | 形容詞 | 副詞 | 連体詞
〈表層表現〉 :: = 〈語幹〉・〈終止形の活用語尾〉 | 〈アトム〉
〈読みリスト〉 :: = [〈カタカナ表記〉 , ...]
〈分解式リスト〉 :: = 〈分解式〉 | [〈分解式〉 , ...]
〈分解式〉 :: = 〈表層表現〉(〈品詞〉)+...+ 〈表層表現〉(〈品詞〉)
〈活用型〉 :: = 〈動詞の活用型〉 | 〈形容詞の活用型〉
〈動詞の活用型〉 :: = 強変化 | 弱変化 | カ変 | サ変 | 弱サ変 | ある | いく | なさる
〈形容詞の活用型〉 :: = ナ型 | 普通 | よいない
〈語義レコードIDリスト〉 :: = [〈語義レコードID〉 , ...]

〈カタカナ表記〉 :: = 〈アトム〉
〈語幹〉 :: = 〈アトム〉
〈終止形の活用語尾〉 :: = 〈アトム〉
〈語義レコードID〉 :: = 〈アトム〉 (品詞識別子2桁+通し番号6桁+ A〜Z 1桁)
〈アトム〉 :: = 〈ESP のアトム〉

2.3 語義レコードのフォーマット

語義レコードのフォーマットは、以下に示す通りである。ただし、当該の項目が記述されていない場合のデフォルト値は、見出し語レコードと同様に、次のように定められる。

* ……当該スロットが存在しない
$ …… null リスト

〈語義レコード〉 :: =
　〈動詞語義レコード〉
　| 〈普通名詞語義レコード〉
　| 〈形容動詞性名詞語義レコード〉
　| 〈サ変動詞性名詞語義レコード〉
　| 〈形容詞語義レコード〉
　| 〈副詞語義レコード〉
　| 〈連体詞語義レコード〉

〈動詞語義レコード〉 :: =
　{識別番号／　　〈語義レコードID〉,
　 深層格／　　〈深層格列〉,
　 能動態表層格／　〈格対応列〉,
　 * 受動態表層格／　〈格対応列リスト〉,
　 引数／　　〈引数対応列〉,
　 意味構造／　　〈意味構造〉,
　 制約／　　〈制約〉,
　 シソーラスコード／〈シソーラスコード〉,
　 態／
　　[直接受動／　〈ありなし〉,
　　 間接受動／　〈ありなし〉,

使役／　〈ありなし〉，
授受表現／　〈ありなし〉］，
相／　〈相〉，
自他／　〈自他〉，
可能動詞化／　〈可能動詞〉，
意志性／　〈ありなし〉，
$ 派生名詞／　〈派生名詞リスト〉｝．

〈普通名詞語義レコード〉::＝
｛識別番号／　〈語義レコードID〉，
意味構造／　〈意味構造〉，
制約／　〈制約〉，
シソーラスコード／〈シソーラスコード〉，
* 副詞的用法／　〈副詞的用法〉，
* にと／　〈にと1〉，
連体法／　〈連体法〉，
格助詞／　〈つくつかない〉，
複合性／　〈可不可〉，
$ 接頭辞／　〈接頭辞リスト〉，
$ 接尾辞／　〈接尾辞リスト〉，
* 助数詞／　〈助数詞リスト〉，
$ 畳語用法／　〈畳語リスト〉，
可算性／　〈可算性〉｝．

〈形容動詞性名詞語義レコード〉::＝
｛識別番号／　〈語義レコードID〉，
意味構造／　〈意味構造〉，
制約／　〈制約〉，
シソーラスコード／〈シソーラスコード〉，
* 副詞的用法／　〈副詞的用法〉，
* にと／　〈にと1〉，
連体法／　〈連体法〉，
格助詞／　〈つくつかない〉，
形容動詞用法／
［深層格／　〈深層格列〉，
表層格／　〈格対応列リスト〉，
シソーラスコード／〈シソーラスコード〉］，
複合性／　〈可不可〉，
$ 接頭辞／　〈接頭辞リスト〉，
$ 接尾辞／　〈接尾辞リスト〉，
* 助数詞／　〈助数詞リスト〉，
$ 畳語用法／　〈畳語リスト〉，
可算性／　〈可算性〉｝．

〈サ変動詞性名詞語義レコード〉::＝
｛識別番号／　〈語義レコードID〉，
意味構造／　〈意味構造〉，
制約／　〈制約〉，
シソーラスコード／〈シソーラスコード〉，
* 副詞的用法／　〈副詞的用法〉，

```
    *にと／              〈にと1〉，
     連体法／            〈連体法〉，
     格助詞／            〈つくつかない〉，
     サ変動詞用法／
        ［活用形／       〈ヲ可不可〉，
          可能用法／     〈可能用法〉，
          態／
             ［直接受動／〈ありなし〉，
               間接受動／〈ありなし〉，
               使役／    〈ありなし〉，
               授受表現／〈ありなし〉］，
          深層格／       〈深層格列〉，
          能動態表層格／〈格対応列〉，
        * 受動態表層格／〈格対応列リスト〉，
          引数／         〈引数対応列〉，
          相／           〈相〉，
          自他／         〈自他〉，
          可能動詞化／〈可能動詞〉，
          意志性／       〈ありなし〉，
        $ 派生名詞／     〈派生名詞リスト〉，
          シソーラスコード／〈シソーラスコード〉］，
     複合性／            〈可不可〉，
   $ 接頭辞／            〈接頭辞リスト〉，
   $ 接尾辞／            〈接尾辞リスト〉，
   * 助数詞／            〈助数詞リスト〉，
   $ 畳語用法／          〈畳語リスト〉，
     可算性／            〈可算性〉｝．

〈形容詞語義レコード〉::＝
        ｛識別番号／      〈語義レコードID〉，
         深層格／         〈深層格列〉，
         表層格／         〈格対応列〉，
         引数／           〈引数対応列〉，
         意味構造／       〈意味構造〉，
         制約／           〈制約〉，
         シソーラスコード／〈シソーラスコード〉，
         形容動詞用法／   〈ありなし〉，
       $ 接尾辞／         〈接尾辞リスト〉，
       $ 畳語用法／
           ［読み／       〈カタカナ表記〉，
             にと／       〈にと2〉，
             サ変動詞用法／｛副詞と同じ｝］｝．

〈副詞語義レコード〉::＝
        ｛識別番号／      〈語義レコードID〉，
         意味構造／       〈意味構造〉，
         制約／           〈制約〉，
         細分類コード／   〈細分類コード〉，
         にと／           〈にと2〉，
       * サ変動詞用法／
```

```
[活用形／      〈ト可不可〉，
 態／
    [直接受動／〈ありなし〉，
     間接受動／〈ありなし〉，
     使役／     〈ありなし〉，
     授受表現／〈ありなし〉]，
 深層格／      〈深層格列〉，
 能動態表層格／〈格対応列〉，
 • 受動態表層格／〈格対応列リスト〉，
 引数／        〈引数対応列〉，
 相／          〈相〉，
 自他／        〈自他〉，
 可能動詞化／〈可能動詞〉，
 意志性／      〈ありなし〉，
 $ 派生名詞／  〈派生名詞リスト〉，
  シソーラスコード／〈シソーラスコード〉]}．

〈連体詞語義レコード〉::＝
      {識別番号／        〈語義レコードID〉，
       意味構造／        〈意味構造〉，
       制約／            〈制約〉}．
```

フォーマット中のそれぞれの属性値の定義は、以下の通りである。ただし、ゴチック体は終端記号を示している。

〈語義レコードID〉::＝〈アトム〉 (品詞識別子２桁＋通し番号６桁＋ A～Z １桁)
〈語義内容〉::＝〈語義内容の説明〉
〈意味構造〉::＝〈意味構造の記号的表現〉
〈制約〉::＝〈意味構造選択・格要素カテゴリ検査の規則〉
〈シソーラスコード〉::＝〈タ－ム〉

〈相〉::＝ **状態**｜**準状態**｜**瞬間(続ケル可)**｜**瞬間(続ケル不可)**
　　｜**継続(終ワル可)**｜**継続(終ワル不可)**
〈自他〉::＝ **絶対自動**｜**絶対他動**｜**相対自動**｜**相対他動**｜**両用**
〈可能動詞〉::＝ **可**｜**不可**｜**可能動詞**

〈可不可〉::＝ **可**｜**不可**
〈ありなし〉::＝ **あり**｜**なし**
〈つくつかない〉::＝ **つく**｜**つかない**
〈ヲ可不可〉::＝ **ヲ可サ変**｜**ヲ不可サ変**
〈可能用法〉::＝ **ガ可**｜**可**｜**不可**
〈ト可不可〉::＝ **ト可サ変**｜**ト不可サ変**

〈深層格列〉::＝(〈深層格名〉，...)
〈格対応列リスト〉::＝[〈格対応列〉，...]
〈格対応列〉::＝(〈深層格名〉：〈表層格列〉，...)
〈引数対応列〉::＝(〈深層格名〉：変数列，...)
〈表層格列〉::＝〈表層格名〉∨...
〈変数列〉::＝〈変数名〉，...

〈派生名詞リスト〉::＝[〈表層表現〉，...]

〈副詞的用法〉 :: = 単独 | 被修飾 | 両方
〈連体法〉 :: = なの | な | たる | なし
〈接頭辞リスト〉 :: = [〈接頭辞, ...]
〈接尾辞リスト〉 :: = [〈接尾辞〉, ...]
〈畳語リスト〉 :: = [〈カタカナ表記〉, ...]
〈可算性〉 :: = 数 | 量 | 唯一物 | 不明
〈細分類〉 :: = 〈マスタ辞書の文法に準拠した細分類〉
〈にと 1〉 :: = にと | に | と
〈にと 2〉 :: = にと | に | と | に要 | と要 | 単独

〈深層格名〉 :: = 〈アトム〉
〈表層格名〉 :: = 〈アトム〉
〈数字〉 :: = 〈アトム〉
〈助数詞〉 :: = 〈アトム〉
〈接頭辞〉 :: = 〈アトム〉
〈接尾辞〉 :: = 〈アトム〉
〈形容詞〉 :: = 〈アトム〉

〈アトム〉 :: = 〈ESP のアトム〉
〈変数名〉 :: = 〈ESP の変数名〉

2.4 マスタ辞書の記載例

2.4.1 動詞の記載例

マスタ辞書の動詞の記載例を以下に示す。
例としてあげたのは、『与える』と『集まる』の 2 つの動詞である。

【見出し語レコード】

```
 {
  識別番号／動詞 000080,
  品詞／動詞,
  活用型／弱変化,
  表層表現／与え・る,
  読み／［アタエル］,
  語義指標／［動詞 000080A］
 }.
 {
  識別番号／動詞 000100,
  品詞／動詞,
  活用型／強変化,
  表層表現／集ま・る,
  読み／［アツマル］,
  語義指標／［動詞 000100A, 動詞 000100B］
 }.
```

【語義レコード】

```
 {
  識別番号／動詞 000080A,
  深層格／(agt, goa, obj),
  能動表層格／(agt：が, goa：に, obj：を),
  受動表層格／[(goa：が, agt：にＶによってＶから, obj：を),
                (obj：が, agt：によってＶから, goa：にＶへ)],
  引数／(agt： A, goa： B, obj： X),
  意味構造／sem,
  制約／[],
  シソーラスコード／2,
  態／[
       直接受動／あり,
       間接受動／あり,
       使役／あり,
       授受表現／あり],
  相／[瞬間],
  自他／[絶対他動],
  可能動詞化／不可,
  意志性／あり,
  派生名詞／[]
 }.
 {
  識別番号／動詞 000100A,
  深層格／(obj),
  能動表層格／(obj：が),
  引数／(obj： A),
  意味構造／sem,
  制約／[],
  シソーラスコード／2,
  態／[
       直接受動／なし,
       間接受動／なし,
       使役／なし,
       授受表現／なし],
  相／[瞬間],
  自他／[相対自動],
  可能動詞化／不可,
  意志性／なし,
  派生名詞／[集まり]
 }.
 {
  識別番号／動詞 000100B,
  深層格／(agt),
  能動表層格／(agt：が),
  引数／(agt： A),
  意味構造／sem,
  制約／[],
  シソーラスコード／2,
  態／[
       直接受動／なし,
```

```
        間接受動／あり,
        使役／なし,
        授受表現／なし］,
    相／［瞬間］,
    自他／［相対自動］,
    可能動詞化／不可,
    意志性／あり,
    派生名詞／［集まり］
  }.
```

2.4.2 普通名詞の記載例

マスタ辞書の普通名詞の記載例を以下に示す。
例としてあげたのは、『空気』、『茎』、『草』の3つの名詞である。

【見出し語レコード】

```
  {
   識別番号／名普 001200,
   品詞／名詞,
   表層表現／空気,
   読み／［クウキ］,
   語義指標／［名普 001200A］
  }.
  {
   識別番号／名普 001210,
   品詞／名詞,
   表層表現／茎,
   読み／［クキ］,
   語義指標／［名普 001210A］
  }.
  {
   識別番号／名普 001220,
   品詞／名詞,
   表層表現／草,
   読み／［クサ］,
   語義指標／［名普 001220A］
  }.
```

【語義レコード】

```
  {
   識別番号／名普 001200A,
   意味構造／ sem,
   制約／［］,
   シソーラスコード／ 1,
   連体法／の,
   格助詞／つく,
   複合性／可,
   接頭辞／［］,
```

```
  接尾辞／[],
  畳語用法／[],
  可算性／不明
},
{
  識別番号／名普 001210A,
  意味構造／sem,
  制約／[],
  シソーラスコード／1,
  連体法／の,
  格助詞／つく,
  複合性／可,
  接頭辞／[],
  接尾辞／[],
  畳語用法／[],
  可算性／数
},
{
  識別番号／名普 001220A,
  意味構造／sem,
  制約／[],
  シソーラスコード／1,
  連体法／の,
  格助詞／つく,
  複合性／可,
  接頭辞／[],
  接尾辞／[],
  畳語用法／[],
  可算性／不明
},
```

2.4.3 形容動詞性名詞の記載例

マスタ辞書の形容動詞性名詞の記載例を以下に示す。
例としてあげたのは、『強力』、『首つ引き』、『広大』の3つの名詞である。

【見出し語レコード】

```
{
  識別番号／名形 000100,
  品詞／名詞,
  表層表現／強力,
  読み／[キョウリョク],
  語義指標／[名形 000100A]
},
{
  識別番号／名形 000110,
  品詞／名詞,
  表層表現／首っ引き,
  読み／[クビッピキ],
```

```
    語義指標／［名形 000110A］
  }．
  {
    識別番号／名形 000120，
    品詞／名詞，
    表層表現／広大，
    読み／［コウダイ］，
    語義指標／［名形 000120A］
  }．
```

【語義レコード】

```
  {
    識別番号／名形 000100A，
    意味構造／ sem，
    制約／［］，
    シソーラスコード／3，
    にと／に，
    連体法／な，
    格助詞／つかない，
    形容動詞用法／［
        深層格／［ods］，
        表層格／（ods：が），
        引数／（ods： X），
        シソーラスコード／3］，
    複合性／可，
    接頭辞／［］，
    接尾辞／［さ］，
    畳語用法／［］，
    可算性／不明
  }．
  {
    識別番号／名形 000110A，
    意味構造／ sem，
    制約／［］，
    シソーラスコード／3，
    にと／に，
    連体法／の，
    格助詞／つかない，
    形容動詞用法／［
        深層格／［ods， obj］，
        表層格／（ods：が， obj：と），
        引数／（ods： X， obj：Y），
        シソーラスコード／3］，
    複合性／可，
    接頭辞／［］，
    接尾辞／［］，
    畳語用法／［］，
    可算性／不明
  }．
```

```
{
 識別番号／名形 000120A,
 意味構造／ sem,
 制約／ [] ,
 シソーラスコード／ 3,
 にと／に,
 連体法／な,
 格助詞／つかない,
 形容動詞用法／ [
      深層格／ [ods] ,
      表層格／ (ods : が),
      引数／ (ods : X),
      シソーラスコード／ 3] ,
 複合性／不可,
 接頭辞／ [] ,
 接尾辞／ [き] ,
 畳語用法／ [] ,
 可算性／不明
} .
```

2.4.4 サ変動詞性名詞の記載例

マスタ辞書のサ変動詞性名詞の記載例を以下に示す。
例としてあげたのは、『加工』、『関係』の２つの名詞である。

【見出し語レコード】

```
{
 識別番号／名サ 000080,
 品詞／名詞,
 表層表現／加工,
 読み／ [カコウ] ,
 語義指標／ [名サ 000080A]
} .
{
 識別番号／名サ 000090,
 品詞／名詞,
 表層表現／関係,
 読み／ [カンケイ] ,
 語義指標／ [名サ 000090A, 名サ 000090B]
} .
```

【語義レコード】

```
{
 識別番号／名サ 000080A,
 意味構造／ sem,
 制約／ [] ,
 シソーラスコード／ 1,
```

```
  連体法／の,
  格助詞／つく,
  サ変動詞用法／［
      活用形／ヲ可サ変,
      可能用法／ガ可,
      態／［
          直接受動／あり,
          間接受動／あり,
          使役／あり,
          授受表現／あり］,
      深層格／［agt, obj］,
      能動表層格／(agt：が, obj：を),
      受動表層格／［(obj：が, agt：に∨によって)］,
      引数／(agt：X, obj：Y),
      相／［継続, 瞬間］,
      自他／絶対他動,
      可能動詞化／不可,
      意志性／あり,
      派生名詞／［］,
      シソーラスコード／2］,
  複合性／可,
  接頭辞／［］,
  接尾辞／［件］,
  慣語用法／［］,
  可算性／不明
}.
 {
  識別番号／名サ 000090A,
  意味構造／sem,
  制約／［］,
  シソーラスコード／1,
  連体法／の,
  格助詞／つく,
  サ変動詞用法／［
      活用形／ヲ不可サ変,
      可能用法／ガ可,
      態／［
          直接受動／なし,
          間接受動／なし,
          使役／なし,
          授受表現／なし］,
      深層格／［obj, par］,
      能動表層格／(obj：が, par：と∨に),
      引数／(obj：X, par：Y),
      相／［準状態］,
      自他／絶対自動,
      可能動詞化／不可,
      意志性／なし,
      派生名詞／［］,
      シソーラスコード／2］,
```

```
    複合性／可,
    接頭辞／［ど，無］,
    接尾辞／［性］,
    畳語用法／［］,
    可算性／不明
  }.
  {
    識別番号／名サ 000090B,
    意味構造／ sem,
    制約／［］,
    シソーラスコード／ 1,
    連体法／の,
    格助詞／つく,
    サ変動詞用法／［
        活用形／ヲ不可サ変,
        可能用法／ガ可,
        態／［
            直接受動／なし,
            間接受動／あり,
            使役／あり,
            授受表現／あり］,
        深層格／［agt，par］,
        能動表層格／（agt：が，par：とヘに），
        引数／（agt：X，par：Y），
        相／［継続，瞬間］,
        自他／絶対自動,
        可能動詞化／不可,
        意志性／あり,
        派生名詞／［］,
        シソーラスコード／2］,
    複合性／可,
    接頭辞／［ど，無］,
    接尾辞／［性］,
    畳語用法／［］,
    可算性／不明
  }.
```

2.4.5 形容詞の記載例

マスタ辞書の形容詞の記載例を以下に示す。
例としてあげたのは、『明るい』、『新しい』の2つの形容詞である。

【見出し語レコード】

```
  {
    識別番号／形容 000010,
    品詞／形容詞,
    活用型／普通,
```

```
  表層表現／明る・い,
  読み／［アカルイ］,
  語義指標／［形容 000010A］
 }.
 {
  識別番号／形容 000020,
  品詞／形容詞,
  活用型／普通,
  表層表現／新し・い,
  読み／［アタラシイ］,
  語義指標／［形容 000020A, 形容 000020B, 形容 000020C］
 }.
```

【語義レコード】

```
 {
  識別番号／形容 000010A,
  深層格／(ods, obj),
  表層格／［(ods：が, obj：に)］,
  引数／(ods： X, obj： Y),
  意味構造／sem,
  制約／［］,
  シソーラスコード／3,
  形容動詞用法／なし,
  接尾辞／［む, み］,
  畳語用法／［］
 }.
 {
  識別番号／形容 000020A,
  深層格／(ods),
  表層格／［(ods：が)］,
  引数／(ods： X),
  意味構造／sem,
  制約／［］,
  シソーラスコード／3,
  形容動詞用法／なし,
  接尾辞／［げ, め］,
  畳語用法／［］
 }.
 {
  識別番号／形容 000020B,
  深層格／(ods),
  表層格／［(ods：が)］,
  引数／(ods： X),
  意味構造／sem,
  制約／［］,
  シソーラスコード／3,
  形容動詞用法／なし,
  接尾辞／［がる, げ, め］,
  畳語用法／［］
```

```
}.
 {
  識別番号／形容 000020C,
  深層格／(ods),
  表層格／[(ods：が)],
  引数／(ods： X),
  意味構造／sem,
  制約／[],
  シソーラスコード／3,
  形容動詞用法／なし,
  接尾辞／[],
  量語用法／[]
}.
```

2.4.6 副詞の記載例

マスタ辞書の副詞の記載例を以下に示す。
例としてあげたのは、『久し振り』、『むやみ』、『折りよく』の３つの副詞である。

【見出し語レコード】

```
 {
  識別番号／副詞 000900,
  品詞／副詞,
  表層表現／久し振り,
  読み／[ヒサシブリ],
  語義指標／[副詞 000900A]
}.
 {
  識別番号／副詞 000910,
  品詞／副詞,
  表層表現／むやみ,
  読み／[ムヤミ],
  語義指標／[副詞 000910A]
}.
 {
  識別番号／副詞 000920,
  品詞／副詞,
  表層表現／折よく,
  読み／[オリヨク],
  語義指標／[副詞 000920A]
}.
```

【語義レコード】

```
 {
  識別番号／副詞 000900A,
```

```
    意味構造／ sem,
    制約／ [] ,
    細分類／ 1,
    にと／に
  } .
  {
    識別番号／副詞 000910A,
    意味構造／ sem,
    制約／ [] ,
    細分類／ 1,
    にと／に
  } .
  {
    識別番号／副詞 000920A,
    意味構造／ sem,
    制約／ [] ,
    細分類／ 1,
    にと／単独
  } .
```

2.4.7 連体詞の記載例

マスタ辞書の連体詞の記載例を以下に示す。
例としてあげたのは、『あくる』、『ある』、『同じ』の3つの連体詞である。

【見出し語レコード】

```
  {
    識別番号／連体 000010,
    品詞／連体詞,
    表層表現／あくる,
    読み／［アクル］,
    語義指標／［連体 000010A］
  } .
  {
    識別番号／連体 000020,
    品詞／連体詞,
    表層表現／ある,
    読み／［アル］,
    語義指標／［連体 000020A］
  } .
  {
    識別番号／連体 000030,
    品詞／連体詞,
    表層表現／同じ,
    読み／［オナジ］,
    語義指標／［連体 000030A］
  } .
```

【語義レコード】

```
 {
  識別番号／連体 000010A,
  意味構造／ sem,
  制約／ []
}.
 {
  識別番号／連体 000020A,
  意味構造／ sem,
  制約／ []
}.
 {
  識別番号／連体 000030A,
  意味構造／ sem,
  制約／ []
}.
```

第3章

LTB マスタ辞書のアクセスメソッド

3.1 アクセスメソッドの使用方法

第1章、第2章ではマスタ辞書の記載内容とフォーマットについて説明した。このマスタ辞書に対しては、その記載内容を検索するための**アクセスメソッド** が、PSI 上に用意されている。本章では、このアクセスメソッドの概要と、インストールの方法、および使用方法について解説する。

3.1.1 アクセスメソッドの概要

はじめに、アクセスメソッドの処理の概要について説明する。 アクセスメソッドの処理の流れを簡単に示すと、次の通りである。

マスタ辞書 (ファイル) → (トランスレータ) → ESPメソッド (ファイル) → (マニピュレータ) → 辞書内容

まず、あらかじめ想定された構文 (次項を参照) に従って、マスタ辞書のテキスト・ファイルを作成する。次に、このファイルを本アクセスメソッドが提供するトランスレータを用いて ESP メソッドに変換し、カタログしておく。以降は、本アクセスメソッドが提供するマニピュレータを利用して、辞書の内容を検索することができる。

なお、SIMPOS 提供のライブラリアンは、一定以上の大きさのファイルを一度にカタログできない。このため、変換後の ESP メソッドは複数のファイルにわたって出力されることに注意すること。

3.1.2 マスタ辞書の構文

トランスレータの入力となるマスタ辞書のファイルは、次の構文に従って記述されていなければならない。

〈マスタ辞書〉 ::= 〈レコード〉 ...
〈レコード〉 ::= 〈部分項表現〉 .
〈部分項表現〉 ::= { 〈属性リスト〉 }
〈属性リスト〉 ::= 〈属性〉 , ...
〈属性〉 ::= 〈属性名〉 / 〈属性値〉
〈属性名〉 ::= 〈ESP アトム〉
〈属性値〉 ::= 〈ESP タ ー ム〉 | 〈部分項表現〉

2章で示した辞書フォーマットは、すべて上記の構文を満たしている。

3.1.3 トランスレータ

トランスレータを利用する際は、まず SIMPOS ライブラリアンを起動し、クラス **dict##trans** をロードする。次に SIMPOS デバッガを起動し、以下の手順でマスタ辞書の変換をおこなう。

まず、デバッガから次のようにキー入力し、トランスレータのオブジェクトを生成する。

```
?- :create(#dict##trans, Obj).
```

これを実行すると、変換の途中経過を表示するためのウィンドウの位置を決めるため、プロンプトが表示されるので、これに対して応答する。応答は、マウス左クリックで行う。

次に、デバッガから次のようにキー入力して、マスタ辞書を ESP メソッドに変換し、カタログ、セーブする。

```
?- :translate_and_catalogue(Obj, 〈マスタ辞書のファイル名のリスト〉, C).
```

変数 C には、変換したレコードの個数がユニファイされる。

現バージョンでは、見出しテーブルと語義テーブルとの識別は、レコードの内容を見て自動的におこなっている。見出しテーブルにはテーブル名 "midasi"、語義テーブルにはテーブル名 "gogi" を自動的に割り当てるようになっている。

マスタ辞書の部分部分をインクリメンタルに変換する機能は、現バージョンではサポートしていない。

また、create/2 を実行する前には、デバッガのパッケージを dict としておかなければならない。

(例) ファイル dic1.p 、 dic2.p に格納されたマスタ辞書を、 ESP メソッドに変換し、さらにカタログ、セーブする。

```
?- :translater_and_catalogue(Obj, ["dic1.p", "dic2.p"], C).
```

3.1.4 マニピュレータ

マニピュレータを利用するには、まずライブラリアンを起動し、クラス dict##manipulatorをロードする。次にデバッガを起動し、以下の手順で辞書の検索を行う。

まず、デバッガから次のようにキー入力し、マニピュレータのオブジェクトを生成する。

```
?- :new(#dict##manipulator, Obj).
```

次に、デバッガから次のようにキー入力して、検索を行う。

```
?- :get_entry(Obj, 〈テーブル名〉, 〈検索条件〉, Str).
```

これによって、テーブル名で表されるテーブルのなかから検索条件にあうレコードが検索され、Strとユニファイされる。

検索条件は、『マスタ辞書の構文』で示した〈レコード〉と同じ形式で表現される。ただし、検索はユニフィケーションによって行うので、属性値に ESP の変数が割り当てられていてもよい。

検索条件中の属性リストにおいて、属性の現れる順序はマスタ辞書と同じでなくともかまわない。しかし、マスタ辞書に記述されていない属性を検索条件中に記述した場合には、呼出しは失敗する。

(例) テーブル "midasi" から、品詞が動詞のものを検索し、その表層表現をSに、検索結果であるレコード全体をRにユニファイする。

```
?- :get_entry(Obj, midasi, ''([品詞/動詞, 表層表現/S]), R).
```

なお、メソッド :get_entry/4 を使うと、フェイル・バックトラックにより、次候補の検索を行うことができる。

3.2 アクセスメソッドのインストール方法

アクセスメソッドのインストールは、以下の手順に従って行う。

1. ユーザ "dict" を登録する。
2. ファイル・マニピュレータを用いて、ユーザ "dict" のホーム・ディレクトリ の論理名を "me:" と定義する。
3. ホーム・ディレクトリ下にディレクトリ "work" を作成し、 その論理名を "work:" と定義する。ディレクトリの論理名は、 あらかじめ login.com 内で定義しておくと便利である。
4. ライブラリアンを用いて、パッケージ "dict" を定義する。
5. ライブラリアンを用いて、デフォールトパッケージを "dict" にする。 これも login.com 内に書いておくとよい。
6. アクセスメソッドのすべてのソースファイルを me: へコピーする。
7. ライブラリアンを用いてファイル "me:macro.esp" をカタログし、クラス をセーブする。
8. ライブラリアンですべての ESP ファイル "me:*.esp" をカタログする。こ のとき発生するエラーについては、"Continue..." を選択する。
9. ライブラリアンですべての ESP ファイル "me:*.esp" をカタログして、セー ブする。

ディレクトリ "work:" は、マスタ辞書の変換結果を出力するためのものである。

また、現バージョンのライブラリアンに問題があるためと思われるが、トランスレータによって辞書を変換した後に、一端ファイル "manip.esp" をカタログ、セーブしておかないとマニピュレータが正常に動作しないことがあるので注意すること。

なお、今回のリリースでユーザに提供するマスタ辞書関係のファイルは、すべて SIMPOS のテンプレートファイルである。ユーザに提供するファイル名の一覧を次に示す。

(トランスレータ)

error.template	file_dict.template	file_tmpl.template
file_dict.template	file_pst.template	files_tmpl.template
files_top.template	my_input_file.template	my_io_window.template
my_macro.template	my_operator.template	my_output.template
name.template	puts.template	win_trans.template

(マニピュレータ)

manipulator.template

(サンプル辞書データ)

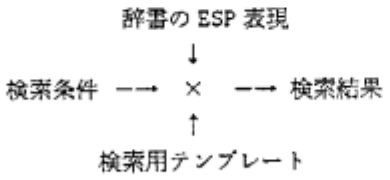
dict.template

dict0.template, dict1.template, dict2.template, dict3.template, ...
template.template
tmpl0.template, tmpl1.template, tmpl2.template, tmpl3.template, ...

3.3 アクセスメソッドの内部構成

3.3.1 検索方式の概要

本アクセスメソッドにおいては、大量の辞書を効率よく検索するために、マスタ辞書の変換結果である ESP メソッドのデータ表現に、以下に説明するような工夫を加えた。



まず、レコードに含まれている各属性をすばやく取り出して検索条件とユニファイするために、検索用テンプレートを用意した。検索用テンプレートとは、レコード中のすべての属性について、その属性値がレコードのどの位置に格納されているかを保持するものであり、次の形式をしている。

```
tmpl(テンプレート ID, 属性パス 1, 属性値を指す変数 1, レコードの形);
tmpl(テンプレート ID, 属性パス 2, 属性値を指す変数 2, レコードの形);
```

「属性パス」とは、部分項表現の外側から目指す属性へのパスのリスト表現をいい、たとえば部分項表現:

[a／[b／[c／C,...] ,...] ,...]

の属性 c を表す属性パスは [c,b,a] である。

また、辞書のレコードは次の形をしている。

```
dict(レコード ID, テーブル名, テンプレート ID, レコード);
```

たとえば、レコードID(=I) が与えられたとき、レコード中のある属性パス P の属性値 Fv を取り出す手順は、次の通りである。

(1) dict(I, _, TID, Record) を呼び出す。
(2) tmpl(TID, P, Fv, Record) を呼び出す。

検索用テンプレートは、マスタ辞書の変換の際にすべてのレコードについて、その中のすべての属性について作成され、記憶される。レコードの形式のバラエティが多くないので、組み合わせ的な数でテンプレートが作られることはない。

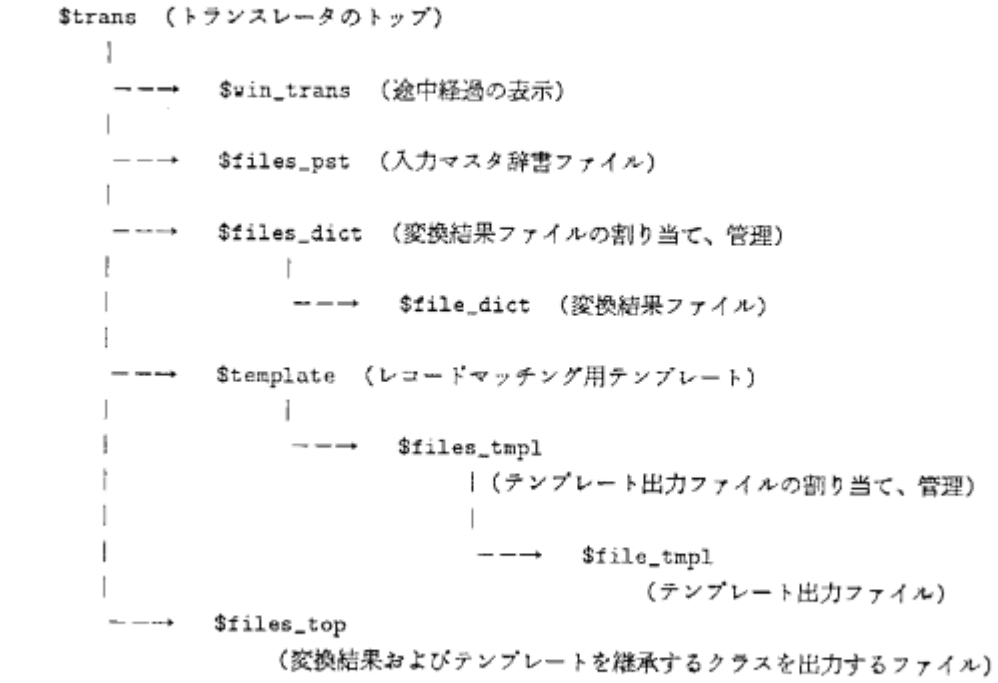
3.3.2 構成

本項では、アクセスメソッドの処理に活躍するクラスおよびオブジェクトの説明をおこなう。

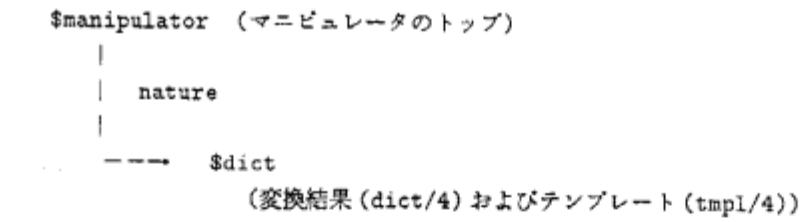
凡例:

#xxx	クラスを表す
$xxx	オブジェクトを表す
⟶	メソッド呼出しを表す

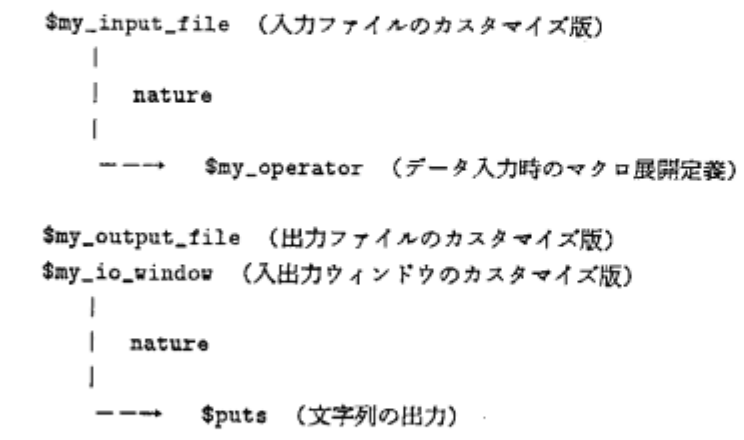
1. トランスレータ



2. マニピュレータ



3. 共通クラスおよびオブジェクト



#name （ファイル名、クラス名の生成）

```
#my_macro （ソースファイルカタログ時のマクロ展開定義）
```

【参考文献】

(1) 瀧塚孝志, et al. "LTB マスター辞書の構成",
ソフトウェア科学会, 論理と自然言語研究会ワークショップ, 1988.

(2) 田中裕一, et al. "LTB マスター辞書の意味記述の構想",
ソフトウェア科学会, 論理と自然言語研究会ワークショップ, 1988.

(3) 田中裕一. "談話理解のための辞書/言語知識ベース",
第五世代コンピュータに関するシンポジウム予稿集, May 1988. p. 15 - 16.

(4) 田中裕一, 海野敏. "LTB マスタ辞書の構造と内容",
情報処理学会第 37 回全国大会, Sep. 1988. p. 1086 - 1087.

(5) 田中裕一, 大島資生, 長澤陽子. "LTB マスタ辞書の意味記述",
情報処理学会第 37 回全国大会, Sep. 1988. p. 1088 - 1089.

(6) Tanaka, Y. and Yoshioka, T. "Overview of the Dictionary and Lexical Knowledge Base Research", *Proceedings of the International Conference on Fifth Generation Computer Systems*, Nov. - Dec. 1988. p. 277 - 284.

LTB 操作説明書

– CIL 編 –

(LTB1.1 版)

ICOT 第二研究室

平成元年 3 月 31 日

本説明書は CIL(Complex Indeterminates based Language) の言語仕様およびその処理系の操作方法について記述している。

- CIL の概要
- 操作仕様
- 言語仕様
- プログラム仕様

汎用日本語処理系 LTB(Language Tool Box) の説明書は次のような構成になっている。

- LTB 概要編
- LTB マスタ辞書編
- CIL 編
- LAX 編
- SAX 編
- 文生成編

目次

図目次

表目次

第 1 章

CIL の概要

本章では CIL の言語の特徴とプログラミング環境の概要について述べる。

1.1 言語の特徴

計算言語学は、形式的にみれば、オブジェクトとそれらの間の制約の対の形で理論が構成されている。CIL (Complex Indeterminates based Language) は、そのような自然言語処理にあらわれる複雑な構造を持つデータ（オブジェクト）、およびそれらの間の制約関係を、より柔軟に記述するために、 Prolog を拡張した言語である。本言語により、GPSG, LFG等の構文論および、状況意味論(situation semantics) 等の意味論をはじめとする自然言語処理システムが容易に記述できる。

論理型言語 Prolog をベースとした言語である、CIL の実行メカニズムの基本は、単一化と後戻りである。

- 単一化 (unification)
- 後戻り (backtracking)

CIL では、新たに二つの基本機能を導入した。

- 部分項(partially specified term)
- 遅延実行制御(lazy evaluation)

これらをもとに幾つかのシンタックス・シュガー（標準マクロ）や組込述語を提供する。

- シンタックス・シュガー (syntax-sugar)
- 組込述語 (built-in predicate)

1.1.1 部分項

部分項の概念

大規模なプログラムの記述には，複雑なデータ構造を表現する機能が不可欠である。自然言語処理システムの場合には、いかに意味・談話構造、知識等を表現し、操作するか、が問題となる。

しかしながら、そのようなデータを表現するために Prolog では、次のような不備な点をもつ複合項(compound term) を提供しているだけで、自然言語処理システムの記述を不必要に困難なものにしていた。

- 引数の位置順序とその意味（役割）を記述者が管理する必要がある。
- いつもすべての引数を書かなければならない。

CIL では、この問題に対し、項の概念を拡張した部分項というデータ構造を導入している。 これは、下に掲げるような計算機言語やデータベース、文法理論等、計算機工学の他の分野で広く見られるデータ構造からの抽象化された概念を、 Prolog 上に実現したものである。

1. 一階述語論理の項

1.1. 言語の特徴

2. データベースにおけるレコード

3. 属性 / 属性値対リスト

4. LISP のプロパティリスト (property list)

5. C.Pollard の anadic relation （状況意味論の一つのバージョン）

6. GPSG のカテゴリ

7. LFG の機能構造 (functional structure)

8. Barwise の状況理論における割り当て (assignment) や事態 (state of affair)

9. Minsky のフレーム (Frame)

部分項は、ラベルと値の対の順序のない集りとして定義される。具体的には、次の形式で記述する。

$$\{a_1/b_1, a_2/b_2, \ldots, a_n/b_n\}$$
$$\text{ここで } n \geq 0, a_i \neq a_j \ (i \neq j)$$

a_i はラベルを表し、b_iは値を表す。 a_i/b_i の出現順序には意味がない。a_iには通常の（変数を含まない）項が、b_i には部分項も含めたあらゆるタイプの項が記述できる。これにより知識を、それが持つ属性の名前を用いて表すことが可能になる。b_iには以下の例に示すように自分自身が記述でき、最近の状況理論 (situation theory) で議論されている自己参照を含むオブジェクトを表現、操作することを可能にする。

$$X = \{a / X\}.$$

部分項はラベル付けされた順序なし木 (labelled unordered tree) とも解釈できる。例えば、{ a/1, b/{c/2,d/3}}は下図のように示される。

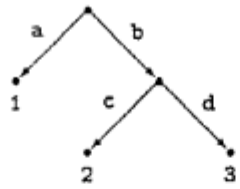


図 1.1: 部分項とラベル付順序なし木

「部分項」という名前は、「項の必要な部分のみを指定する記述法」という面を強調している。それはまた部分項で表現されるデータが、そのオブジェクトが本来持っている構造の一部が明示されたものにすぎず、処理が進むにつれてデータ（引数すなはちラベル－値の対）が追加される事を意味する。このような性質は、部分項が本質的に拡張可能なデータ構造であることにつながる。

部分項で表現されるデータに対する参照、操作は、ラベルa_iを用いて行われ、また、部分項の単一化は Prolog の単一化の仕様を拡張することで実現される。 CIL の拡張された単一化の実質的意味は併合 (merge) となる。これらの機能をプログラミングの際に充分に利用できるように、さまざまな組込述語や記法を CIL では提供する。

さらに部分項を上のように木（グラフ）と考えると、意味ネットワーク(semantic network) の一種の表現とみることもできるが、その場合に必要となる照合操作 (pattern matching) の簡単なものを組込んでいる。

部分項と拡張単一化

部分項の記法を以下に示す。ここで，一般項とは部分項を含む項のことであり，通常項とは部分項を含まない項のことである。

```
<部分項>          ::=  "{" <要素並び>  "}"  |  "{}"
<要素並び>        ::=  <部分項の要素> { "," <部分項の要素> }  …
<部分項の要素>    ::=  <ラベル> "/" <値>
<ラベル>          ::=  <通常項>
<値>              ::=  <一般項>
```

［例］

- { a/1, f(1)/X, c/{d/g(Y)}, c/{d/h(Z)} }
- { 関係 / 愛する,
  主格 / { 名前 / 健,
  年齢 / 25 },
  対格 / { 名前 / 奈緒美,
  年齢 / 24 } }

ただし、同じラベルを持つ矛盾した要素が同一レベルに存在してはいけない。つまり、

$$X = \{ a/1, b/2, a/2 \}.$$

のような部分項は生成できない。

部分項の導入に伴い、単一化の仕様は、次のように拡張されている。この拡張単一化の実質的意味は、”マージ（併合）”である。

1. 部分項が関与しない単一化（通常項同士の単一化）
   KL0 の unify 述語が実行する。したがって、その仕様は Prolog の単一 化と同じである。

```
f(X) = f(a)          ==>        X = a.
g(b, a) = g(a, b)               失敗
```

2. 部分項と通常項の単一化

   (a) 部分項と未定義変数の単一化
   単一化はつねに成功する。未定義変数は相手の部分項と同じになる。

```
h(X) = h({x/a, y/b})      ==>      X = {x/a, y/b}
```

   (b) 部分項と未定義変数以外の単一化
   単一化はつねに失敗する。要素に変化はない。

```
{f/p, a1/x, a2/y} = p(x,y)       失敗
```

3. 部分項同士の単一化

   (a) 二つの部分項ともに存在するラベルに関しては，ラベルに対応するそれ ぞれの値の単一化（この単一化もここで述べている仕様に従って行われる）を行う。

```
{a/{c/1}, b/X} = {b/2, a/Y}      ==>      X = 2, Y = {c/1}
{a/{b/X}} = {a/{b/2}}            ==>      X = 2
{a/1} = {a/2}                             失敗
```

   (b) 一方の部分項にしか存在しないラベルに関しては，そのラベルと値から なる対を、新たに、他方の部分項に追加する。

```
X = {a/1}, Y = {b/1}, X = Y              ==> X = Y = {a/1, b/1}
X = {x/1, y/1}, Y = {z/1, x/1}, X = Y    ==> X = Y = {x/1, y/1, z/1}
```

部分項用シンタックス・シュガー

部分項の操作を簡単に記述するために，以下の諸記法が提供されている。

1. 部分項
   部分項は，内部的に x(_,_) という通常項で実現されている。中括弧で囲まれた要 素列のすべてが X/Y の形式のとき，部分項として扱われる。

   ..., p({a/1,b/2}), ... は
   ..., mk_assoc((a/1,b/2), X), p(X), ... に展開される。

2. 拡張単一化
   部分項を含む可能性のある項の単一化は， CIL 組込述語の unify_cil/2 に展開 される。

   ..., X = Y, ... は
   ..., unify_cil(X,Y), ... に展開される。

3. 部分項のラベル指定による値の参照記法
   「部分項 P のラベル L に置かれている値」を P!L と書く。 P!L は、組込述語 role(L,P,V) に展開され、 V の値を持つ。

   ... f(X!a), ... は
   ... role(a, X, V), f(V), ... に展開される。

   [例]
   • {a/1, b/X}!b = 3. は、 X = 3 となる。
   • X = {a/1, b/2}, X!c = 3. は、 X = {a/1, b/2, c/3} となる。

4. 条件付項
   「条件 C を満たす X」を X:C と書く。 条件 C として述語が記述できる項記述となっている。
   [例]
   T = X:father_of(X, 太郎).
   これは「太郎の父（である X）を T と単一化しなさい」という意味である。

5. 同格記法
   X#Y は、 X:(X=Y) の略記である。 X は Y と同格の項であることを示す。
   [例]
   X# {a/1, b/X!a} = Y. は、 X = Y = {a/1, b/1} となる。

1.1.2 遅延実行制御

遅延実行制御の概念

CIL の遅延実行制御のオリジナルは、 Prolog-II(Colmerauer) のフリーズ制御である。 PSI でもバインドフック (bind_hook) と呼ばれる述語で提供される。これらは「ある変数に対し、その変数が具体化された時点で、特定の述語 (デーモン) を実行するように指定できる機能」を持つ述語である。

遅延実行制御を用いると、以下のようなプログラミングでは Generate and Test) を Test and Generate とすることで効率を上げることができる。

「次々と generate(X) で生成される X を、 process(X) で処理する。ただし X が意図されたものかどうかは test(X) で調べる。」 これを実現するには、下のゴール列ではいかにも効率が悪い。

..... generate(X), process(X), test(X), ...

このような時、遅延実行制御を用いると、次のように記述することができる。

..... test(X?), generate(X), process(X), ...

ここで、test(X?) は、遅延実行制御のCIL記法で、「変数Xに対してデモーンとして述語 test を登録する」という意味をもつ。generate でXが生成されるたびに test が起動され正しいXだけが process にわたるので効率は改善される。test を generate 内の適当な位置に埋め込めば同様の効率が得られるが、それでは test すべき位置を管理する必要が生じ、記述面で、大きな負担となる。

このように遅延実行制御は、実行順序を意識せずに、いわゆる宣言的にプログラムを記述するための一つの方法を与えている。上の場合では、test(X?) は「X は test を満足する」という宣言であるとみなせる。

遅延実行制御のプリミティブ

CILの遅延実行制御は、PSIのbind_hookを用いて実現される。その仕様は次のとおりである。

... bind_hook(X, demon(X)), ...

のような記述は、X に値が入った時点で述語 demon(X) を実行するように指定しているものである。ヘッド・ユニフィケーションや組込述語による変数への代入が発生すると，デーモンはそのゴールのサブゴールとして起動される。

... bind_hook(X, demon(X)), ... X=a, ...
デーモン demon(a) がこの単一化のサブゴールとして実行される。

もし bind_hook 実行時に、既にX が具体化されていればデーモンは即座に実行される。

... X = a, .. bind_hook(X, demon(X)), ...
デーモン demon(a) がサブゴールとして即座に実行される。

同じ変数に複数回フックをかけると、その変数が具体化されたときには両方が起動される。例えば、

... bind_hook(X, p(X)), bind_hook(X, q(X)), ...

というゴール列は、

... bind_hook(X, (p(X), q(X))), ...

と同じである。（この場合、起動される順序は保証しない）またフックのかかった変数同士を単一化すると、その結果の変数には両方のフックがかかった状態になる。例えば

... bind_hook(X, p(X)), bind_hook(Y, q(Y)),X = Y, ...

というゴール列は

... bind_hook(X, p(X)), bind_hook(X, q(X)),X = Y, ...

と同じことになる。

遅延実行制御用シンタックス・シュガー

遅延実行制御を簡単に記述するために，以下の諸記法が提供されている。

1. 遅延実行を指定する制御変数の記法
   「引数が未定義の場合、具体化されるまで実行を中断する述語中の変数」を X? と書く。
   print(X?) は bind_hook(X, print(X)) を意味する記述である。

   ［例］ print(X?), X = ok. は、ok が表示される。
   print(X?). だけでは、X は未定義のままで何も表示されない。

2. @ 記法 （遅延実行型の条件付項）
   @p は、X:p(X?) の略記であり、X@p は X:p(X?) の略記である。
   ［例］
   X = {a/b}, Y = {a / @print}, X = Y. は、
   b が表示され、X = Y = {a/b} となる。

この記法と部分項を用いることで知識表現でいう継承関係を表すことができる。
［例］
man(@animal@clever). は、
man(X) :- animal(X), clever(X). とほぼ同じである。

3. ?? 記法
X?? は、T:(T?=X) の略記である。
［例］
X = a(Y??) は freeze(T, (T=Y)), X=a(T) と同じである。
これを実行すると Y の状態にかかわらず X は具体化する。
しかし、X = a(Y?) は freeze(Y, X=a(Y)) と同じのため、
Y が未定義の場合、X は具体化しない。

1.1.3 制約言語

一般に、構文解析や意味解析などの制約を効率良く解釈するための一般的な制御手段の提供は難しい。CIL ではその出発点として、上記の遅延実行制御の応用として簡単な制約言語を設定し、そのインタプリタとして、組込述語 constrを提供している。現在、この言語の上で、等式、不等式、基本算術演算等を接続子で結合した論理式が制約として記述できる機構を備えている。

制約言語は組込述語 constr の第一引数に記述できる。その文法を以下に示す。

```
<制約>   ::= <基本制約>
           | "not(" <制約> ")"
           | <順接続子> "(" <制約> "," <制約> ")"
           | <制約> <接続子> <制約>
           | <IF 接続>

<基本制約> ::= <論理制約> | <原始制約> | <計算制約> | <ASSIGN 制約>
<論理制約> ::= "true" | "false" | <変数>
<原始制約> ::= <項> "=" <項> | <項> "\==" <項>
<計算制約> ::= <計算式> <計算関係子> <計算式>
<計算関係子> ::= "=:=" | "=\=" | "<" | ">"
<計算式> ::= <数> | <変数> | <計算式> <演算子> <計算式>
<演算子> ::= "+" | "-" | "*" | "/" | "mod"
<ASSIGN 制約> ::= "assign(" <項> , <項> ")"
<順接続子> ::= "exor" | "nor" | "nand" | "and" | "or" | "imply"
<接続子> ::= "," | ";" | "->" | "<->"
<IF 接続> ::= "if(" <制約>, <制約>, <制約> ")"
```

［例］ constr((X!refl = (+) -> X!gr = sbj)).
上の例は「" 自分 "(refl 素性が +) には必ず主格となる句 (gr 素性が sbj) が束縛される」という JPSG の原則を表現している。

1.1.4 目標と現在の機能

遅延実行制御を用いた別の応用機能として、ストリームプログラミングがある。これは、数個のモジュールが密接に関連しながら動作する、いわゆるコルーチン処理プログラムの記述には有効な機能を果たす。

CILの目指すところは、現在、次図のようにまとめることができる。

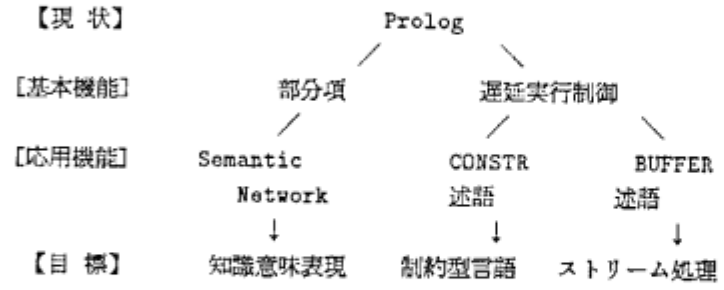


図 1.2: CILの目標と現在の機能

1.2 プログラミング環境

CIL処理系(CILプログラミング・システム)はPSI・SIMPOS環境でCILによるプログラミングを支援するシステムである。本節では処理系の特徴やプログラム開発の標準的手順、ユーザ・インターフェイスの概要について述べる。

1.2.1 処理系の特徴

CILの処理系はコマンド・インタプリタを介して行う対話型のシステムであり、プログラムの編集, デバッグ, 実行, ソース・ファイル管理, 運用管理などのCILによるプログラミング全般を支援することを目的としている。CILプログラミング環境の機能と特徴を以下に挙げる。

1. メニューとマウスを用いた操作仕様
   CILの操作仕様はマルチウインドウ機能とマウスを使用した操作仕様である。操作可能なオブジェクトやそれに対するコマンドをメニュー化することによって、ユーザがコマンドやオブジェクトの名前を暗記していなければならないような事態を排除している。

2. ユニット機能とファイル操作機能
   複数のCILプログラムによって一つの応用システムを作る場合の実行環境とプログラミング環境の管理を支援するユニット機能が提供されている。また、ユニット中のファイルを扱うための操作機能がある。

3. シンタックス・チェッカ付のテキスト・エディタ
   プログラム・エディタはPmacsエディタの機能を基本としてプログラム形式, クローズ内変数の誤りチェック, 未定義述語のチェックと編集を対話的に行うことが出来る。この機能により従来では実行時に判明していた誤りを編集時に修正することができる。

4. ソース・プログラムを即座に実行できるインタプリタ
   編集したプログラム・ファイルをコンサルトすれば、そのプログラムは実行可能となる。プログラムの実行はゴールを入力することによって行う。

5. スクリーン・イメージで行うデバッグ支援環境
   デバッグ支援環境は、ソース・プログラムをスクリーン・イメージで表示し、スクリーン上でのプログラムの編集が出来る。

6. プログラムの実行を高速化する最適化コンパイラ
   デバッグの完了したプログラムは最適化コンパイラを用いて高速化することができる。

7. 部分項、デーモン付変数を調べるインスペクタ
   CIL インスペクタは部分項や変数に掛かっているデーモンを調べることができ、必要に応じてトップ・レベルやデバッグ支援から呼び出すことが出来る。

8. 組込述語と組込シンタックス・シュガー
   CIL のプログラミングは PSI-II の KL0 組込述語と CIL の組込述語をベースに行う。組込シンタックス・シュガーによって部分項と遅延実行制御機能に関するプログラミングを容易に行う事ができる。

9. SIMPOS クラスのメソッド・コールを含む ESP マクロ機能
   CIL の言語機能のみではなく、SIMPOS クラスに対するメソッド・コール等の ESP マクロがプログラム上で使用できる。

10. ユーザ・シンタックス・シュガー機能
    これによって、ユーザーによる言語機能拡張が行える。頻繁に使うプログラム・パターンはシンタックス・シュガーの定義により、自動的に展開させることができる。 CIL はユーザー・シンタックス・シュガーを含んだプログラミング環境を提供している。

11. ツールボックス機能
    汎用的なプログラムはツールボックスに登録することにより、同じ PSI 上のすべてのユーザのユニットから利用可能となる。ツールボックスの管理と利用の機能を提供している。

1.2.2 プログラム開発手順

CIL 処理系を用いて、プログラム開発を行う場合の標準的手順を紹介する。 CIL を起動すると、インタプリタが動作しゴール列の入力待ちとなる。インタプリタ・コマンドでプログラムを格納するユニットを作成する。

ユニットを選択して、プログラム・エディタを呼出し、ソース・プログラムを入力する。プログラム・エディタのシンタックス・チェック機能を使用して、文法エラーを取る。文法エラーが無くなったならば、ソース・プログラムをセーブする。ただし、ソース・プログラムのファイル・タイプは cil でなければならない。

プログラムをコンサルトすると、ユーザ述語をゴール列として入力できる。インタプリトしてみて、結果が正しくなければデバッグを行う。インタプリト・コマンドでそのプログラムのデバッグ支援環境を作成する。再度トップ・レベルからゴールを入力すると、プログラムの実行箇所がスクリーン上で反転表示されるので、正しく動作してない箇所をデバッグ・コマンドを用いて究明する。原因が判明したら、デバッグ支援環境上でソース修正を行うと、自動的にリコンサルトされる。再度、トップ・レベルからインタプリトを行い、プログラムが正しく動作するようになるまでこれを繰り返す。

デバッグが完了したら、プログラム・ファイルをセーブしてコンパイルすると、プログラムを高速に実行することができる。

1.2.3 ユーザ・インターフェイス

処理系のユーザ・インターフェイスには、コマンド・インタプリタ， プログラム・エディタ，デバッグ支援，インスペクタがある。コンパイラとインタプリタは直接のユーザ・インターフェイスを持たない。コンパイラとインタプリタはコマンド・インタプリタによって呼ばれる。

コマンド・インタプリタは、インタプリタ・コマンドを入力して、実行する。インタプリタはソース・プログラムを入力して、クローズ・データベースに保存し、コマンド・インタプリタが入力したゴール列の実行を行う。コンパイラは、ソース・プログラムを入力してコンパイルド・プログラムを生成する。プログラム・エディタは、プログラムの編集に用いる。デバッグ支援はソース・プログラムを入力して，ソース・プログラムのデバッグの支援を行う。インスペクタはトップレベル変数やゴールを入力して、データ構造の解析を行う。

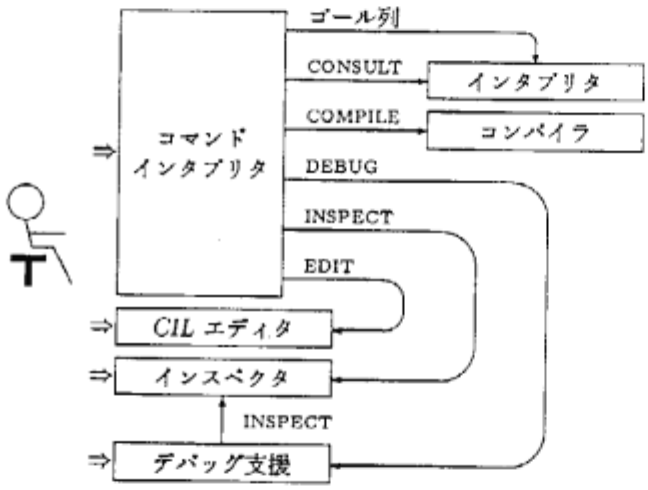


図 1.3: ユーザ・インターフェイス

第2章

操作仕様

本章ではプログラム開発および実行の際に必要とされる処理系の操作仕様について述べる。

2.1 インタプリト時の操作

2.1.1 トップレベル操作の概要

SIMPOSシステム・メニューからCILを起動すると、コマンド・インタプリタ・ウインドウが生成され、コマンド・インタプリタが起動される。コマンド・インタプリタは、ゴール列をウインドウから入力し、解釈・実行を行う。ゴール列の入力待ちの状態をトップレベルと呼び、インタプリタ・コマンド、CIL組込述語、ユーザ述語、ESPメソッド、KL0組み込述語を入力することができる。

コマンド・インタプリタ・ウインドウは、ゴール入力のためのウインドウ、コマンド・メニューおよびバリアブル・メニュー からなる。ゴール入力のためのウインドウはPmacs ウインドウになっていて、Pmacsの各種機能を使用することができる。コマンド・メニューはスクローリング・メニューで、インタプリタ・コマンドの一覧が表示されている。またトップレベルで使用した変数はトップレベル変数と呼ばれ、バリアブル・メニュに保持される。バリアブル・メニューもスクローリング・メニューになっていて、トップレベル変数の一覧が表示されている。

インタプリタ・コマンドとしてはユニットの作成、設定、削除、編集やCILプログラムのエディット、コンサルト、デバッグ、コンパイル、また、各種実行モードの設定等があり、コマンド・メニューの項目をマウスで指定することにより指令することができる。

2.1.2 起動と終了

CILをSIMPOSシステム・メニューから呼び出すとウインドウの位置と大きさを聞いてくるので、マウスを使って指定する。CILの初期化が終わるとゴール列の入力待ちとなり、上側にコマンド・メニューが、右側にバリアブル・メニューが表示される。CILプロセスを終了させるにはhalt_cilコマンドを使う。

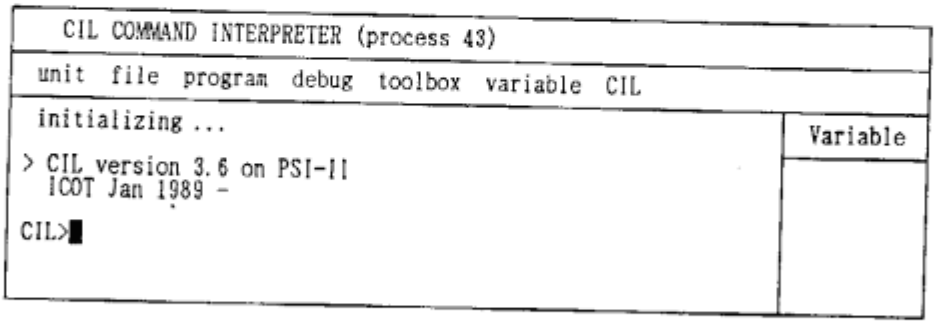


図 2.1: CILの起動

2.1.3 ゴール入力と結果の表示

インタプリタの入力要求に対して、例えば次のようにゴール列を入力する。

```
CIL> A = {a/1},  B= A!a.
```

この時、入力は". "（ピリオド）"改行"で終わらなければならない。" CIL> "はプロンプトである。インタプリタは入力されたゴール列を解釈・実行し、次のように結果を表示する。

```
A => {a/1}
B => 1
yes
```

A と B はトップレベル変数として保持され、次のゴール入力の変数として使用できる。次のゴール入力の変数として使用したくない場合は、バリアブル・メニュー上のその変数をマウスにより反転させると、ゴール列の解釈・実行の前に、その変数は削除される。

また、ゴールの入力中に Pmacs コマンドが使用できる。ただし、Pmacs コマンドを使って画面を編集しても、インタプリタの入力としてみなされるのは Pmacs の入力カーソル（「）の後ろだけである。

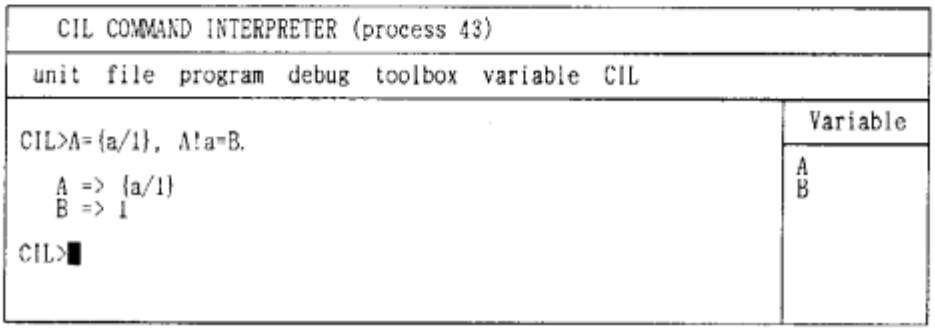


図 2.2: ゴール入力と結果の表示

2.1.4 インタプリタ・コマンド

以下にインタプリタ・コマンドを説明する。

ユニットに対するコマンド

複数の CIL プログラムによって一つの応用システムを作る場合、ソース・プログラムを管理するためのディレクトリやコンパイルド・プログラムのクラスおよびパッケージを管理しなければならない。また実行時の既定のディレクトリも、そのシステム毎に必要である。

CIL 処理系では、システムの実行環境とプログラミング環境を一つの単位としてとらえ、システムの名前を指定することによってプログラミング環境と実行環境を自動的に設定する機能を提供している。その単位をユニットと呼ぶ。

1. make_unitコマンド
   新しいユニットを作る。具体的には、ユーザ名にユニット名を付加したものを名前とする SIMPOS パッケージと" cil "にユニット名を付加したものを名前とするディレクトリをユーザ・ディレクトリ (me:) の下に作成する。

   コマンド・メニューの unit をマウスセレクトすると、サブメニューが表示される。その中から make をマウスセレクトするとユニット名を聞いてくるのでキーボードから入力する。この際、ユニット名として許される文字数は 10 文字以内である。

2. delete_unitコマンド
   ユニット（SIMPOS パッケージ、ディレクトリ）を削除する。コマンド・メニューの UNIT をマウスセレク

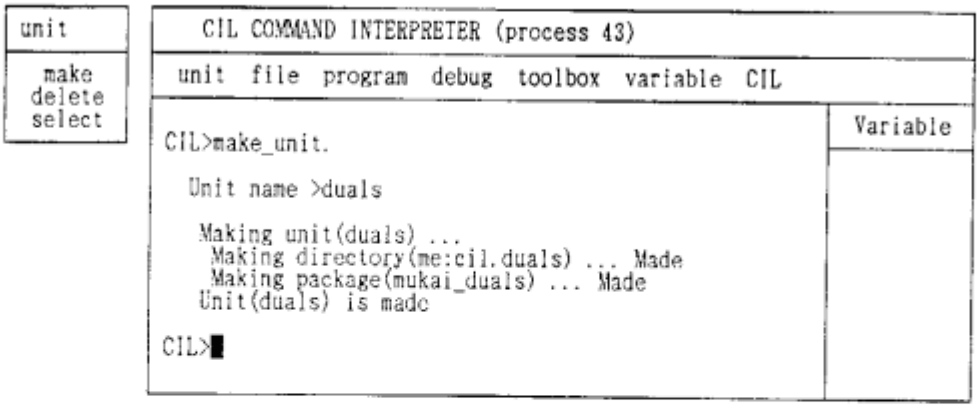


図 2.3: make_unit コマンドの例

トすると、サブメニューが表示される。その中から delete をマウスセレクトすると、既に作成済みのユニット名一覧を表示したユニット・メニューが現れるので、削除したいユニット名をマウスセレクトする。確認メニューが現われるので yes を選択する。

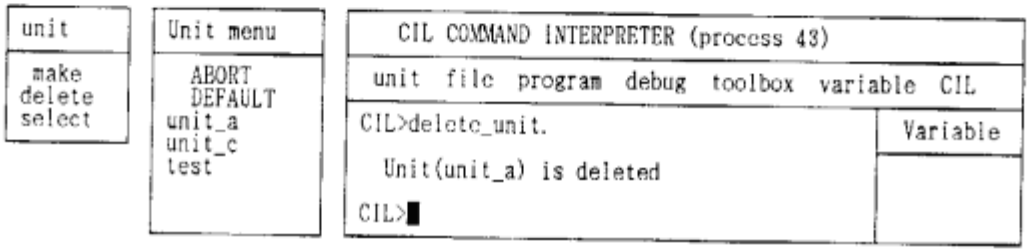


図 2.4: delete_unit コマンドの例

3. select_unitコマンド
   使用するユニットを選択する。選択されるとそのユニットのパッケージとディレクトリがデフォルトとして設定される。(ユニットを select するまでは CIL 起動時のパッケージとディレクトリが設定されている)
   コマンド・メニューの UNIT をマウスセレクトすると、サブメニューが表示される。その中から select をマウスセレクトすると、ユニットの一覧を表示したユニット・メニューが現れるので、設定したいユニット名をマウスセレクトする。 DEFAULT を選択すると CIL 起動時の状態が設定される。

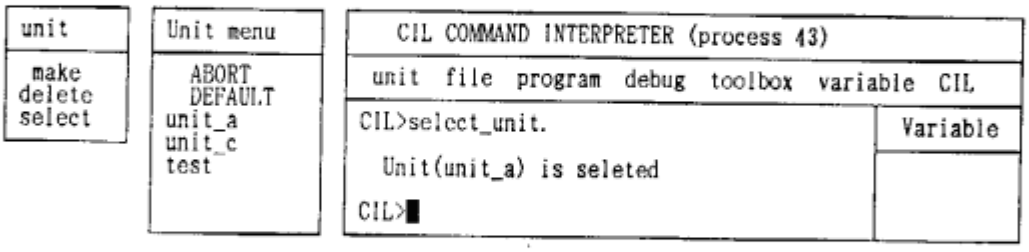


図 2.5: select_unit コマンドの例

ファイルに対するコマンド

1. bring_fileコマンド
   bring_file コマンドは他のディレクトリから現在のユニットにファイルをコピーする時に使用する。

コマンド・メニューのFILEをマウスセレクトすると、サブメニューが表示される。その中のbringをマウスセレクトすると、bringメニューが現れる。このメニューはユニットの枠を越えてディレクトリを上下できるようになっていて、ディレクトリの内容が表示される。表示中のディレクトリ名はメニューのラベルに示される。コマンド起動時は現在選択されているユニット内のファイル一覧が表示される。

PARENTをクリックすると親のディレクトリの内容が表示される。また、メニュー上に表示されたディレクトリをクリックするとそのディレクトリの内容が表示される。 DEFAULTをクリックすると、現在選択されているユニットの内容が表示される。

このメニューはマルチセレクトメニューになっているのでコピーしたいファイルをマウスで指定した後、DO_ITをクリックする。

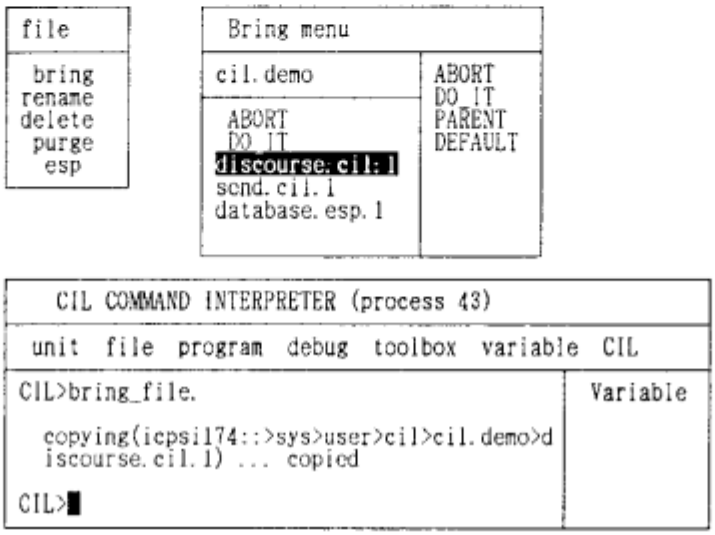


図 2.6: bring_file コマンドの例

2. rename_fileコマンド
rename_file コマンドは現在のユニット内のファイルの名前を変更する時に使用する。コマンド・メニューのFILEをマウスセレクトすると、サブメニューが表示される。その中のrenameをマウスセレクトすると、現在選択されているユニット内のファイル名一覧がrenameメニューに表示されるので変更したいファイル名をマウスで指定する。このメニューはマルチセレクトメニューになっているので改名するファイルをすべて指定した後、DO_ITをクリックする。インタプリタ・ウインドウに入力が要求されるので新しいファイル名をキーボードから入力する。

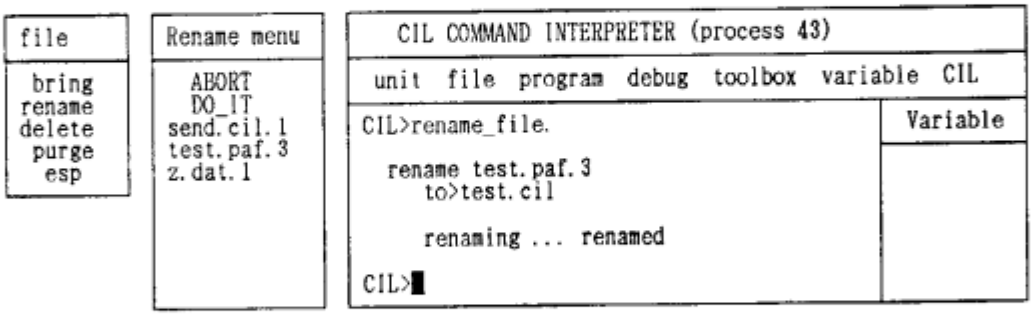


図 2.7: rename_file コマンドの例

3. delete_fileコマンド
delete_file コマンドは現在のユニット内のファイルを消去する。コマンド・メニューのFILEをマウスセレク

トすると、サブメニューが表示される。その中の delete をマウスセレクトすると、現在のユニット内のファイル名の一覧が delete メニューに表示されるので削除したいファイル名をマウスで指定する。 delete メニューはスクローリング・マルチセレクトメニューである。

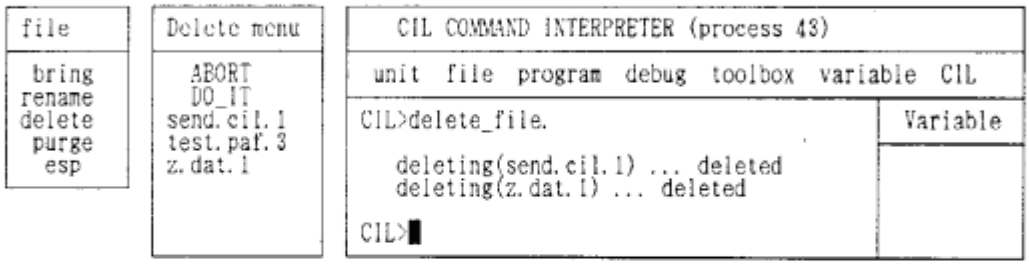


図 2.8: delete_file コマンドの例

4. purge_fileコマンド

purge_file コマンドは現在のユニット・ディレクトリ内のパージを行う。コマンド・メニューの FILE をマウスセレクトすると、サブメニューが表示される。その中の purge をマウスセレクトすると、パージが行われる。パージはファイルのバージョンの最新のものを残し、古いバージョンのファイルを削除する。

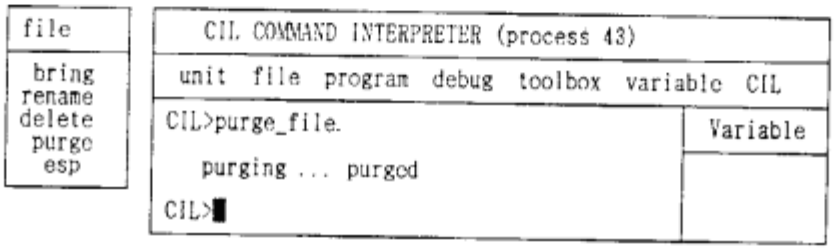


図 2.9: purge_file コマンドの例

5. esp_fileコマンド

esp_file コマンドは現在のユニット中の ESP プログラムをユニット・パッケージ内にカタログ・セーブする。コマンド・メニューの FILE をマウスセレクトすると、サブメニューが表示される。 esp をマウスセレクトすると、現在のユニット内の ESP プログラムの一覧が esp メニューに表示されるのでカタログ・セーブしたいファイル名をマウスで指定する。 esp メニューはスクローリング・マルチセレクトメニューである。

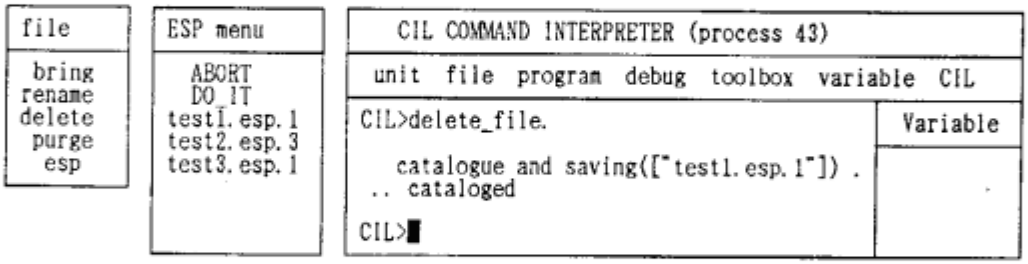


図 2.10: esp_file コマンドの例

プログラムに対するコマンド

1. edit_program コマンド
   現在選択しているユニット中の CIL プログラムの編集を行うために、プログラム・エディタを呼び出す。コマ

ンド・メニューの PROGRAM をマウスセレクトすると、サブメニューが表示される。その中から edit をマウスセレクトすると、プログラム・エディタ・ウインドウの位置と大きさを聞いてくるのでマウスで指定する。それ以後の呼出しの場合はその位置に表示される。プログラム・エディタの操作については『エディット時の操作』を参照のこと。

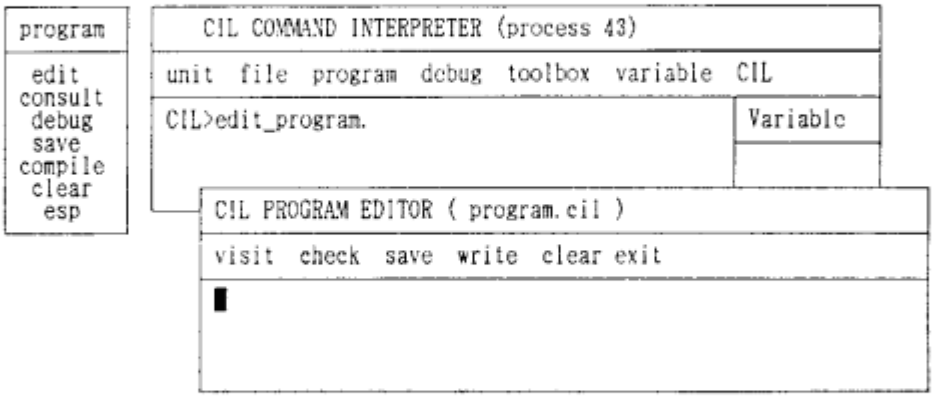


図 2.11: edit_program コマンドの例

2. consult_program コマンド

プログラムのコンサルトおよびコンサルトの解除を行う。 コマンド・メニューの PROGRAM をマウスセレクトすると、サブメニューが表示される。その中から consult をマウスセレクトすると、現在選択されているユニット中の CIL プログラムの一覧を表示したコンサルト・メニュが現れる。既にコンサルトされているプログラムは、コンサルト・メニュー上で反転表示されている。コンサルト・メニューは、マルチプルセレクト・メニューになっているので、複数のプログラムをセレクトすることが出来る。

プログラムをコンサルトする場合は、コンサルト・メニュー上のコンサルトしたいプログラムをセレクトした後、DO_IT をクリックする。

コンサルトされているプログラムを解除する場合は、コンサルトを解除したいプログラムをクリックして反転表示を解除した後、DO_IT をセレクトする。

同じファイル識別子のプログラムを同時にコンサルト状態にすることは出来ないので注意すること。 例えば、send.cil.1 をコンサルトした状態のままでは、さらに send.cil.2 や send.cmp.1 などをコンサルトすることは出来ない。そのような状態で、send.cmp などをコンサルトする場合、 send.cil.1 のコンサルトを解除しなければならない。

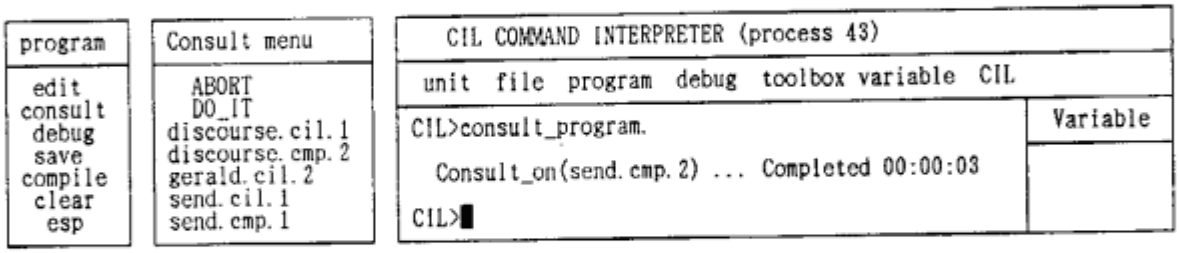


図 2.12: consult_program コマンドの例

3. debug_programコマンド

プログラムのデバッグの開始およびデバッグの終了を行う。コマンド・メニューの PROGRAM をマウスセレクトすると、サブメニューが表示される。その中から debug をマウスセレクトすると、現在コンサルトされて

いるCILソース・プログラムの一覧を表示したデバッグ・メニューが現れる。既にデバッグ中のプログラムは、デバッグ・メニュー上で反転表示されている。デバッグ・メニューはマルチプルセレクト・メニューになっているので、複数のプログラムをセレクトすることが出来る。

プログラムのデバッグを開始する場合は、デバッグ・メニュー上のデバッグしたいプログラムをセレクトした後、DO_ITをクリックする。するとデバッグ支援ウインドウの位置と大きさを聞いてくるので、マウスを使って指定する。

デバッグを終了する場合は、デバッグ・メニュー上の終了したいプログラムをクリックして反転表示を解除した後、DO_ITをセレクトする。

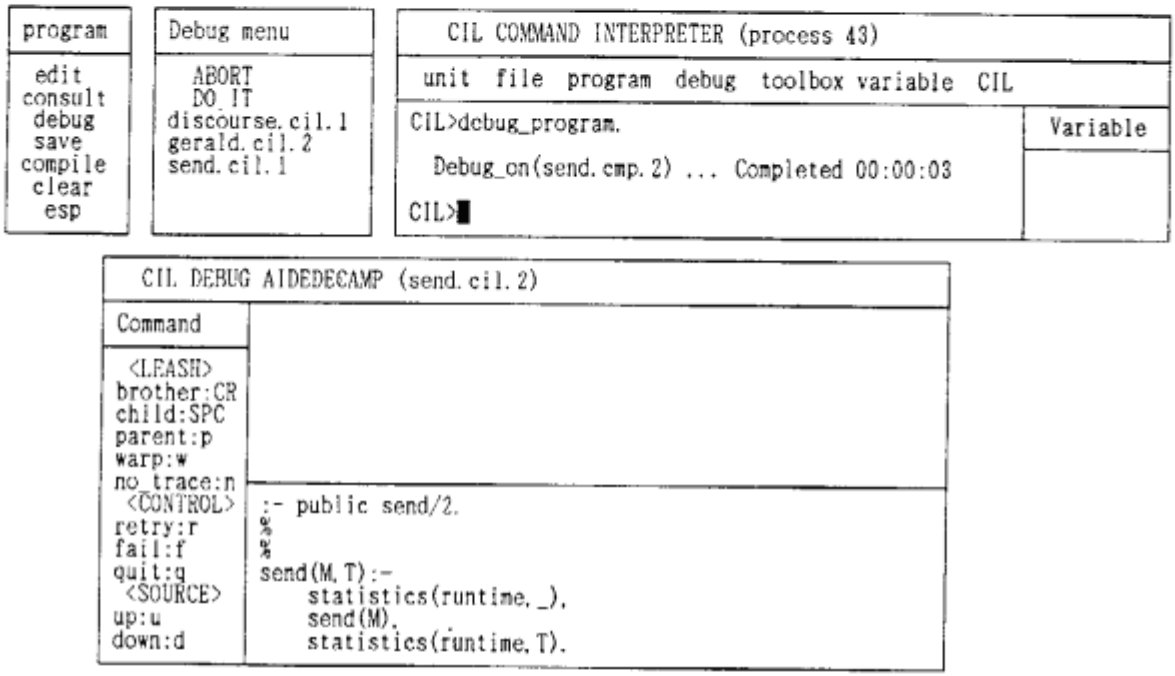


図 2.13: debug_program コマンドの例

4. save_programコマンド

デバッグ支援で修正したプログラムをセーブする。デバッグ支援のエディット機能を使ってデバッグ中のプログラムを修正すると、そのプログラムは自動的にセーブ・メニュに登録される。

コマンド・メニューのPROGRAMをマウスセレクトすると、サブメニューが表示される。その中からsaveをマウスセレクトすると、現在登録されているCILソース・プログラムの一覧を表示したセーブ・メニューが現れるので、セーブしたいプログラムをセレクトした後、DO_ITをクリックする。セーブ・メニューはマルチプルセレクト・メニューになっているので、複数のプログラムをセレクトすることが出来る。

プログラムのセーブが終了するとセーブ・メニューから取り除かれる。またデバッグ終了時、CILの終了時にセーブされていないプログラムがあった場合、確認メニューが現われてセーブするかどうか確認の問い合せを行う。

5. compile_program コマンド

CILソース・プログラムをコンパイルする。コマンド・メニューのPROGRAMをマウスセレクトすると、サブメニューが表示される。その中からcompileをマウスセレクトすると、現在選択されているユニット中のCILソース・プログラムの一覧を表示したコンパイル・メニューが現れるので、コンパイルしたいプログラムをセレクトした後、DO_ITをクリックする。コンパイル・メニュは、マルチプルセレクト・メニューになっているので、複数のプログラムをセレクトすることが出来る。

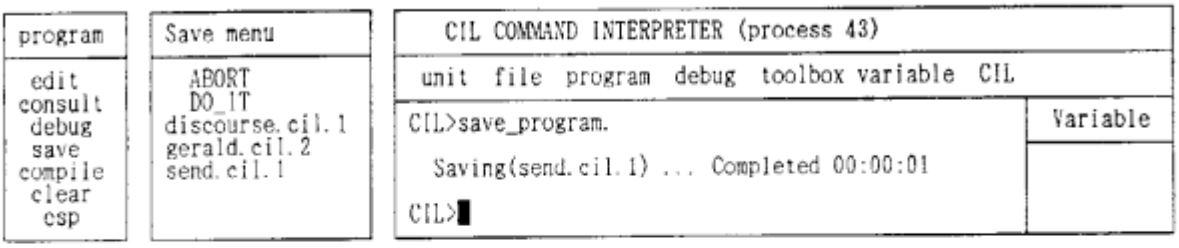


図 2.14: save_program コマンドの例

CIL ソース・プログラムをコンパイルすると、同じファイル識別子でファイル・タイプが cmp のコンパイルド・プログラム用ファイルが作られる。コンパイルしたプログラムを実行するには、このファイルをコンサルトすれば良い。例えば、send.cil.2 コンパイルすると、send.cmp.1 が作られるので、これをコンサルトする。

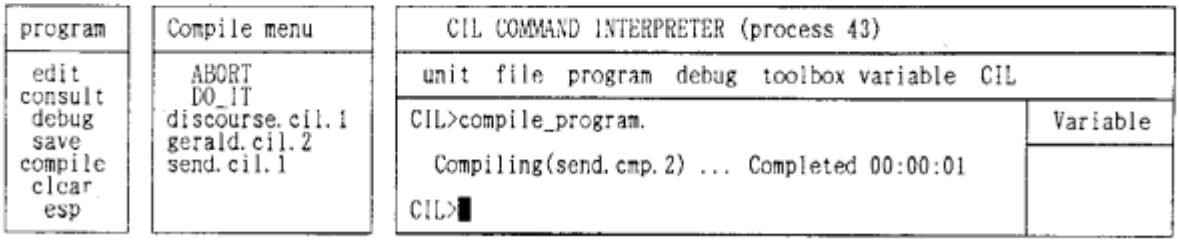


図 2.15: compile_program コマンドの例

6. clear_programコマンド

現在コンサルトされているすべてのプログラムのコンサルトを解除する。コマンド・メニューの PROGRAM をマウスセレクトすると、サブメニューが表示される。その中から clear をマウスセレクトすると、確認メニューが現れる。 yes をセレクトした場合のみ clear が実行される。

デバッグ支援に対するコマンド

1. spy_debugコマンド

デバッグ支援の spy コマンドがインタプリタから使用できる。スパイポイントの設定を行う。

コマンド・メニューの DEBUG をマウスセレクトすると、サブメニューが表示される。その中から spy をマウスセレクトするとスパイ・メニューが現れる。設定方法等はデバッグ・コマンドの spy と同じである。デバッグ・コマンドを参照のこと。

2. gate_debug コマンド

デバッグ支援の gate コマンドがインタプリタから使用できる。ゲートの設定を行う。コマンド・メニューの DEBUG をマウスセレクトすると、サブメニューが表示される。その中から gate をマウスセレクトするとゲート・メニューが現れる。設定方法等はデバッグ・コマンドの gate と同じである。デバッグ・コマンドを参照のこと。

3. see_debugコマンド

デバッグ支援の see コマンドがインタプリタから使用できる。デバッグ中のプログラムを参照する。コマンド・メニューの DEBUG をマウスセレクトすると、サブメニューが表示される。その中から see をマウスセレクト

するとデバッグ支援ウインドウが現れる。操作方法等はデバッグ・コマンドの see と同じである。デバッグ・コマンドを参照のこと。

トップレベル変数に対するコマンド

1. inspect_variableコマンド

インスペクタを起動して、現在バリアブル・メニュー上にあるトップレベル変数の内容を表示する。コマンド・メニューの VARIABLE をマウスセレクトすると、サブメニューが表示される。その中から inspect をマウスセレクトするとウインドウの位置を聞いてくるのでマウスで指定する。するとインスペクタ・ウインドウが現れ、変数にバインドされている内容が表示される。

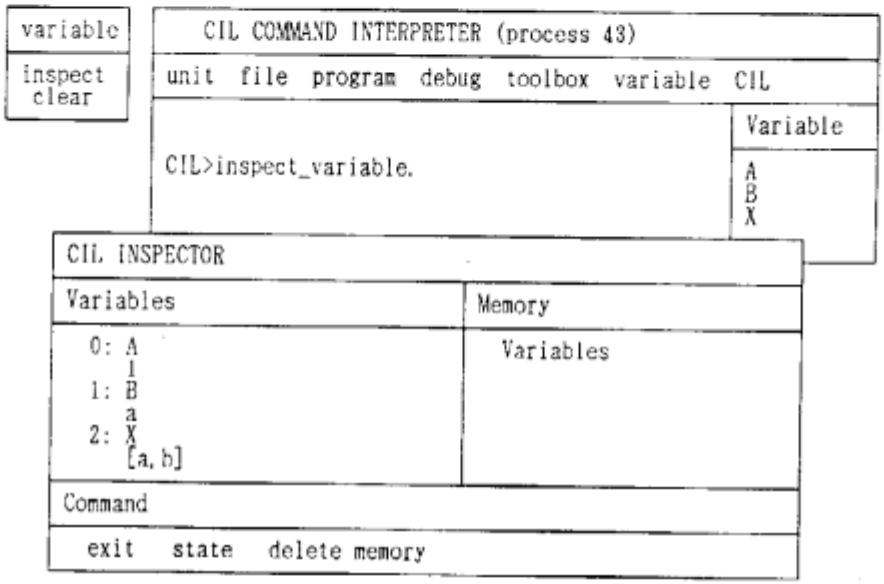


図 2.16: inspect_variable コマンドの例

2. clear_variableコマンド

トップレベル変数をすべて解除する。コマンド・メニューの VARIABLE をマウスセレクトすると、サブメニューが表示される。その中から clear をマウスセレクトすると、すべての変数のバインドが解除され、バリアブル・メニューがクリアされる。

また、指定した変数のみ解除したい場合は、バリアブル・メニュー上の解除したい変数をマウスクリックすればよい。

処理系に対するコマンド

1. halt_cilコマンド

CIL プロセスを終了する。コマンド・メニューの CIL をマウスセレクトすると、サブメニューが表示される。その中から halt をマウスセレクトすると、確認メニューが現れる。 yes をセレクトした場合のみ halt が実行される。

2. state_cilコマンド
CIL の状態を変更する。コマンド・メニューの CIL をマウスセレクトすると、サブメニューが表示される。その中から state をマウスセレクトするとステート・メニューが現れる。現在設定されている状態が反転表示さ

れている。ステート・メニューはチョイス・メニューになっているので、設定したい状態をマウスセレクトした後、do_it をクリックする。

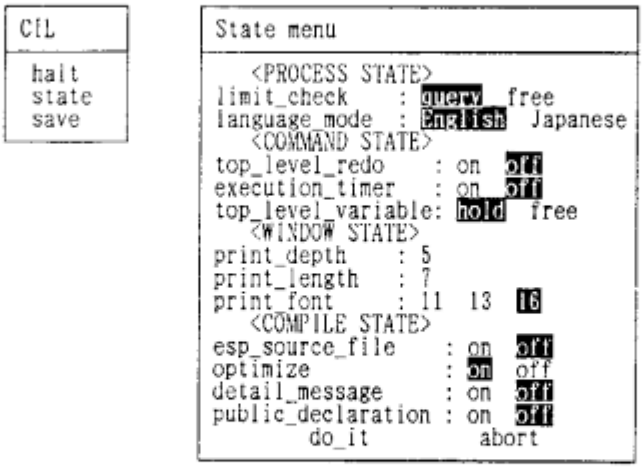


図 2.17: state_cil コマンドの例

CIL の状態としては以下のものがある。

- プロセスの状態
  - limit_check
    SIMPOS のリミット・チェック機能の処理モードを設定する。 query か free のどちらかを選択できる。既定値は query である。
  - language_mode
    メッセージを英語で表示するか日本語で表示するかを設定する。既定値は英語である。
- コマンド・インタプリタの状態
  - top_level_redo
    on ならばトップレベルでの実行後、別解探索を行うかどうか聞いてくる。このとき ; を入力すると別解を求めることが出来る。既定値は off である。
  - execution_timer
    on ならばトップレベルでの実行後、応答時間が表示される。既定値は off である。
  - top_level_variable
    hold ならばトップレベル変数の保持を行う。 free ならばトップレベル変数は保持されない。既定値は hold である。
- ウインドウの状態
  - print_depth
    ホロフラスティング機能の構造体表示の深さを決める。既定値は 5 である。 -1 にすると全て表示される。
  - print_length
    ホロフラスティング機能の構造体表示の長さを決める。既定値は 7 である。 -1 にすると全て表示される。
  - print_font
    ウインドウ・フォントを設定する。 11, 13, 16 を選択できる。漢字は 16 の時のみ表示可能である。
- コンパイルの状態

- esp_source_file
  CIL ソース・プログラムはコンパイルの際に ESP クラスに変換されるが、変換されたクラスを ESP ソース・ファイルとして生成するかどうかを選択する。 on にすると、CIL ソース・ファイルと同じファイル識別子でファイル・タイプが esp のファイルが生成される。 off にするとファイルは生成されない。既定値は off である。
- optimize
  コンパイルの際に最適化を行うかどうかを設定する。 on にすると最適化を行う。既定値は on である。最適化は単一化と制約言語に対して行われる。最適化を行うとプログラムの実行効率が向上する。しかしながら、そのプログラムをコンパイルするための時間がかかる。
- detail_message
  コンパイル中に各段階の進行状況のメッセージを表示するかどうかを選択する。 on にすると表示される。既定値は off である。
- public_declaration
  on ならば、コンパイルド・プログラムのエントリとしてパブリック宣言 された述語のみとする。 off ならば、全ての述語が コンパイルド・プログラムのエントリとなる。既定値は off である。パブリック宣言を on でプログラムをコンパイルする場合、プログラム中で組込述語のメタ述語を使用しているならばそのメタ述語によって呼び出される述語はパブリック宣言しなければならない。

3. save_state コマンド
   CIL の状態をセーブする。コマンド・メニューの CIL をマウスセレクトすると、サブメニューが表示される。その中から save をマウスセレクトすると CIL の状態がセーブされる。セーブ情報はファイル me:cil_state.init に保存される。セーブされた値はその次に CIL を起動した時の既定値となる。この機能により CIL 使用者は自分の好みの既定値を設定できる。セーブされる項目には以下のものがある。

   - state_cil コマンドで設定された項目
   - コマンド・インタプリタのウインドウ・サイズ
   - プログラム・エディタのウインドウ・サイズ
   - デバッグ支援のウインドウ・サイズ
   - ツール・ボックス・ブラウザのウインドウ・サイズ
   - インスペクタのウインドウ・サイズ
   - ゲートのリーシュとトレースの設定

   また、SIMPOS のセーブ機能によって、ログイン時 CIL プロセスを起動することもできる。その時にセーブされる項目も上述のものである。

2.2 エディット時の操作

2.2.1 プログラムの編集の開始と終了

コマンド・メニューの PROGRAM をマウスセレクトすると、サブメニューが表示される。その中から edit をマウスセレクトし、プログラム・エディタ・ウインドウの位置と大きさをマウスで指定すると、プログラム・エディタが起動される。2回目以降の呼出しの場合はその位置に表示される。

プログラム・エディタのラベルには現在編集中のプログラム名が表示されている。そのデフォルトは program.cil である。ラベルの下には、エディタ・コマンド・メニューがありエディタ・コマンドが表示されている。その下のウインドウが編集用のウインドウである。編集用ウインドウは Pmacs ウインドウであり、各種 Pmacs コマンドが使用できる。

プログラムの編集を終えて、コマンド・インタプリタのトップレベルに戻るには、エディタ・コマンド・メニューの EXIT をマウスセレクトする。

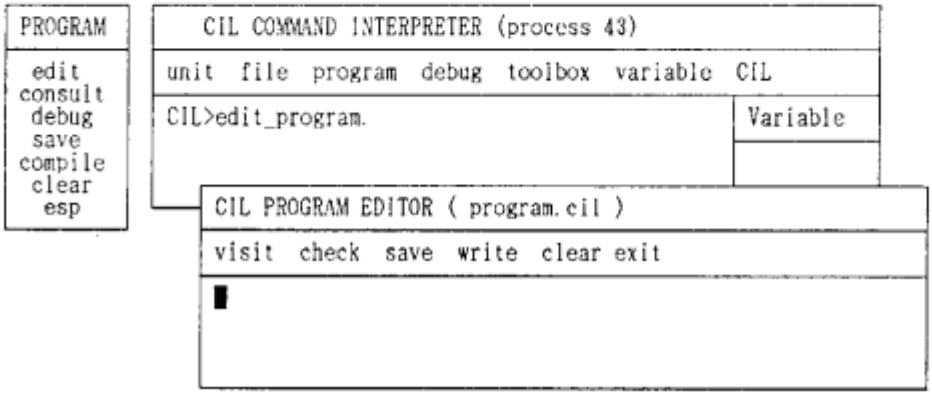


図 2.18: プログラム・エディタの起動

2.2.2 プログラム・エディタ・コマンド

1. visit コマンド

   visit コマンドは編集するソース・プログラム名を指定する。エディタ・コマンド・メニュー中の visit をマウスクリックするとビジット・メニューが表示される。ビジット・メニュー中には現在選択されているユニット中の CIL プログラムの一覧が表示される。ビジット・メニューはスクローリング・シングルセレクト・メニューである。

2. check コマンド

   check コマンドは編集中のソース・プログラムの文法チェックを行う。文法チェックとしては、プログラム形式 のチェック、クローズ単位のチェック、述語単位のチェックを行う。

   プログラム形式のチェックでは、正しい項の記述をしているかどうかを調べる。エディタ・コマンド・メニュー中の check をマウスクリックすると、始めにプログラムのチェックが行われ、program cheking ... とエコー・エリア・ウインドウに表示される。不正な項が発見されると、エディタ確認メニューが表示されるので、誤り箇所を確認の後、 edit を選択し編集状態に戻る。

   プログラム形式のチェックが終わると、つぎにクローズ単位のチェックが行われ、 clause checking ... とエコー・エリア・ウインドウに表示される。クローズ単位のチェックでは、クローズ内に現われる変数名が有効であるかどうかを調べる。無効な変数 名を持つクローズが発見されると、そのクローズが反転表示され、エディタ確認メニューが表示される。ここで edit を選択すると編集状態になり、 continue を選択すると、チェックが続行される。下の例では変数 Y を無名変数 _ にするのが望ましい。

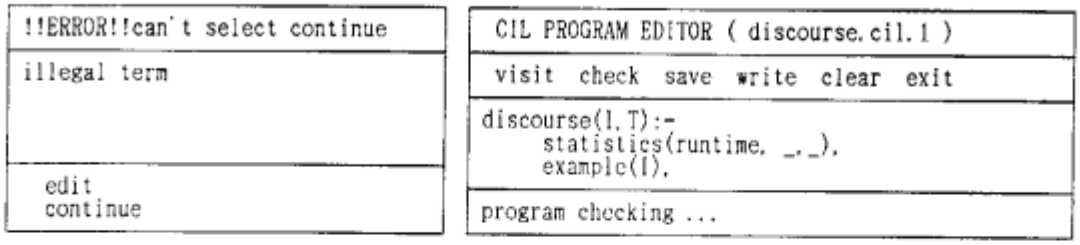


図 2.19: プログラム形式チェック時の例

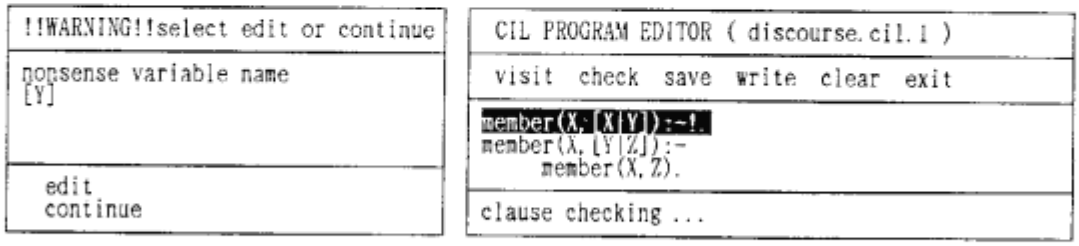


図 2.20: クローズ単位チェックの例

クローズ単位のチェックが終わると、つぎに述語単位のチェックが行われ、 predicate checking ... とエコー・エリア・ウインドウに表示される。述語単位のチェックでは、ボディ中の述語が定義されているかどうか、またはヘッドの述語が参照されているかどうかを調べる。未定義の述語や未参照の述語が発見されると、その述語が反転表示され、エディタ確認メニューが表示される。ここで edit を選択すると編集状態になり continue を選択するとチェックが続行される。

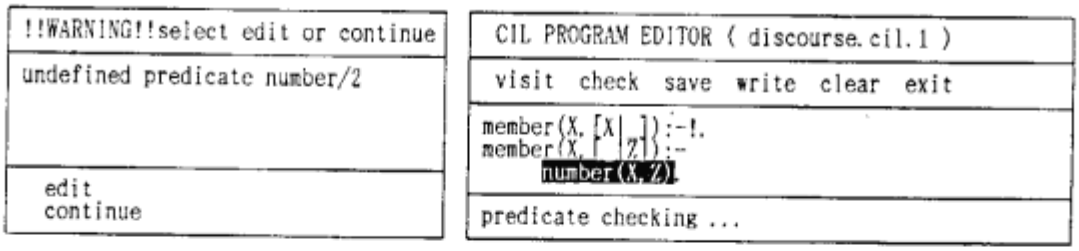


図 2.21: 述語単位チェックの例

3. save コマンド
   save コマンドは編集中のソース・プログラムのセーブを行う。

4. write コマンド
   write コマンドは編集中のソース・プログラムをファイルにする。ファイル名を指定する事がsave コマンドと異なる。

5. clear コマンド
   編集中のバッファを消去する。この時，確認を求めて来る。

6. exit コマンド
   exit コマンドはプログラムの編集をやめてCIL のトップ・レベルに戻る。現在のバッファーの内容は次回の編集時の既定値となる。

2.2.3 Pmacs コマンド

プログラムの編集中に Pmacs コマンドが使用できる。詳細は SIMPOS 操作説明書を参照すればよいが、以下に有用なコマンドを述べる。ここで c-x は control キーと x キーを同時に打つことを示し、m-x は esc キーを打ち、x キーを打つことを示す。

1. カーソルの移動と画面のスクロール

   c-f , → カーソルを文字進める

   c-b , ← カーソルを文字後退させる

   c-p , ↑ カーソルを行上げる

   c-n , ↓ カーソルを行下げる

   c-a カーソルを行の先頭へ移動する

   c-e カーソルを行末へ移動する

   m-< カーソルをバッファの先頭へ移動する

   m-> カーソルをバッファの最後へ移動する

   m-v 前の画面を表示する

   c-v 次の画面を表示する

   c-space リージョン・マーカをカーソル位置に設定する

2. テキストの削除と移動

   c-d カーソル上の文字を削除する

   delete カーソルの左の文字を削除する

   c-k 行末までの文字列を削除し、ヤンクバッファに入れる

   c-w リージョン内の文字列を削除し、ヤンクバッファに入れる

   c-y ヤンクバッファの文字列をカーソルの左側に挿入する

3. ウインドウの操作

   c-x 2 2ウインドウ・モードにする

   c-x 1 1ウインドウ・モードにする

   c-x o ウインドウをセレクトする
   2ウインドウ・モードのときはもう一方のウインドウをセレクトし、1ウインドウ・モードのときは隠れているウインドウが表示され、元のウインドウは隠れる

4. 文字列の検索

   c-s 指定された文字列をカーソルから最後の方向へ検索し、カーソルを文字列の右側へ移動する

   c-r 指定された文字列をカーソルから先頭の方向へ検索し、カーソルを文字列の先頭へ移動する

   上記コマンドを実行中は以下のキー入力を受付ける。

   esc コマンドの実行をやめ、カーソルを現在の位置にする

   c-g コマンドの実行をやめ、カーソルを元の位置にする

   c-s 次の文字列を最後の方向に検索する

   c-r 次の文字列を先頭の方向に検索する

   delete 前回のカーソル位置に戻る

2.3 デバッグ時の操作

2.3.1 デバッグ機能の概要

インタプリタ・コマンドでプログラムに対してデバッグ指定するとそのプログラムに対してデバッグ支援ウインドウが作成される。トップレベルゴールの実行過程中にデバッグ指定されたプログラム中の述語が参照されると、その述語の実行過程をスクリーン・イメージで追跡することができる。 そして、その述語の実行過程の中断点で各種デバッグ・コマンドを用いてデバッグを行う。

デバッグ支援環境に対するコマンドはデバッグ支援ウインドウのメニューに一覧として表示されていて、マウスによって指定する。デバッグ・コマンドには次の中断点であるリーシュに対するコマンドやゴールのコントロールを変更するコマンド、表示中のソースプログラムを扱うコマンド、ゴールをインスペクトしたりスパイやゲートを設定するコマンド、CIL に対するコマンドなどがある。各種操作はメニュー化されている。

デバッグ支援環境の基本的イメージを以下に挙げる。

1. デバッグしたいプログラム毎にデバッグ支援環境 (ウインドウ) を生成する。
2. ウインドウ上にソースプログラムが表示されている。(スクリーンエディタ)
3. 中断点にある述語は反転表示されていて、実行過程にしたがって反転表示が移動する。
4. ヘッド・ユニフィケーションのときは選択中のヘッドが反転表示される。
5. レベル、ゲート名、具体化された述語等の表示用ウインドウは ソースプログラムを表示してるウインドウとは別にある。
6. 中断点にあるゴールの内容を調べるときはインスペクタを使う。
7. 通常はソース・レベルのトレースを行うが、シンタックス・シュガーの展開形 のトレースも行うことができる。
8. 誤り箇所が判った時点でそのソース・プログラム表示用のウインドウで 修正を行うことができる。 この時、プログラムは自動的にリコンサルトされる。
9. 複数のプログラムをコンサルトしている場合、 デバッグ用にウインドウを開いているプログラムのみがトレースの対象となる。 また実行の制御が移動したプログラムのウインドウがアクティブウインドウになる。

2.3.2 デバッグ・モデル

ボックス・モデル

トレース・コントロール・モデルとして、 Prolog では手続きに対するボックスモデル (procedure box model) が用いられている。 CIL ではヘッド中のシンタックス・シュガーの実行、ヘッド・ユニフィケーションによるデーモンの実行、ボディ中のユニフィケーションによるデーモンの実行、シンタックス・シュガーの展開形の実行などの詳細なトレースを実現するために四つのボックス (ヘッド・ボックス、ボディ・ボックス、デーモン・ボックス、イクスパンド・ボックス) に対する制御が行えるモデルを用いた。

各ボックスはプログラムの制御の流れを示す四つのゲート (CALL, EXIT, REDO, FAIL) を持ち、デーモンボックスに関しては更に生成と消滅を示す二つのゲート (DGEN, DDEL) を持つ。ゲートの名前は各ボックスのイニシャル (H, B, D, E) を付けて識別する。

つぎに各ゲートを説明する。

- CALL: ボックスの最初の呼出しを表す。
- EXIT: ボックスの成功を表す。
- REDO: ボックスのバックトラックによる再試行を表す。
- FAIL: ボックスの失敗を表す。

図 2.22: ボックスとゲート名

- DGEN: デーモンの生成を表す。 遅延実行引き数を伴うボックスがデーモンとなる時CALL と EXIT の間でこのゲートが表示される。
- DDEL: デーモンの消滅を表す。 デーモンを生成したボックスがバックトラックにより REDO と FAIL ゲートを通過するときこのゲートが表示される。

このモデルによるトレースは例えば以下のようになる。

プログラム

```
a(X):- integer(X?), one(X).
one(1).
```

トレース

```
?-a(X).
  1 HCALL a(X)                      … ヘッドボックスの呼出し
  1 HEXIT a(X)                      … ヘッドボックスの成功
  1 BCALL integer(X?)               … 遅延実行引き数付のボディボックスの呼出し
  1 ECALL bind_hook(X,integer(X))   … イクスパンドボックスの呼出し
  1 DGEN integer(X?), X             … デーモンボックスの生成
  1 EEXIT bind_hook(X,integer(X))   … イクスパンドボックスの成功
  1 BEXIT integer(X?)               … 遅延実行引き数付のボディボックスの成功
  1 BCALL one(X)                    … ボディボックスの呼出し
  2 HCALL one(1)                    … ヘッドボックスの呼出し
  3 DCALL integer(1)                … デーモンボックスの呼出し
  3 DEXIT integer(1)                … デーモンボックスの成功
  2 HEXIT one(1)                    … ヘッドボックスの成功
  1 HEXIT one(1)                    … ボディボックスの成功
  X => 1
  yes
```

実際にはソース・プログラムの実行箇所が反転されるのでボックスの種類をあまり意識する必要はない。またイクスパンドボックスは普通はゲートをオフしてあるためトレースされない。

ボックス・レベル

上記で説明したボックスは各ゲートで呼出しのレベルを持っている。呼出しのレベルはそのボックスが現在呼ばれている深さを示している。

トップレベルのゴールから生成されるボディ・ボックスのレベルは 0 である。ただしトップレベルはデバッグの対象外であるためレベル 0 のボックスはトレースされない。

処理系はボディボックスに対応するヘッドをプログラム中から捜し出し、それが存在すれば、ヘッドボックスが生成され HCALL が発生する。ヘッド・ユニフィケーションが成功すると HEXIT となり、つぎにそのボディ部のサブゴールの実行を行う。サブゴールの実行のためにボディ・ボックスが生成され、 BCALL が発生する。

最初のボディ・ボックスのレベルを N とするとそれにより生成されたヘッドボックスとボディボックスのレベルは N+1 となる。また、処理系はヘッド・ユニフィケーションの失敗やボディ部のサブゴールの失敗により別解の探

索を行う。新しいヘッドを見付けてヘッドボックスが生成されるが、このボックスのレベルは先程のボックスと同じレベルのN+1となる。デーモンボックスのレベルはその起動のきっかけとなったボックスのレベルに1を足したレベルとなる。注意しなければならないのは、クローズ内ORのレベルである。OR内のゴール列のレベルはそのヘッドのレベルに1を足したレベルとなる。

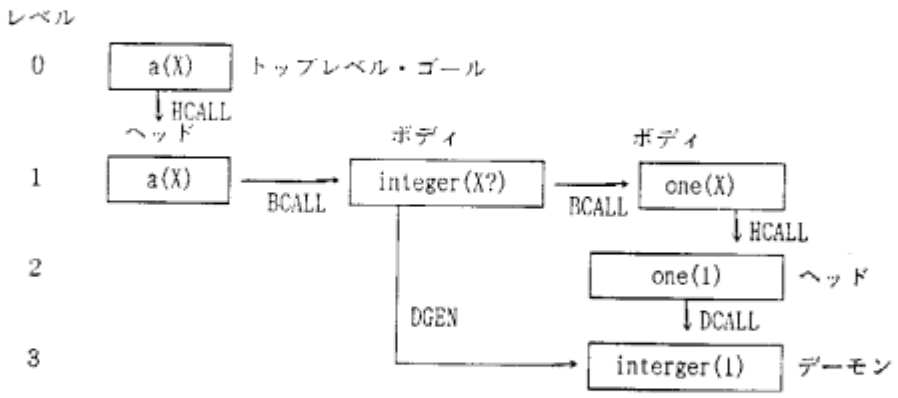


図 2.23: ボックス・レベル

デバッグ方式

ゴールの実行過程において、レベルに着目してボックス間の関係を見ると、ボックスの間には親、子、兄弟の関係がある。ある述語が正しくない場合として二つの現象がある。一つは失敗する場合であり、もう一つは成功したが具体化した変数の値が正しくない場合である。その原因を調べるためには、その述語を親とする一つ下のレベルの兄弟が正しいかをまず確かめる必要がある。正しく動作しない述語が判明すれば、つぎにその述語を親とする一つ下の兄弟を調べる。これを繰り返すことによって、正しく動作しない原因を突き止めることが出来る。

CILのデバッグ支援のリーシュ・コマンドは上記の方式を用いてデバッグを行うことを前提として、Prologとは異なる概念が用いられている。リーシュ・コマンドは次のリーシュのレベル番号が上か下か同じかを指定するコマンドとなっている。従来のリーシュ・コマンドでは、バックトラック時のskipや深くトレースしていった時に元に戻る場合に操作が煩わしくなるという問題がある。しかし、CILのリーシュ・コマンドを用いれば、誤って調べたい述語と別の子供に入ってしまっても、いったん親に戻り別の兄弟を調べることが出来る。レベル番号に着目したリーシュ・コマンドはボックス・モデルとも一致し、EXITとFAILの両ゲートで再実行することが出来るコントロール・コマンドのretryとこのリーシュ・コマンドを使えばShapiroのDivide and Query方式に近い操作が可能となる。

2.3.3 デバッグの開始と終了

インタプリタ・コマンドのdebug_programを指令すると、デバッグ・メニューが表示される。デバッグ・メニューには現在コンサルトされているソース・プログラムの一覧が表示される。更にその中でデバッグ中のプログラムは反転表示されている。デバッグ・メニューはマルチセレクト・オンオフ・スクローリング・ポップアップ・メニューである。

デバッグの開始を指定するには、反転されていないプログラムをマウスによって反転することによって行う。またデバッグの終了を指定するには、反転しているプログラムをマウスによって反転していない状態にすることによって行う。そしてDO_IT項目をマウスクリックすることにより処理が開始される。処理を行わない場合にはABORT項目を選択する。

デバッグの開始を指定するとプログラム毎に作成するデバッグ支援ウインドウ の位置を聞いてくるので、マウスによって指定する。デバッグの終了を指定するとそのプログラムのデバッグ支援ウインドウが消去される。

ウインドウ・タイトルには選択されたプログラム名が表示される。左側のメニューをデバッグ・コマンド・メニューと呼ぶ。デバッグ・コマンド・メニューはシングルセレクト・スクローリング・メニューとなっていてデバッグ・コマンドの一覧を表示している。また右上のウインドウをゴール・ウインドウと呼び、右下のウインドウをソース・ウインドウと呼ぶ。

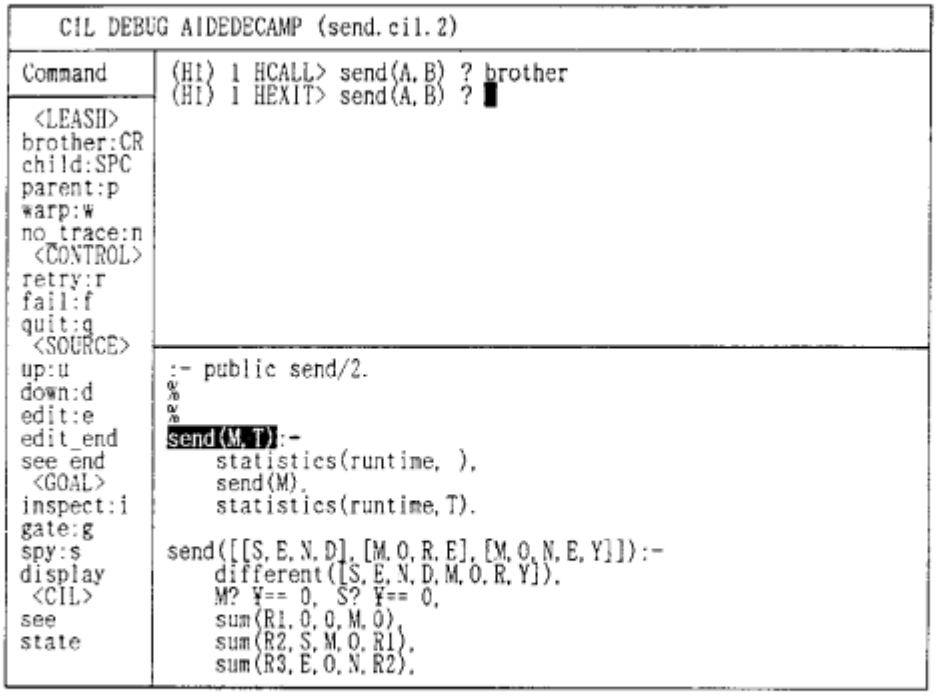


図 2.24: デバッグ支援ウインドウの例

トップ・レベルからゴールを入力すると、参照された述語のあるプログラムのデバッグ支援ウインドウがアクティベートされ、実行中の述語がソース・ウインドウ上で反転表示される。そしてゴール・ウインドウにはトレース・ゴールが表示されデバッグコマンドの入力待ちとなる。ここで各種デバッグ・コマンドを用いてデバッグを行う。またゴール・ウインドウでは Pmacs コマンドが使用可能であり、前ページのトレース状態を見たり、カーソルの移動が出来る。誤りが判明したならば、ソースプログラムの編集を行い再度実行の確認を行う。ソース・プログラムの編集はソース・ウインドウ上で行うことができる。

2.3.4 トレースとリーシュ

トレース・ゲートとリーシュ・ゲート

トレース・ゲートとは、各ボックスの各ゲートでトレース（表示）を指定したゲートのことである。実行中にトレース・ゲートを通過するとそのゴールが表示される。

リーシュ・ゲートとはトレース・ゲートのうち、リーシュ（中断）を指定したゲートのことである。実行中にリーシュ・ゲートを通過するとそのゴールを表示後、デバッグコマンドの入力が要求される。ゲートのトレースとリーシュの設定は gate コマンドで行う。

スパイ・ポイント

スパイ・ポイントとはスパイとして設定された述語のことである。特定の述語までの不要なトレース情報をスキップするために使用する。リーシュ・コマンドの warp を使用するとスパイ・ポイントのゲートを通過するまでのリーシュ・ゲートやトレース・ゲートは抑制される。スパイの設定は spy コマンドで行う。

コントロールＣによるリーシュ

デバッグ支援ウインドウに対して任意の時点でコントロールＣを入力すると，その時点での実行中のゲートがリーシュ・ゲートとなる。この機能は no_trace 中でも有効であり、実行時ループを調べるのに有効である。

トレース出力とリーシュ入力

トレースの表示形式は、最初のかっこ内が箱識別番号であり、次の数字が呼出しのレベルである。その次の>までの文字がゲート名を示し、つぎに述語が表示される。箱識別番号は先頭に各ボックスのイニシャル (H,B,D,E) が

つき、その後に数が続く形式となっている。

リーシュ・ゲートの場合は入力プロンプトとして?が表示され、キーボードからの一文字入力か, マウスによるメニュー選択のいずれかによりデバッグ・コマンドを指定することができる。

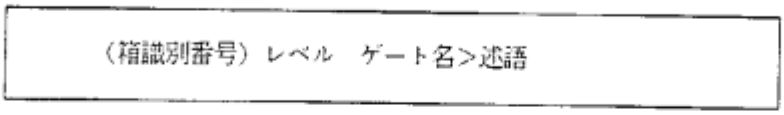


図 2.25: トレースの表示形式

2.3.5 デバッグ・コマンド

リーシュに対するコマンド

次のリーシュ・ポイントを指定するコマンド群である。

1. brother コマンド
   現在のレベルと同じレベルのリーシュゲートを次のリーシュ・ポイントとする。その間は下位レベルのトレース・ゲートとリーシュゲートは抑制される。同じレベルのゲートの実行の最後の場合は次の上位レベルのゲートがリーシュ・ポイントとなる。全ゲートで適用可能である。キーボードからは <cr> または b を入力すると brother と解釈される。このコマンドはそのボックスが正しく動作するかどうかをその結果のみによって調べるときに使う。

2. child コマンド
   現在のレベルより下位レベルのリーシュゲートを次のリーシュ・ポイントとする。下位レベルのリーシュゲートが存在しない場合は同じレベルまたは上位レベルのゲートが次のリーシュ・ポイントとなる。全ゲートで適用可能である。キーボードからは <space>または c を入力すると child と解釈される。このコマンドはあるボックスの結果がおかしい時にそのボックス内を更に調べていく時に用いる。

3. parent コマンド
   現在のレベルより上位レベルのリーシュゲートを次のリーシュ・ポイントとする。全ゲートで適用可能である。キーボードからは p を入力すると parent と解釈される。あるボックスの結果がおかしい時にそのボックス内を更に調べていって問題のない所に入っていった時にいったん上位レベルに戻りまた別の下位レベルを調べる必要がある。そのときにこのコマンドを用いる。またループ時にコントロールによってリーシュ・ポイントにした場合にその親を調べるときにも有効である。

4. warp コマンド
   次のスパイ・ポイントまでのリーシュ・ゲートとトレース・ゲートを抑制して実行する。全ゲートで適用可能である。キーボードからは w を入力すると warp と解釈される。

5. no_trace コマンド
   トップ・レベルまでトレース・ゲート、リーシュ・ゲートを抑制して実行する。全ゲートで適用可能である。キーボードからは n を入力すると no_trace と解釈される。

コントロールに対するコマンド

リーシュ・ポイントで、ゴールの制御の流れを変えるコマンド群である。

1. fail コマンド
   その述語を失敗させる。適用ゲートは CALL と REDO である。 キーボードからは f を入力すると fail と解釈される。

2. retry コマンド
   その述語を再実行させる。適用ゲートは EXIT と FAIL である。キーボードからは r を入力すると retry と解釈される。 brother コマンドで結果が不正またはフェイルしたときにそのボックスの下位レベルを調べるためにこのコマンドを使う。

3. quit コマンド
   現在実行中の状態を放棄してインタプリタに戻ることを指定する。全ゲートで適用可能である。キーボードからは q を入力すると quit と解釈される。

ソースに対するコマンド

1. up コマンド
   ソース・ウインドウを上にスクロール・アップする。キーボードからは u を入力すると up と解釈される。

2. down コマンド
   ソース・ウインドウを下ににスクロール・ダウンする。キーボードからは d を入力すると down と解釈される。

3. edit コマンド
   ソース・プログラムを編集する。キーボードからは e を入力すると edit と解釈される。edit コマンドを指定すると、ソース・ウインドウが入力可能状態となる。ソース・ウインドウは Pmacs ウインドウなので各種 Pmacs コマンドが使える。ソース・プログラムを修正しなければ、プログラムの編集を終了してもデバッグを続行できるが、修正すると、プログラムのリコンサルトが起こり、強制的にトップレベルに戻る。編集したプログラムに文法エラーがあると、リコンサルトは失敗して編集モードに戻る。文法エラーが取れるまでこれは繰り返される。

4. edit_end コマンド
   edit コマンドで始まったプログラムの編集を終了する。

5. see_end コマンド
   トップ・レベルからの see_debug コマンドまたは他のデバッグ支援からの see コマンドを終了する。

ゴールに対するコマンド

1. inspect コマンド
   CIL インスペクタを呼び出す。全ゲートで適用可能である。キーボードからは i を入力すると inspect と解釈される。

2. gate コマンド
   ゲートのトレースとリーシュ を設定するためにゲート・メニューを呼び出す。全ゲートで適用可能である。キーボードか らは g を入力すると gate と解釈される。このコマンドを指定するとゲート・メニューが表示される。ゲート・メニューはポップアップ・マルチセレクト・メニューである。反転しているゲートがトレース、リーシュの対象となっていることを表す。各ゲートをマウスで選択すると、on は off に、off は on になる。また、box 項目をマウスで選択すると box 内の全てのゲートが on/off する。最後に EXIT を選択するとゲートが設定され、ゲートメニューが消える。このメニューはトップ・レベルからも呼び出すことができる。

3. spy コマンド
   スパイ・ポイントを設定するためにスパイ・メニュー を呼び出す。全ゲートで適用可能である。キーボードからは s を入力すると spy と解釈される。このコマンドを指定するとスパイ・メニューが表示される。スパイ・メニューのラベルにはプログラム名が表示されている。スパイ・メニューはポップアップ・マルチセレクト・スクローリング・メニューであり、反転している述語がスパイ・ポイントとなっている述語である。マウスで選択すると on は off に、off は on になる。最後に DO_IT を選択するとスパイが設定され、スパイ・メニューが消える。設定しない場合は ABORT を選択する。このメニューはインタプリタからも呼び出すことができる。

   複数のプログラムをデバッグ中のときはスパイ・メニューが表示される前にどのプログラムのスパイ・ポイントを設定するのかの確認が行われる。デバッグ支援ウインドウが順番にアクティベートされて確認メニューが現われるので、yes か no かを指定する。

4. display コマンド
   現在のゴールを再表示する。全ゲートで適用可能である。キーボードからは指定が出来ない。state コマンドでホロフラスティング・レベルをトレース中に変更した後現在のゴールを再表示するコマンドである。

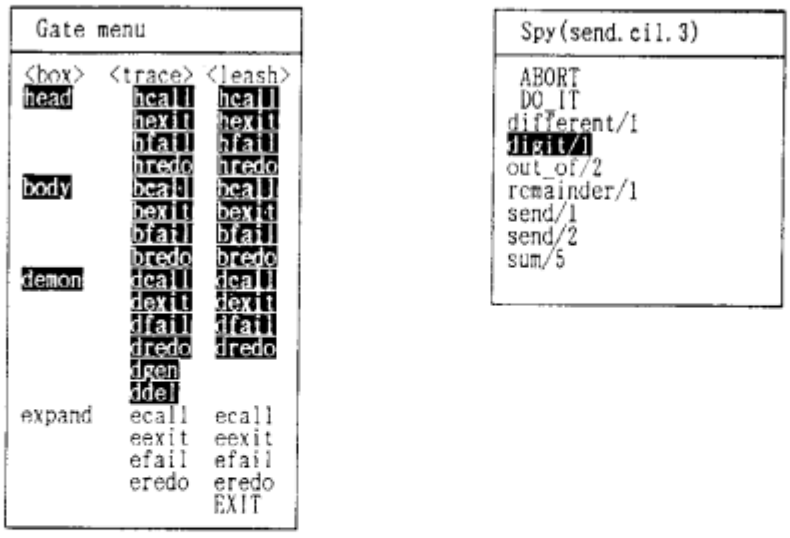


図 2.26: ゲート・メニューとスパイ・メニュー

処理系に対するコマンド

1. see コマンド
   デバッグ中に他のデバッグ中のプログラムを参照したいときこのコマンドを指定する。デバッグ支援ウインドウが順番にアクティベートされ、どのプログラムをみたいのかをどうかを確認してくるので yes か no を指定する。see 中にはソースに対するコマンドしか受付けない。プログラムの参照が終わったら、see_endコマンドで元に戻る。

2. state コマンド
   CIL の各種の状態を変更するためにステート・メニューを呼び出す。全ゲートで適用可能である。キーボードからは指定できない。呼び出し以後の設定についてはインタプリタ・コマンドの章を参照のこと。

2.4 インスペクト時の操作

2.4.1 インスペクタの概要

インスペクタの機能

CIL インスペクタ（以下、単にインスペクタと呼ぶ）は、CIL プログラムのデバッグツールとして、CIL データ型の値をウインドウ上に表示し、デバッグの支援を行う。

インスペクタは、次の様な機能、特徴を持つ。

1. CIL の各データの構造の解析、および、値の表示機能
2. 構造データには、ノード番号を振り、同一の構造をノード番号で指示する機能
3. デーモン付変数のデーモンの表示機能
4. 一度インスペクトしたデータを覚えておく、メモリ機能
5. ウインドウとマウスによる簡単な操作
6. ESP データ型にも対処

データ型

インスペクタで取り扱うデータを CIL データ型と呼び、その分類は以下のとうりである。CIL データ型には、ESP データ型も含まれている。

CIL データ型		
	アトミック	アトム……atom
		整数……integer
		浮動少数点数……float
	ストリング	ビット列……string
		バイト列……string
		2バイト列……string
		4バイト列……string
	構造型	ターム……term
		部分項……partial
		リスト……list
		スタックベクタ……stack
	変数	デーモン付……freezed
		デーモンなし……unbound
	その他	ヒープデータ……heap
		オブジェクト……object
		コード……code
		ロケーション……location
		未知……unknown

図 2.27: CIL データ型

ノード番号

インスペクタでは、データの各ノードにノード番号を割り振り、データ内の同じ構造の部分をノード番号によって指示している。ノード番号は、識別子 (id) とレベル番号 (n) によって次のように示される。

id:n1:n2:n3...

1. 識別子
   次の三つがある。

   - p : トレース中の述語を示す。
   - vN: CIL トップレベルで使用されている変数を示す。
     N は、インスペクタが変数に対して任意に割り振った番号
   - dN: デーモンを示す。
     N は、デーモンが発生した時にデバッグ支援によって示される箱識別番号

2. レベル番号

   レベル番号は、構造を下向きの木に見立てて割り振られる。 同じ深さの枝は、左から順に割り振られる。更に深く枝がある時は、 コロン (:) に引き続いて該当する番号が割り振られる。レベル番号は 0 オリジンで構造の要素に割り振られるが、タームの場合は、 1 オリ ジンで引数に割り振られる。

   以下にノード番号の振り方の例を示す。括弧内がノード番号を示している。

図 2.28: ノード番号の例

2.4.2 インスペクタの起動と終了

インタプリタ・コマンドの inspect_variable またはデバッグ・コマンドの inspect を選択するとインスペクタが起動される。最初のときは、ウインドウを表示する位置を聞いてくるので、マウスで指定するとインスペクタ・ウインドウが表示される。以後の呼出しでは、その位置に表示される。インスペクトを終了するときは、コマンド・メニュの exit を選択する。

インスペクタ・ウインドウの左側のウインドウをディスプレイ・メニューと呼ぶ。データを解析した結果は、このウインドウに表示される。表示された項目をマウス・セレクトすることにより、更に深く構造を調べることができる。ディスプレイ・メニューには最初にトレース中の述語が表示されている。

ディスプレイ・メニューのラベルには解析したデータ型とノード番号 が示される。表 2.1に型表示の形式とデータ例を示す。

インスペクタ・ウインドウの右側のウインドウをメモリ・メニューと呼ぶ。ディスプレイ・メニュー上で選択された項目は、このウインドウに登録される。登録された項目をマウス・セレクトすることにより、更に深く構造を調べることができる。メモリ・メニューには最初に Variable と Predicate が登録されている。 Variables をマウス・セレクトすることにより、トップレベル変数を調べることができる。また、 Predicate をマウス・セレクトすることにより、トレース中の述語を調べることができる。

インスペクタ・ウインドウの下側のウインドウをコマンド・メニューと呼ぶ。 exit, state, delete memory の三つのコマンドを持っている。

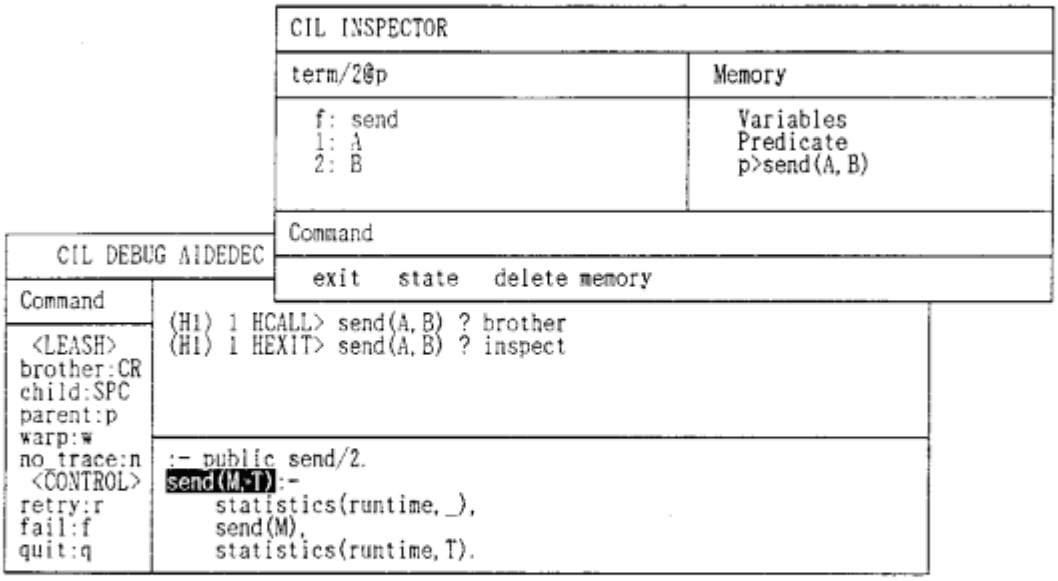


図 2.29: インスペクタの起動

2.4.3 データのインスペクト

メモリ・メニュー上には、あらかじめ Variables と Predicate という項目が設定されている。Variables は、トップレベルの変数、Predicate は、トレース中の述語が対応付けられている。ただし、インタプリタ・コマンドの inspect_variable で起動した場合は Predicate 項目がない。

ディスプレイ・メニュー上には、トップレベルから起動したときは、トップレベルの変数が設定されていて、デバッグ支援から起動したときは、述語が設定されている。

データのインスペクトは、以下の要領で行う。

1. メモリ・メニュー上の調べたい項目にマウスを使ってカーソルを移動する。

2. 項目全体がカーソルで囲まれたら、マウスをクリックする。

3. 選択した項目が新たにディスプレイ・メニュー上に表示される。

4. ディスプレイ・メニュー上の調べたい項目にマウスを使ってカーソルを移動する。

5. 項目全体がカーソルで囲まれたら、マウスをクリックする。

6. 選択した項目をインスペクトした結果が、新たにディスプレイ・メニュー上に表示される。

7. ディスプレイ・メニュー上で選択された項目はメモリ・メニュー上に登録される。メモリ・メニュー上の項目を選択することにより、再度その項目をインスペクトすることができる。

表 2.1: ディスプレイ・メニューのラベル

データ型	型表示の形式	データ例
アトム	atom@N	abc, 'You'
整数	integer@N	12, 0, -99
浮動少数点	float@N	3.14, -1.0^-4
ビット列	string@N	bits#{0,1,0,1}
バイト列	string@N	bytes#{0,FE,A0}
2 バイト列	string@N	"aAbB"
4 バイト列	string@N	words#{0,FFFFFFFF}
ターム	term/A@N	a(b,c)
部分項	partial/E@N	{a/1,b/2}
リスト	list/E@N	[1,2,3]
スタックベクタ	stack/E@N	{X,Y,Z}
変数	variable@N	X, _1
オブジェクト	object@N	$list, #test
未知	unknown@N	

ここで N はノード番号、A は引数の数、E は要素数である。

2.4.4 デーモンのインスペクト

デーモン付変数をインスペクトすると、その変数に登録されているデーモンが表示される。さらに、そのデーモンを選択することによってデーモンの引き数を調べることができる。

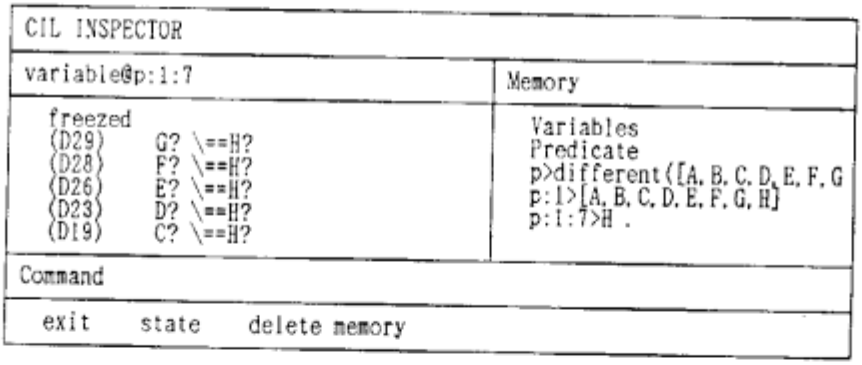


図 2.30: デーモン付変数の表示

2.4.5 デリート機能

メモリ・メニュー上に登録された項目は不要ならば、削除することができる。以下にメモリ・メニュー上の項目を削除する要領を示す。

1. コマンド・メニューの delete memory を選択すると、メモリ・メニューの位置にデリート・メニューが表示される。デリート・メニューはスクローリング・マルチセレクト・メニューである。
2. メニュー上の削除したい項目にマウス・カーソルを移動させ、マウスをクリックすると、その項目が反転表示される。これを繰返す。
3. DELETE をマウス・クリックすると、反転表示されている項目が削除されメモリ・メニューに戻る。削除を中止する場合は ABORT をマウス・クリックする。ここで DELETE か ABORT を選択してデリート・メニューから脱け出ないうちはデリート・メニュー以外からの入力は受付けられない。

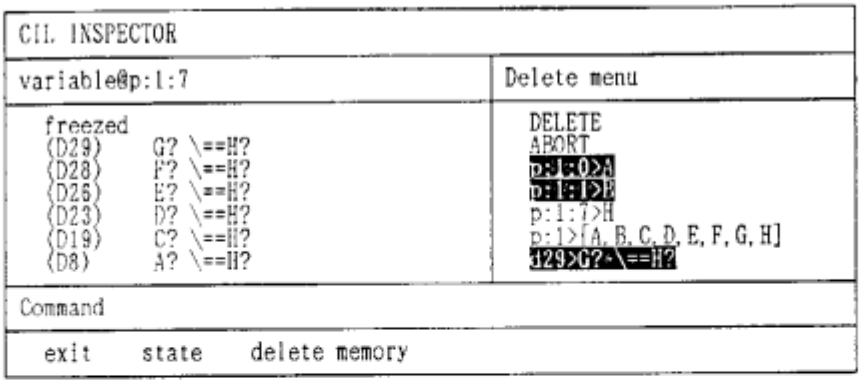


図 2.31: デリート・メニューの例

2.4.6 パラメータの変更

インスペクタは、構造型データ (term,partial,list,stack) とストリングを解析し表示する場合、その開始位置と長さをパラメータとして持っていてる。コマンド・メニューの state を選択することによりそのパラメータを変更することができる。

start パラメータは解析を開始する要素のオフセットである。例えば、1000 個の要素を持つリストの場合、start を 500 に設定することにより、500 番目の要素以降を解析し表示する。start の既定値は 0 である。

length パラメータは解析する要素の長さを示す。例えば、1000 個の要素を持つリストの場合、length を 10 に設定することにより、リストの start の値から 10 個の要素を解析し、表示する。length の既定値は 200 である。

パラメータの変更は、以下の要領で行う。

1. コマンド・メニュー上の state をマウスクリックにより選択すると、ステート・メニューが現われる。ステート・メニューはテンポラリ・チョイス・メニューである。
2. 設定したいパラメータにマウスを使ってカーソルを移動すると、数字の部分がカーソルで囲まれる。
3. カーソルで囲まれた状態でマウスをクリックすると、キーボードからの入力が可能になる。設定したい値をキーボードから入力し、リターンキーを押す。
4. 値を入力した後、do_it をマウスをクリックすると、入力値が設定される。中止する場合は、abort をクリックするかまたはカーソルをステート・メニューから外へ出す。

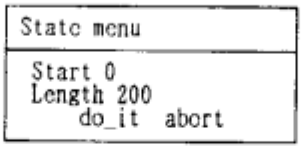


図 2.32: ステート・メニュー

2.5 ツールボックスの操作

2.5.1 ツールボックスの概要

ツールボックスはユニットの特殊なもので、ツールボックスに登録されたプログラムは、同一PSI上であればユーザの区別なくどのユニットからでも呼びだす事ができる。ツールボックスはユニットとは異なる独自のプログラム管理機能を持つ。

以下に主な管理機能を示す。

- ユニット内のプログラムをツールボックスに登録する。
- ツールボックス内のプログラムの登録を抹消する。
- ツールボックス内のプログラムをコンサルトする。
- ツールボックス内のプログラムをユニットに複写する。
- ツールボックス内のプログラムの述語を照会する。
- ツールボックス内のプログラムのソース・テキストを表示する。

ツールボックスにはデバッグ機能はない。ツールボックス内のプログラムは登録時にコンパイルされているのでデバッグする事はできない。ツールボックスにプログラムを登録する時は、ユニット内でデバッグが完了してから登録しなければならない。また、既に登録済みのプログラムをデバッグする必要がある時には、プログラムをユニットに転記してユニット内でデバッグを行ってから再登録しなければならない。

ツールボックスはプログラムのファイルおよびクラスの管理を行う。管理資源としてファイルはディレクトリ>sys>cil_toolbox、クラスはパッケージ cil_toolbox を使用する。これらの資源はCIL起動時に自動的にチェック／生成される。

2.5.2 ツールボックス内のプログラムの使い方

ツールボックスに登録されているプログラムは、プログラムからの呼び出し、コンサルト、ソースの流用の3つの方法で利用するものとする。

1. プログラムから呼びだす方法
   ツールボックス内のプログラムをユーザのプログラム中で使う場合はインクルード宣言 を使用する（インクルードに関する詳細は「インクルード宣言」を参照のこと）。インクルードするプログラムは"toolbox/ ファイル識別子"で指定する。また、ESPのクラスの特定は明示指定を使用する。

   例
   ```
   :-include_program(toolbox/discourse).
   :-include_declaration(toolbox/my_sugar).

   my_assert(X):-
         :my_assert(#cil_toolbox##database, X).
   ```

2. コンサルトする方法
   ツールボックス内のプログラムを直接単独で動かす場合は、プログラムをコンサルトする。コンサルトするには、consult_toolboxコマンドを使用する。

3. ソースを流用する方法
   ツールボックス内のプログラムを流用する場合は、プログラムのソースをユニットに複写する。複写は、copy_toolboxコマンドを使用する。

2.5.3 ツールボックス・コマンド

1. catalog_toolbox コマンド

このコマンドは、ユニット内のプログラムをツールボックスに登録する時に使用する。コマンド・メニューのTOOLBOX をマウスセレクトすると、サブメニューが表示される。その中の catalog をセレクトすると、現在選択されているユニット内のファイル一覧メニューが現れる。登録の対象となるのはファイルタイプが esp, dcl, cil のファイルで、メニューにはこれらのファイルが示される。このメニューはマルチセレクトメニューになっているので登録するプログラムをすべてセレクトした後、DO_IT をクリックする。

この時、インクルードに必要なファイルやメソッド呼び出しに使用している ESP ファイルも忘れずに指定しなければならない。

登録は、ファイルタイプが esp,dcl,cil の順に行われる。

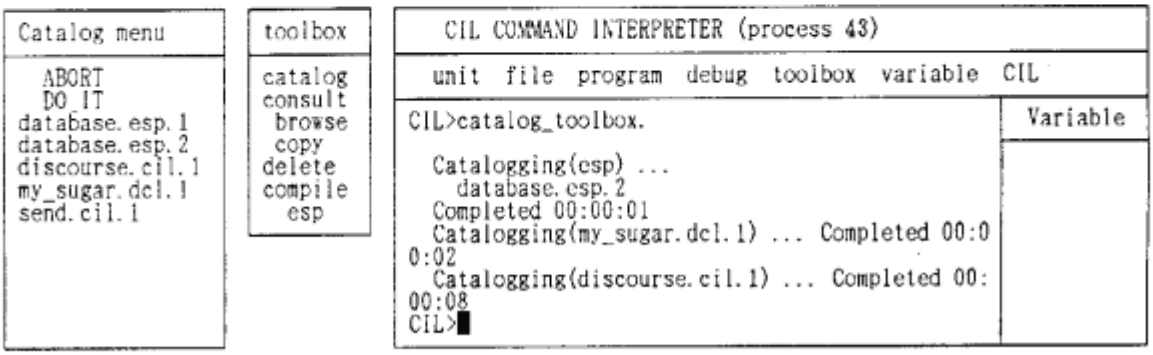


図 2.33: catalog_toolbox コマンドの例

2. consult_toolbox コマンド

このコマンドは、ツールボックスに登録されているプログラムをコンサルトする時に使用する。

操作仕様は consult_program と同じである。コマンド・メニューの TOOLBOX をマウスセレクトすると、サブメニューが表示される。その中から consult をセレクトすると、現在登録されている CIL プログラム一覧メニューが現れる。この時、既にコンサルトされているプログラムは反転表示されている。このメニューはマルチセレクトメニューになっているので複数のプログラムをセレクトする事ができる。

プログラムをコンサルトする場合は、メニュー上のコンサルトしたいプログラムをセレクトした後、DO_IT をクリックする。コンサルトされているプログラムを解除する場合は、解除したいプログラムをクリックして反転表示を解除した後、DO_IT をセレクトする。

ツールボックスに登録されたプログラムはすべてコンパイルされており、コンパイルド・プログラムしかコンサルトできない。

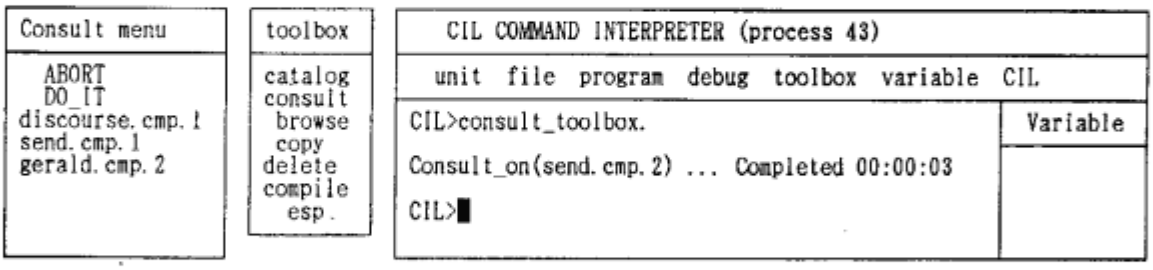


図 2.34: consult_toolbox コマンドの例

3. browse_toolbox コマンド

このコマンドは、ツールボックスに登録されている述語の照会と、プログラムのソース・テキストを表示する時に使用する。

コマンド・メニューのTOOLBOXをマウスセレクトすると、サブメニューが表示される。その中からbrowseをセレクトすると、ウインドウの位置と大きさを聞いて来る。マウスでこれらを指定するとツールボックス・ブラウザが現れる。 ツールボックス・ブラウザのウインドウには最初現在登録されているプログラムのファイル一覧メニューが表示される。このメニューはシングルセレクトメニューになっている。 exitをクリックするとトップレベルに戻る。

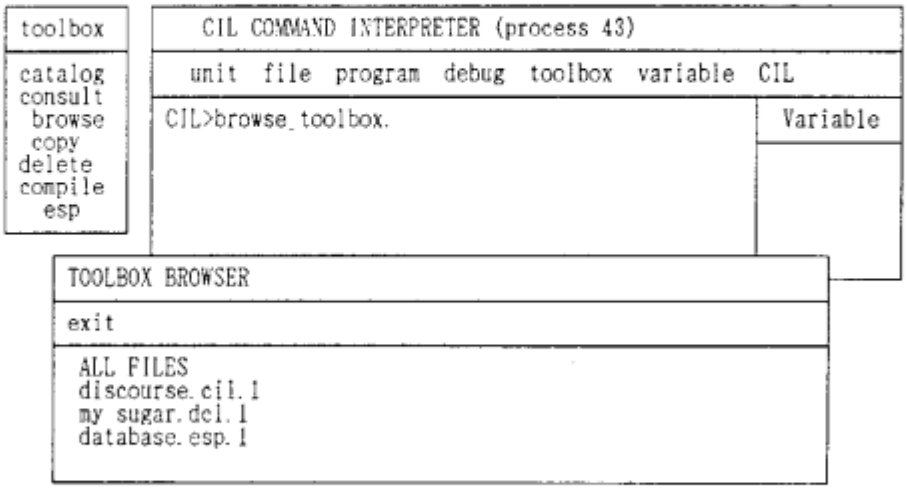


図 2.35: ファイル一覧メニューの例

(a) 述語の参照

ファイル一覧メニューでALL FILES、または、ファイルタイプがcilのファイルをセレクトする。ALL FILESの時はツールボックス中のすべての述語、ファイルをセレクトした時はそのファイル中に定義されている述語の述語一覧メニューが現れる。

述語一覧メニューの述語項目は、"述語名/引数の数@ファイル識別子"の形をしている。このメニューはシングルセレクトメニューになっていて項目をセレクトすると、その述語が定義されているプログラムのソーステキストが表示される。 exitをクリックすると一つ前の状態に戻る。

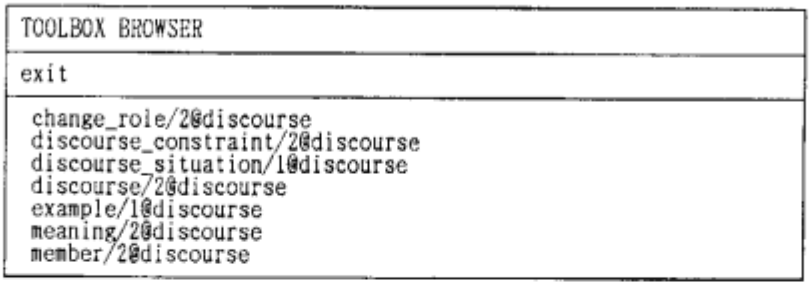


図 2.36: 述語一覧メニューの例

(b) ソーステキストの表示

ファイル一覧メニューでファイルタイプがcil以外のファイルをセレクトするか、または、述語一覧メニューで述語項目をセレクトするとソーステキストを表示したテキスト表示ウインドウが現れる。

このウインドウはPmacs機能付きウインドウになっているので、文字検索その他のPmacsコマンドを使用する事ができるが、リードオンリーなので内容を変更する事はできない。exitをクリックすると一つ前の状態に戻る。

4. copy_toolbox コマンド

このコマンドは、ツールボックス内のプログラムを現在選択されているユニットに複写する時に使用する。コマンド・メニューのTOOLBOXをマウスセレクトすると、サブメニューが表示される。その中からcopyを

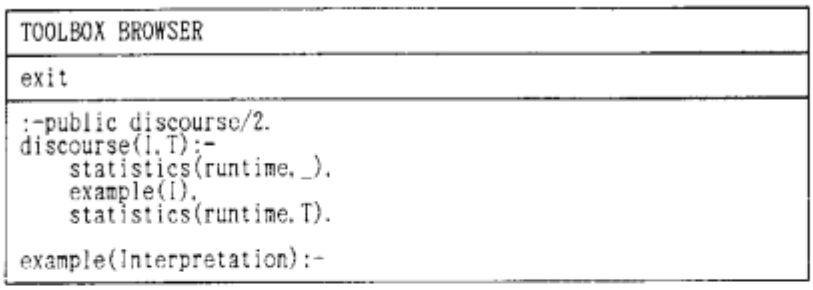


図 2.37: ソーステキストの表示例

セレクトすると、現在登録されているプログラム一覧メニューが現れる。このメニューはマルチセレクトメニューになっているので複写するプログラムをすべてセレクトした後、DO_IT をクリックする。

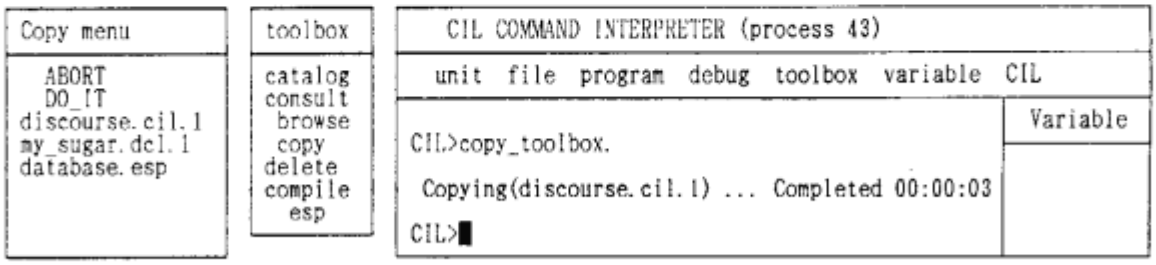


図 2.38: copy_toolbox コマンドの例

5. delete_toolbox コマンド

このコマンドは、ツールボックス内のプログラムの登録を抹消する時に使用する。コマンド・メニューの TOOLBOX をマウスセレクトすると、サブメニューが表示される。その中から delete をセレクトすると、現在登録されているプログラム一覧メニューが現れる。このメニューはマルチセレクトメニューになっているので登録を抹消するプログラムをすべてセレクトした後、DO_IT をクリックする。

6. compile_toolbox コマンド

このコマンドは、ツールボックス内の CIL プログラムをコンパイルする時に使用する。コマンド・メニューの TOOLBOX をマウスセレクトすると、サブメニューが表示される。その中から compile をセレクトすると、現在登録されているプログラム一覧メニューが現れる。このメニューはマルチセレクトメニューになっているのでコンパイルするプログラムをすべてセレクトした後、DO_IT をクリックする。

7. esp_toolbox コマンド

このコマンドは、ツールボックス内の ESP プログラムをカタログする時に使用する。

コマンド・メニューの TOOLBOX をマウスセレクトすると、サブメニューが表示される。その中から esp をセレクトすると、現在登録されているプログラム一覧メニューが現れる。このメニューはマルチセレクトメニューになっているのでカタログするプログラムをすべてセレクトした後、DO_IT をクリックする。

2.6 メッセージ一覧

2.6.1 コマンド・インタプリタ・メッセージ

コマンド・インタプリタが表示するメッセージの意味とその処置の一覧を示す。

メッセージ:
Can not add operator(xxx,yyy,zzz)
意味:
オペレータの追加ができません（名前 xxx, 型 yyy, 優先度 zzz ）
処置:
不正なオペレータが宣言されているので、修正する。

メッセージ:
Can not consult same ident name program(xxx)
意味:
同じファイル識別子のプログラム（xxx）は同時にコンサルトできません
処置:
前にコンサルトしたプログラムを解除してからコンサルトする。

メッセージ:
Can not include same ident name program(xxx)
意味:
識別子（xxx）が同じプログラムをインクルードすることはできません
処置:
前にコンサルトしたプログラムを解除してからインクルードする

メッセージ:
Can not open (xxx)
意味:
ファイル (xxx) がオープンできません
処置:
ファイル名を確認し、正しいファイル名に修正する。

メッセージ:
Can not remove operator(xxx)
意味:
オペレータ xxx は削除できません
処置:
不正なオペレータが宣言されているので、修正する。

メッセージ:
Debugging program is nothing
意味:
デバッグ中のプログラムはありません
処置:
デバッグ オンにする。

メッセージ:
Illegal format (xxx) in external
意味:
イクスターナル宣言（xxx）が不正です

処置:
　正しく書きなおす。

メッセージ:
　Illegal format (xxx) in public
意味:
　パブリック宣言 (xxx) が不正です
処置:
　正しく書きなおす。

メッセージ:
　Illegal head in
　xxx
意味:
　以下のクローズ中のヘッドが不正です
　xxx
処置:
　正しい述語に修正する。

メッセージ:
　Illegal term
　xxx
意味:
　不正な項があります
　xxx
処置:
　括弧の始まりや終わりが合っていない場合や演算子が正しく使用されていない場合などがある。 項を正しい書き方にする。

メッセージ:
　Include_declaration in include_declaration
意味:
　宣言のインクルードファイル中に宣言のインクルードはできません
処置:
　宣言のインクルードファイル中の宣言のインクルードをやめる。

メッセージ:
　Include_program in include_declaration
意味:
　宣言のインクルードファイル中にプログラムのインクルードはできません
処置:
　プログラムのインクルードは宣言のインクルードファイルの外にだす。

メッセージ:
　Load fail(xxx)
意味:
　xxx のロードに失敗しました
処置:
　該当プログラムを再コンパイルする。

メッセージ:

Open fail(xxx)
意味:
ファイル(xxx)のオープンに失敗しました
処置:
正しいファイル名に修正する。

メッセージ:
Open fail program(xxx)
意味:
プログラム(xxx)がオープンできません
処置:
ファイル名が正しいか確認し、修正する。

メッセージ:
Open fail declaration (xxx)
意味:
宣言(xxx)のオープンに失敗しました
処置:
ファイル名が正しいか確認して、修正する。

メッセージ:
Program (xxx) is consulted
意味:
コンサルト中のプログラム(xxx)は削除できない。
処置:
コンサルトを解除してから削除する。

メッセージ:
Syntax_sugar definition must begin with 'define_syntax_sugar'
意味:
シンタックス・シュガーの定義の開始はdefine_syntax_sugarにしてください
処置:
シンタックス・シュガーの定義はdefine_syntax_sugarで始める

メッセージ:
Syntax_sugar definition must end with 'end'
意味:
シンタックス・シュガーの定義の終わりはendにしてください
処置:
endをつける。

メッセージ:
Syntax_sugar xxx is defined duplicately
意味:
シンタックス・シュガーの定義のxxxは二重定義です
処置:
OR形式のオルタナティブに修正する。

メッセージ:
This clause is not declaration(xxx)
意味:

このクローズ (xxx) は宣言ではありません
処置:
宣言ファイル中には宣言クローズしか書けない。
宣言ファイルの外にだす。

メッセージ:
Unbound clause(xxx) input
意味:
プログラムの先頭から xxx 番目のクローズが変数です
処置:
正しいクローズに修正する。

メッセージ:
Unit name over 10 characters
意味:
ユニット名は 10 文字以内にしてください
処置:
ユニット名を 10 文字以内にする。

メッセージ:
xxx/yyy becomes KL0, two arguments added by compiler in
zzz
意味:
以下のクローズ中の述語 (xxx/yyy) はコンパイラによって引き数が 2 追加され
KL0 と同じとなるため定義できません
xxx
処置:
述語名を変更する。

メッセージ:
xxx/yyy is CIL BUILTIN in
zzz
意味:
以下のクローズ中の述語 (xxx/yyy) は CIL 組込述語と同じですので定義できません
zzz
処置:
述語名を変更する。

メッセージ:
xxx is duplicated declaration
意味:
xxx は二重宣言です
処置:
どちらかを取り除く。

メッセージ:
xxx/yyy is KL0 BUILTIN in
zzzz
意味:
以下のクローズ中の述語 (xxx/yyy) は KL0 と同じですので定義できません
zzz

処置:
述語名を変更する。

メッセージ:
xxx is unknown declaration
意味:
xxx は宣言として認められません
処置:
正しい宣言に修正する。

2.6.2 シンタックス・チェッカ・メッセージ

プログラム編集操作時にシンタックス・チェックを行うと、メッセージはシンタックス・チェッカ・ウインドウに表示される。プログラム・チェック時には、コマンド・インタプリタ・メッセージと同じメッセージが表示される。クローズ・チェック時や述語チェック時に表示されるメッセージの一覧を以下に示す。

メッセージ:
nonsense variable name
xxx
意味:
以下の変数名はクローズ中で一度しか現れません
xxx
処置:
変数名が正しいかどうかチェックし、誤っている場合は修正する。
意図したとおりの変数名ならば、なるべく無名変数 _ に修正する。

メッセージ:
unbound goal is contained
意味:
アンバウンドのゴールが含まれています
処置:
ゴール列が正しく記述されているかどうかチェックし、
誤っている場合は修正する。

メッセージ:
undefined predicate xxx
意味:
xxx は未定義です
処置:
述語名が正しいかどうかチェックし、誤っている場合は修正する。

メッセージ:
unreffered predicate xxx
意味:
述語 xxx は参照されていません
処置:
述語名が正しいかどうかチェックし、誤っている場合は修正する。

2.6.3 インスペクタ・メッセージ

インスペクタが表示するメッセージの一覧を以下に示す。

2.6. メッセージ一覧

メッセージ:
Fail !!
意味:
解析に失敗しました !!
処置:
コマンド・メニュの項目に正しい値を入力する。

メッセージ:
Invalid input !!
意味:
不正な値が入力されました !!
処置:
コマンド・メニュの項目に正しい値を入力する。

メッセージ:
Invalid state(start or length) !!
意味:
現在の開始位置と長さの範囲には表示するものがありません !!
処置:
コマンド・メニュの項目に正しい値を入力する。

第3章

言語仕様

本章ではプログラミングの際に必要とされる言語仕様について述べる。

3.1 表現形式

この節ではCILの言語仕様を記述するための前準備として、文字集合、基本的言語要素および文法記述のメタ言語について説明する。

3.1.1 文字集合

CILの言語要素は文字を並べることによって記述する。そのさいに使用できる文字はJISコードで規定されている文字集合であり、以下のように分類する。

1. 数字　　0 1 2 … 9
2. 英大文字　　A B … Z
3. 英小文字　　a b … z
4. 下線　　_
5. 区切り文字　　, ; :- . ( ) [ ] { } " ' |
6. 英記号　　! @ # $ % …
7. 漢字　　あ イ 有 …
8. 書式用文字　　空白　改行文字

3.1.2 言語要素

CILのプログラムは言語要素とその間に区切り文字と任意の書式用文字を記述することによって構成する。基本となる言語要素とその書き方を以下に示す。

1. コメント
   %から改行文字までの文字の並びであり、書式用文字と同じに扱われる。
   [例] % box definition

2. 整数 <integer>
   数字よりなる文字の並びである。
   [例] 12　560

3. アトム <atom>
   先頭の文字が英大文字または下線でなく、数字・英大文字・英小文字・下線・漢字からなる文字の並びである。

   [例] abc　make_it　difAtom　花子

また ' から ' までの文字の並びもアトムとして扱われる。クオートの内側の文字の並びがそのアトムの値である。たとえば abc は 'abc' と同じである。
［例］'abc'　'ABC'　'$$void'

さらに、英記号の一文字および文字の並びもアトムである。演算子として定義されている英記号は、アトムであるが、処理系によって別の意味に解釈される場合が多いので注意を要する。
［例］!　-　/　*　+

区切り文字からなる一文字または文字の並びもアトムとして扱うことができる。また、空リストの [] もアトムである。 {} は部分項であり、アトムではない。 (X,Y) のカンマはアトムである

4. 変数 <variable>
先頭の文字が英大文字または下線であり、数字・英大文字・英小文字・下線・漢字からなる文字の並びである。

［例］ _太郎　X　Atom　Void_string _string　_

5. 文字列 <string>
" から " までの文字の並びであり, " 自身を書くときは二つ続けて書く。
［例］ "abc"　"pick_up"　"愛してる"　""""

3.1.3 メタ言語

以下の節ではプログラムの形式的定義を記述するためにメタ言語を用いる。そのさいに上記の言語要素は定義済みの文法要素として扱う。以下にメタ言語の記号と意味を示す。

	記号	意味
(1)	<xxx> ::=	xxx の定義
(2)	<xxx>	文法要素（ノンターミナル）
(3)	"xxx"	言語要素（ターミナル）
(4)	{xxx}	xxx の０回または１回の出現
(5)	{xxx} …	xxx の０回以上の繰返し
(6)	xxx, yyy	xxx に yyy が続く
(7)	xxx | yyy	xxx または yyy

3.2 プログラム

CIL のプログラムはクローズの集りで構成される。クローズには事実を記述するクローズ、規則を記述するクローズ、処理系に対する宣言を行うクローズがある。

```
<program> ::= {<clause>}  …

<clause> ::=  <fact clause> | <rule clause> | <declaration clause>

<fact clause>   ::= <head>, "."
<rule clause>   ::= <head>, ":-", <body>, "."
<head>          ::= <user predicate>
<body>          ::= <predicates>
<predicates>    ::=  "(", <predicates>, ")"
                | <predicate>, { ",", <predicates>} …
                | <predicate>, { ";", <predicates>} …
```

事実を記述するクローズは、ヘッドのみからなるホーン節を書く。ヘッドに書く述語は、組込述語（KL0 組込述語、CIL 組込述語）と述語名が同じで引き数の数が同じであってはならない。また、引き数の数に 2 をたして KL0 組込述語と同じになる述語も記述してはならない。ヘッドに書いた述語はユーザ定義の述語となる。

```
[例]  love(taro, hanako).
      man(taro).
      woman(hanako).
```

規則を記述するクローズは、ヘッドとボディよりなるホーン節を書く。ボディには幾つかの述語が書ける。ボディで記述する述語は組込述語でもユーザ定義の述語でもよい。ボディ内の述語の間を "," でつなげると、AND結合を意味し、";"でつなげるとOR結合を意味する。

```
[例]  love(X,Y):- man(X), woman(Y), !, print(love(X,Y)).
      person(X):- (man(X) ;woman(X)), !, print(person(X)).
```

3.3 宣言

処理系に対する宣言を行うクローズには、オペレータ宣言、コンパイラ用宣言、シンタックス・シュガー宣言、インクルード宣言がある。

```
<declaration clause> ::=  <operator declaration>
                        | <compiler declaration>
                        | <syntax sugar declaration>
                        | <include declaration>
```

3.3.1 オペレータ宣言

オペレータ宣言は処理系に対してオペレータの順位と型を与える。add_operatorクローズはオペレータの追加を行い、remove_operatorクローズはオペレータの削除を行う。opクローズはadd_operatorと同じである。そのファイル内で宣言をした場所の下から有効になる。既定のオペレータの型および順位の詳細は『オペレータ順位表』を参照のこと。

```
<operator declaration>  ::=    <op clause>
                        | <add_operator clause>
                        | <remove_operator clause>

<op clause>   ::= ":-", "op(",<precedence>, ",", <type>, ",",  <op name>, ")."
<add_operator clause> ::=
          ":-", "add_operator(",<op name>, ",", <type>, ",", <precedence>,")."
<remove_operator clause> ::= ":-", "remove_operator(", <op name>, ")."
<precedence>    ::= <number>
<op name>       ::= <atom>
<type>          ::= "xfx" | "xfy" | "yfx" | "fx" | "fy" | "xf" | "yf"
```

```
[例]  :-add_operator(#, xfx, 1020).
      :-remove_operator(!).
```

3.3.2 コンパイラ用宣言

コンパイラ用宣言にはパブリック宣言、イクスターナル宣言がある。コンパイルするプログラム中から他のプログラムの述語を参照するためにはイクスターナル宣言をしなければならない。コンパイルしたプログラム中の述語を参照するためにはパブリック宣言をしなければならない。

```
<compiler declaration> ::= <public declaration>
                         | <external declaration>
```

述語のパブリック宣言は述語名をアトムで指定し、引き数の数を整数で指定することにより行う。

```
<public declaration>  ::= ":-public", <public_identifiers> , "."
<public identifiers>  ::= <public identifier>, {"," , <public identifier>} …
<public identifier>   ::= <user predicate name>, "/", <arity>
<user predicate name> ::= <atom>
<arity>               ::= <integer>
```

[例]　:-public a/1, b/0.
　　:-public test/5.

述語のイクスターナル宣言はファイル識別子, 述語名, 引き数の数を指定して行う。ファイル識別子はアトムで記述し、その述語があるプログラムのファイル識別子を指定する。

```
<external declaration>  ::= ":-external(", <external_identifiers> ")", "."
<external identifiers>  ::=
                  <external identifier>, {"," , <external identifier>} …
<external identifier>   ::=
            <program identifer>, "/", <user predicate name>, "/", <arity>
<program identifier>    ::= <atom>
```

[例]　:-external(test/a/1, pool/b/0).

3.3.3　シンタックス・シュガー宣言

シンタックス・シュガー宣言はユーザがプログラミング時に使いたい記法の定義を処理系に与えるための宣言である。シンタックス・シュガーの宣言は、そのファイル内でその宣言の下にあるすべてのクローズに対して有効となる。

応用例としてはプログラム中で何度も使う表記をシンタックス・シュガーとして宣言してプログラミングに用いると有効である。特に constr 述語中の複雑な制約を何度も記述する場合にはシンタックス・シュガーを用いるとよい。

シンタックス・シュガーとして処理系に与えることの出来る文法はリカージョンを含まない正規文法である。シンタックス・シュガーにはターム・レベルのシンタックス・シュガーとクローズ・レベルのシンタックス・シュガーがある。ターム・レベルのシンタックス・シュガーはクローズ中のヘッド部、ボディ部中のタームがその対象となる。

[例] CIL 組み込みのシンタックス・シュガー

```
a(X#Y)    =>  X=Y, a(X)
X={a/1}   =>  mk_assoc(a/1,A), unify_cil(X,A)
```

クローズ・レベルのシンタックス・シュガーは、ヘッドのみからなるクローズがその対象となる。

[例] Prolog の DCG

```
s --> np, vp.    =>  s(S0,S):-np(S0,S1),vp(S1,S).
```

さらに単純なパターン展開のみではなく複雑な変換を定義するために、展開の条件を記述することができる。

```
<syntax sugar declaration> ::=
        "define_syntax_sugar", <syntax sugar definitions>, "end_syntax_sugar."
<syntax sugar definitions> ::= {<syntax sugar definition>, ";"} …
<syntax sugar definition> ::=
          <term level definition>
        | <clause level definition>
        | <condition definition>
<term level definition> ::=
          "syntax_sugar(", <replacing term>, ",", <replaced term>, ",",
             <semantic predicates>, ")" , {":-", <replacing condition>}
<clause level definition> ::=
```

```
        "clause_syntax_sugar(", <replacing clause>, ",", <replaced clause>, ")",
                {":-", <replacing condition>}
<replacing term> ::= <term>
<replaced term> ::= <term>
<semantic predicates> ::= <predicates>
<replacing clause> ::= <variable> | <head>
<replaced clause> ::=  <variable> | <head> | <head>, ":-", <body>
<replacing condition> ::= <predicates>
<condition definition> ::= <head> | <head>, ":-", <body>
```

syntax_sugar/2 の第一引き数に記述する項は置き換える項を指定する。第二引き数は置き換え後の項を指定する。 syntax_sugar/3 の第三引き数は展開の意味を指定するものであるが、単なる項の置き換えの場合は true を記述する。シンタックス・シュガー展開後において、意味述語列は項の置き換えが発生したボディ部の述語の実行の直前に実行される述語列を表す。ヘッド部で発生した時はヘッドユニフィケーションの後に実行される述語列を表す。

syntax_sugar/3 を二つ以上記述する場合, 第一引き数に記述する項の形式はファンクタが同じで引き数の数が同じであってはならない。また第一引き数が変数であるものは一つしか書けない。そしてその優先順位はそれらの一番後になる。

クローズ・レベル・シンタックス・シュガーはヘッドのみからなるクローズがその対象である。したがって clause_syntax_sugar/2 の第一引き数に記述するクローズは置き換えるヘッドを指定する。第二引き数は置き換え後のクローズであり、ヘッドのみからなるクローズまたはヘッドとボディよりなるクローズを指定する。

clause_syntax_sugar/2 を二つ以上記述する場合, 第一引き数に記述するヘッドの形式は述語名が同じで引き数の数が同じであってはならない。また第一引き数が変数であるものは一つしか書けない。そしてその優先順位はそれらの一番後になる。

置き換え条件で記述できる述語列は条件定義で定義した述語または組込述語でなければならない。置き換え条件と条件定義では組み込みのシンタックス・シュガーのみが使える。

[ターム・レベル・シンタックス・シュガーの例]

CIL組込シンタックス・シュガーの簡易版の記述

・入力プログラム

```
define_syntax_sugar
  syntax_sugar(X!Y, E, role(Y,X,E));
  syntax_sugar(X=Y, unify_cil(X,Y), true);
end_syntax_sugar.
a(X):- X!a = 1.
```

・展開プログラム

```
a(X):- role(a,X,A), unify_cil(A,1).
```

[クローズ・レベル・シンタックス・シュガーの例]

辞書記述言語のシンタックス・シュガーによる記述

・入力プログラム

```
define_syntax_sugar
  clause_syntax_sugar((X => Y)), E):- select(X,Y,E),!;
  select({P, E}, [], (jg_dict(P, E, X):-!,name(X))):-!;
  select({P, E}, A, (jg_dict(P, E, A):-!)):-!;
end_syntax_sugar.
noun(taro)      =>  [].
verb(yakudateru)  =>
        {stem/yakudate,case/[(agent,ga),(object,wo)]}).
```

・展開プログラム

```
jg_dict(noun, taro, X):-!, name(X).
jg_dict(verb, yakudateru, A):-
     mk_assoc((stem/yakudate,case/[(agent,ga),(object,wo)]), A),!.
```

3.3.4 インクルード宣言

インクルード宣言はプログラム中に他のファイルから、クローズを取り込む事を指定する宣言である。プログラムのインクルードと宣言のインクルードがある。

```
<include declaration> ::= <program include> | <declaration include>
<program include> ::= ":-include_program(", <file identifier>, ")."
<declaration include> ::= ":-include_declaration(", <file identifier>, ")."
<file identifier> ::= <atom>| "toolbox/" <atom>
```

1. プログラムのインクルード

   プログラムのインクルードはファイル識別子を指定する。ファイル識別子はアトムで記述する。インクルードしたプログラム・ファイル中にインクルード宣言があってもよい。

   コンサルトで指定したファイル中にプログラムのインクルード宣言があればそれはコンサルトの指定と同じ意味となる。ファイル識別子に対応するコンパイルド・プログラム（ファイル識別子.cmp ）があれば、それをコンサルトし、無ければ、ソース・プログラム（ファイル識別子.cil ）をコンサルトする。

   コンパイルで指定したファイル中にプログラムのインクルードがあれば、それはそのソースプログラムの宣言箇所に指定したソース・ファイル（ファイル識別子.cil ）をマージしたプログラムとしてコンパイルされる。

   ［例］ `:-include_program(discourse).`

2. 宣言のインクルード

   宣言のインクルードもファイル識別子を指定する。インクルードするファイル中には処理系に対する宣言のクローズしか書けない。インクルードしたファイル（ファイル識別子.dcl）中の宣言クローズがそのプログラム中で有効となる。インクルードした宣言ファイル中にインクルード宣言があってはならない。

   オペレータ宣言、シンタックス・シュガー宣言の有効範囲はそのファイル内であり、宣言を含むプログラムをインクルードしてもそれをインクルードしたファイルには宣言が適用されない。同じ宣言を別々のファイルに適用したいときには宣言のインクルードをそれぞれのファイル内で行う必要がある。

   ［例］ `:-include_declaration(my_operator).`

3.4 述語

CIL で記述できる述語には、単一化述語、CIL 組込述語、KL0 組込述語、ユーザ定義述語、オペレータ述語、ESP 述語、制約述語がある。

```
<predicate>     ::= <unify predicate>    | <CIL builtin predicate>
                  | <KL0 predicate>      | <user predicate>
                  | <operator predicate> | <ESP predicate>
                  | <constraint predicate>
```

3.4.1 単一化述語

CIL の組込シンタックス・シュガーによる記法である。CIL 組込述語の unify_cil/2 または KL0 組込述語の unify/2 と同じに扱われる。

```
<unify prediacte> ::= <term>, "=", <term>
```

```
［例］  X=Y
        a=A
        PST = {a/X, b/2}
```

3.4.2 CIL 組込述語

CIL 組込述語の一般形式は以下のとおりである。

```
<CIL builtin predicate> ::= <CIL builtin predacate name>
                          | <CIL builtin predicate name>,  "(", <argument>, ")"
<CIL builtin predicate name> ::= <atom>

<argument>       ::= <terms>
<terms>          ::= "(" <terms> ")"
                   | <term>, {",", <term>} ···
```

[例] role(X,a,V)
unify_cil(X,Y)

3.4.3 KL0 組込述語

KL0 組込述語は『KL0 組込述語説明書』で説明されている。その一般形式は以下のとおりである。

```
<KL0 predicate> ::= <KL0 predicate name>
                  | <KL0 predicate name>,  "(", <argument>, ")"
<KL0 builtin predicate name> ::= <atom>
```

[例] !
equal_string(X,Y)

3.4.4 ユーザ定義述語

ユーザ定義述語はプログラム内で定義した規則と事実のクローズのヘッド部の述語である。その一般形式は以下のとおりである。

```
<user predicate > ::= <user predicate name>
                    | <user predicate name>, "(", <argument>, ")"
<user predicate name> ::= <atom>
```

[例] a(X,Y)

3.4.5 オペレータ述語

オペレータ述語は述語名がオペレータとして定義されたときに使うことが出来る。形式としてプレフィックス、インフィックス、ポストフィックスがある。

```
<operator predicate>    ::= <prefix predicate>
                          | <infix predicate>
                          | <postfix predicate>
<prefix predicate>      ::= <operator predicate name>, <term>
<infix predicate>       ::= <term>, <operator predicate name>, <term>
<postfix predicate>     ::= <term>, <operator predicate name>
```

[プレフィックスの例] prefix_p X は prefix_p(X) と同じ
[インフィックスの例] X infix_p Y は infix_p(X,Y) と同じ
[ポストフィックスの例] X postfix_p は postfix_p(X) と同じ

3.4.6 ESP 述語

ESP マクロによる記法である。それぞれ対応する KL0 組込述語と同じに扱われる。

```
:create(X,Y)    =>  method_call(X, create, Y)
X is Y          =>  unify(X,Y)
X == Y          =>  egual(X,Y)
X \== Y         =>  not_equal(X,Y)
X =:= Y         =>  identical(X,Y)
X =\= Y         =>  not_identical(X,Y)
X < Y           =>  less(X,Y)
X =< Y          =>  not_less(Y,X)
X > Y           =>  less(Y,X)
X >= Y          =>  not_less(X,Y)
```

```
<ESP predicate> ::= <method call>       | <term> , "is", <term>
                  | <term> ,"==", <term>  | <term> ,"\==", <term>
                  | <term> ,"=:=", <term> | <term> ,"=\=", <term>
                  | <term> ,"<", <term>   | <term> ,"=<", <term>
                  | <term> ,">", <term>   | <term> ,">=", <term>

<method call> ::= ":", <method name>,  "(", <argument>, ")"
```

[例]　:create(#standard_input_file, File)

3.4.7 制約述語

制約の記述できる CIL 組込述語に constr/1, constr/2,constr/3, wif/2, wif/3 がある。制約の言語仕様はこの述語内でのみ有効である。 wif の第一引き数には ESP マクロ抑制が必要である。 constr, wif の各引き数の仕様に関しては「組込述語」を参照のこと。

```
<constraint predicate> ::=
          "wif(''(", <constraint>, "),", <predicate>, ")"
        | "wif(''(", <constraint>, "),", <predicate>,",", <predicate>, ")"
        | "constr(", <constraint>, ")"
        | "constr(", <constraint>, <constraint response>, ")"
        | "constr(", <constraint>, <constraint response>, <constraint response>, ")"

<constraint response> ::= "true" | "false" | <variable>
```

[例]　constr(X<Y)
　　wif(''(X<Y), X=Z)

3.5 制約

制約には基本制約として論理制約、原始制約、計算制約、ASSIGN 制約がある。そしてそれらの制約を結合する否定、接続、順接続、IF 接続がある。そのときに制約の接続子の強度と指定したい結合にしたがって適当に括弧で区切る必要がある。

```
<constraint> ::=  <logical constraint>
                | <primitive constraint>
                | <arithmetic constraint>
                | <assign constraint>
                | "(", <constrtaint>, ")"
```

```
| "not(", <constraint>, ")"
| <sequential connective>, "(", <constraint>, ",", <constraint>,")"
| <if connective>
```

3.5.1 基本制約

1. 論理制約

制約の値として真偽値 (true/false) を扱う制約である。 constr(true) は成功し、 constr(false) は失敗する。 constr(X) の場合、 X は true か false の値をとる。 X が未定義の場合、評価は遅延実行される。

```
<logical constraint> ::= "true" | "false" | <variable>
```

[例] constr(X), X = true. は成功する。

2. 原始制約

項と項の等価性を定義する制約である。項1= 項2 は項1 と項2 が等しいことを示す。項1\== 項2 は項1 と項2 が等しくないことを示す。項が未定義変数の場合は評価は遅延実行される。制約が決定的になれば単一化が行われる。

```
<primitive constraint> ::= <term>, "=", <term>
                         | <term> , "\==", <term>
```

[例] constr(X=1). を実行すると、 X = 1 となる。

3. 計算制約

整数および計算式に対する等価性、大小関係を定義する制約である。計算関係子はそれぞれ =:= 等しい、 =\= 等しくない、< 小さい、>大きいを意味する。 計算の演算子はそれぞれ、 + 足し算、 - 引き算、 * 掛け算、 / 割り算、 mod 余りを意味する。式が未定義変数の場合は評価は遅延実行される。制約が決定的になれば単一化が行われる。

```
<arithmetic constraint> ::=
        <arithmetic expression>, <arithmetic relation>,<arithmetic expression>
<arithmetic relation> ::= "=:=" | "=\=" | "<" | ">"
<arithmetic expression> ::=
        "(", <arithmetic expression>, ")"
        | <number>
        | <variable>
        | <arithmetic expression>, <arithmetic operator>, <arithmetic expression>
<arithmetic operator> ::= "+" | "-" | "*" | "/" | "mod"
```

[例] constr(1+X=:=Y), Y = 4. は、 X = 3, Y = 4 となる。
constr(1+X=\=Y), X = 3, Y = 4. は失敗する。

4. ASSIGN 制約

assign(X,Y) は片方向の単一化を行う。 X が具体化した時点で X を Y に代入する。

```
<assign constraint> ::= "assign(", <term>, ",", <term>, ")"
```

[例] constr(assign(X,Y)), X = 1. は、 X = 1, Y = 1 となる。

3.5.2 制約の結合

1. 否定
   not(制約) は、制約の否定を指定する。

   [例] constr(not(true)). は、失敗する。

2. 接続
   1つめの制約が具体化していなくても、2つめの制約の評価を行う。，は論理積、；は論理和、－＞は包含、＜－＞は必要十分の関係を示す。

   ```
   <connective>    ::= "," | ";" | "->" | "<->"
   ```

   [例] constr((X,Y)). は、X = true, Y = true となる。
   constr((X,Y),false,Z), Y= true. は、X = false, Z = false となる。

3. 順接続
   1つめの制約が具体化していなければ、2つめの制約の評価は2つめの制約が決定するまで中断される。 and は積、orは和、implyは包含、 exorは排他論理和、norは和の否定、 nandは積の否定を示す。

   ```
   <sequential connective> ::= "exor" | "nor" | "nand" | "and" | "or" | "imply"
   ```

   [例] constr(and(X,Y)). は、X = true, Y = true となる。
   constr(and(X,Y),false,Z). は、X, Y, Z は未定義となる。
   constr(and(X,Y),false,Z), Y= true. では、X, Z は未定義となる。

4. IF 接続
   if(X,Y,Z) で X が成立すれば Y を評価し、成立しなければ Z を評価する。

   ```
   <if connective> ::=
           "if(", <constraint>, "," <constraint>, "," <constraint>, ")"
   ```

   [例] constr(if(P=1, Q=1, R=1)), P=1, Q=1, R=2. を実行すると成功する。

3.6 項

CIL で記述できる項には Prolog と同様な通常項、ESP マクロで記述できる ESP 項、CIL の組込シンタックス・シュガーで記述できる CIL 項がある。そして、マクロやシンタックス・シュガーを抑制することを示す文法抑制項がある。

```
<term> ::= <normal term> | <ESP term> | <CIL term> | <syntax suppressed term>
```

3.6.1 文法抑制項

文法抑制項にはユーザ・シンタックス・シュガー抑制項 、組込シンタックス・シュガー抑制項 ESP マクロ抑制項がある。

CIL 処理系は、＋や-，!，# などの記号を ESP マクロやシンタックス・シュガーと見なして自動的に展開する。その展開順序はユーザー定義シンタックス・シュガー、組込シンタックス・シュガー、ESP マクロである。文法抑制項はその展開を抑制するためのものである。

```
<syntax suppressed term>  ::=
                  <user syntax sugar suppressed term>
                | <builtin syntax sugar suppressed term>
                | <ESP macro suppressed term>
```

```
<user syntax sugar suppressed term>      ::= "$$(", <term>, ")"
<builtin syntax sugar suppressed term>   ::= "$(", <term>, ")"
<ESP macro suppressed term>       ::= "‘(", <term>, ")" | "‘‘(", <term>, ")"
```

注) ‘ はバッククオート

1. CIL組込シンタックス・シュガーと衝突しているESPマクロをESPマクロとして用いる場合, $を用いる。現在, !, :(infixのみ), #(infixのみ) が衝突している。

!	スロットアクセス	スロットアクセス
:	条件付き項	クラス指定メソッドコール
#	同格項	文字コード, 文字列

[例] `W = $(Obj!window), :putc(W, $(key#tab))`

2. 述語 wif で 制約の記述に

`=, ==, >, <, +, -, *, /`

のESPマクロ対象記号を用いる場合, ‘ または “ を用いる。

[例] `wif(‘(A=B), A=C) .`

3. マクロ記号を普通のファンクタとして用いる時。以下の例では、差分リストの区切りとして - を用いている。

[例] `sentence(A, B, ‘(S0-S1))`

4. ユーザ定義のシンタックス・シュガーをある箇所だけシンタックス・シュガーとみなしてほしくないとき、$$を用いる。

5. constr の第一引き数はCIL処理系がESPマクロ抑制を行うので抑制する必要はない。

6. 部分項中のラベルと値の区切りの／はESPマクロとは見なされない。

7. 各抑制が重なった場合の記述順序は次のようにする。

```
$$($(‘‘( <term> )))
```

3.6.2 通常項

通常項には変数、アトム、数、複合項、リストがある。

```
<normal term>   ::= <variable> | <atom> | <number> | <compound term> | <list>
<number>        ::= <integer> | "+", <integer> | "-", <integer>
<compound term> ::= <atom>, "(", <terms>, ")"
<list>          ::= "[]"
                  | "[", <terms>, "]"
                  | "[", <term>, "|", <list>, "]"
                  | "[", <term>, "|", <variable>, "]"
```

3.6.3 ESP項

ESP項には浮動少数、ストリング、ベクタ、数式項、クラス参照項がある。

```
<ESP term>        ::= <float> | <string> | <vector> | <arithmetic term>
                  |<class reference term>
<float>           ::= <point expression> | <exponent expression>
<point expression>        ::= <number>, "." , <integer>
<exponent expression>     ::=  <number>, "^", <number>
                             | <point expression> ,"^", <number>
<vector>          ::= "{", <terms>,"}"
<arithmetic term>         ::= <integer> | <float> | <variable>
                            |<arithmetic term> "+" <arithmetic term>
                            |<arithmetic term> "-" <arithmetic term>
                            |<arithmetic term> "*" <arithmetic term>
                            |<arithmetic term> "/" <arithmetic term>
                            |<arithmetic term> "mod" <arithmetic term>
<class reference term>    ::= "#", <class name>
```

3.6.4 CIL 項

CIL 項は組込シンタックス・シュガーで提供される項であり、部分項、ラベル指定値項, 条件付項、同格項、遅延実行指定変数、遅延実行型条件付項、遅延実行変数単一化項がある。

```
<CIL term>        ::= <partial specified term>
                  | <labeled value term>
                  | <conditioned term>
                  | <tagged term>
                  | <lazy variable>
                  | <lazy conditioned term>
                  | <lazy unify term>
```

1. 部分項

部分項は、ラベルと値の対の順序のない集りであり、ラベル - 値対の出現順序には意味がない。ラベルには変数と部分項を含まない制限された項が、値には部分項も含めたあらゆるタイプの項が記述できる。同じラベルを持つ要素は同一レベルに存在してはいけない。つまり、 X = { a/1, b/2,a/2 }. のような部分項は記述できない。

ベクターと部分項は同じ様な形をしているが、ベクターのうちでその引き数の形が全て X/Y のものが部分項である。 以下に例を挙げる。

部分項 {a/1, 2/3, f(X,Y)/Z}
{10/2}
ベクタ {1, 5/3}
{a/2, X,c/3} …アトムは割り算出来ないためエクセプションとなる。

```
<partial specified term> ::= "{", <PST element list>, "}" | "{}"
<PST element list> ::= < PST element>, { <PST element>} …
<PST element>     ::= <label>, "/", <value>
<label>           ::= <restricted term>
<value>           ::= <term>

<restricted term>         ::= <atom> | <number>
                            | <restricted compound term>
<restricted compound term>  ::= <atom>, "(", <restricted terms>, ")"
<restricted terms>        ::=  "(" <restricted terms> ")"
```

```
                        | <restricted term>, {",", <restricted term>} …
```

［例］ { a/1, f(1)/X, c/{d/g(Y)}, e/{d/h(Z)} }
{ 関係 / 愛する,
主格 / { 名前/ 健, 年齢/ 25 },
対格 / { 名前/ 奈緒美, 年齢/ 24 } }

2. ラベル指定値項
ラベル指定値項は部分項 P のラベル L を持つ要素の値を示す。

```
<labeled value term>    ::=  <variable>, "!", <label>
                           | <partial specified term>, "!", <label>
```

［例］ {a/1, b/X}!b
X!c

3. 条件付項
条件付項はある条件を満たす項を示す。条件には述語を指定する。

```
<conditioned term>  ::= <term>, ":", <predicate>
```

［例］ X:father_of(X, 太郎)

4. 同格項
同格項は同格である二つの項を一つの項として記述する項である。X#Y は、X:(X=Y) の略記である。

```
<tagged term> ::= <term>, "#", <term>
```

［例］ X#{a/1, b/X!a}

5. 遅延実行指定変数
遅延実行指定変数は遅延実行制御を行う述語中の遅延実行制御変数を示す。

```
<lazy variable> ::= <variable>, "?"
```

［例］ print(X?), X = ok. は、ok が表示される。
print(X?). だけでは、X は未定義のままで何も表示されない。

6. 遅延実行型条件付項
遅延実行型条件付項は遅延実行制御が必要なある述語を条件として満たす遅延実行制御変数を示す。@p は、X:p(X?) の略記であり、X@p は X:p(X?) の略記である。条件の述語の引き数の数は 1 でなければならない。

```
<lazy conditioned term> ::=
          "@", <one arity predicate name>
        | <variable>, "@", < one arity predicate name>
        | <partial specified term>, "@", < one arity predicate name>
< one arity predicate name> ::= <atom>
```

［例］ X = {a/b}, Y = {a / @print}, X = Y. は、b が表示され、X=Y={a/b} となる。
man(X@animal@clever). は man(X):-animal(X?), clever(X?). と同じである。

7. 遅延実行変数単一化項

遅延実行変数単一化項は複合項の中身だけを遅延実行制御変数として、外側は単一化 したいときに用いる。
T:(T?=X) の略記である。

```
<lazy unify term>   ::= <variable>, "??"
```

[例]　X = a(Y??) は bind_hook(T, (T=Y)), X=a(T) と同じである。
これを実行すると Y の状態にかかわらず X は具体化する。
しかし、X = a(Y?) は bind_hook(Y, X=a(Y)) と同じのため、
Y が未定義の場合、X は具体化しない。

3.7 組込述語

3.7.1 組込述語の概要

1. 単一化、等価性、複写

単一化述語 (unify_cil/2,=/2) には、部分項に対する単一化の機能が組込まれている。単一化のように、部分項も含めた項を対象としている述語は、部分項の値が部分項であれば再帰的に適用される。データの同一性を調べる述語 (same/2) でも同様である。

遅延実行制御を用いると等価性に関して、情報の受け渡しの方向を限定したり (assign/3) 非等価性を制約として宣言することができる (dif/2)。ただし、拡張性という性質をもつ部分項は、遅延実行される等価性の判定はできないため、この2つの述語に関しては、部分項を適用外としている。

部分項を含めたデータの複写を行う述語を設ける (fullCopy/2)。ただし、実装の都合上、変数にかかっているデーモンは複写できない。

2. 部分項

名前とその値の対の集合から成る部分項の意義の1つは「属性が持つ名前でその値を操作できる (role/3,locate/3)」ことである。これは逆に言えば「名前が分からなければデータを操作できない」ことであり、プログラミングの際には、部分項の中身を別の方法で知る述語が必要であることを示す。現在は、部分項が持つラベルを全て集めてきたり (setOfkeys/2)、部分項内のデータをレコード形式に変換したり (record/2)、部分項の対をバックトラックベースで全て調べる (getRole/3) ことができる述語を揃えている。

3. 照合操作

部分項は意味ネットワークなどの一種のグラフ表現と考えることも出来る。その場合必要となる照合操作述語のサブセットを準備する。

まず、部分項を別の部分項に併合する述語がある (merge/2,d_merge/2,t_merge/2)。併合された方の部分項が変化するだけで、他方の部分項は変化しないという点で単一化と異なる。

部分項の拡張は、単一化の仕様の一部であるが、部分項の削除に関しては、ラベルを指定して削除する述語 (delete/3) と、部分項のサブパターンを指定して削除したりする述語 (masked_merge/3) を設ける。

この他にも、2つの部分項に共通部分の単一化を行ったり (glue/2)、共通部分を調べたりする (meet/3) 述語がある。

また2つの部分項が包含関係にあるかどうかを調べる述語がある (subpat/3, t_subpat/2)。さらに強制的に包含関係になるように、片方の部分項を拡張する述語 (extend/3) がある。

4. 遅延実行制御

「変数が具体化された時に、デーモンを実行する (freeze/2)」遅延実行制御述語の応用として、デーモンの実行のタイミングをもう少し細かく指定できるような幾つかの述語を用意する。例えば、その構造体の中の変数がすべて具体化された時に実行するように制御できたり (when/2)、制約言語などで用いる論理値 (true, false) に応じてデーモンの実行を制御できる (wif/2, wif/3) 述語である。また「複数のデーモンのうちどれか1つを実行せよ」という排他制御 (pv/2) や、「複数の変数のうちどれかが具体化されればデーモンを実行せよ」という制御 (freeze/3) も可能となる。

5. 制約言語と遅延実行型述語

どのような制約が記述できるかはすでに述べた。制約言語や幾つかの遅延実行型述語には、動作モードを、述語の引数として指定できる機能がある。

制約言語のインタプリタ (constr/3) には、次のようなモードがある。すなわち、制御モードとして、

- チェックするだけ
- 正しい代入を遅延的に探せ
- 否定を正しくするような代入を遅延的に探せ

が指示でき、結果のモードとして、

- 未定
- 正しい (true)
- 正しくない (false)

を返させる事ができる。また、デフォールトのモードが入る引数の数が異なる述語 (constr/1,constr/2) を設ける。2引数の場合は、制御モードが「正しい代入を遅延的に探せ」となる。1引数の場合は、制御モードが「正しい代入を遅延的に探せ」となり、結果モードが「正しい (true)」となる。どちらも正しい答えを遅延的に探しにいく。

他の遅延実行型の述語 (equal/5,t_equal/5,add/6,multiply/6,mod/6,evaluate/5,exp/5) に関しても、これらのモードを設け、遅延実行的な演算を可能にしている。

6. その他
その他、プロセス間通信などにおいて、受け取り側からの要求に応じて要求量だけのデータを流すストリームプログラミングを可能にする述語も準備する (buffer/2)。

以下の説明では、引数変数名は、その引数に許されるデータタイプやその役割を次の原則に従って表す。

- X,Y …… 全ての CIL データタイプ
- A,B …… 部分項を除く CIL データタイプ
- P …… 部分項
- L …… ラベル
- V …… 値
- T …… 論理値 (true, false)
- C …… 制御値
- R …… 結果値
- G …… 述語
- F …… フラグ
- D …… 差分リスト形式
- I,J,K …… 数
- Constr …… 制約
- Buffer …… バッファ
- Record …… レコード

3.7.2 単一化

unify_cil(X,Y)

X と Y を単一化する。［記法］ X = Y

［例］ {a/P, b/2} = {b/Q, a/1} は、P = 1, Q = 2 となる。
X = {f/1}, Y = {g/2}, X = Y は、X = Y = {f/1, g/2} となる。

［注］ 1章で述べた CIL の単一化のアルゴリズムはこの単一化述語を用いて実現される。現在のところ, ヘッド内の引数間の単一化 (ヘッド内ユニフィケーション) では CIL の単一化は行われない。従って部分項を含むような構造体をヘッド内ユニフィケーションする場合は, それらの単一化は上の述語を用いてボディに展開しておく必要がある。例えば、p(X, X) :- !, q, r .... なる述語 p に対しては p(a/1, b/2) は正しく動作しない。 p の定義を p(X, Y) :- X = Y , !, q, r .... のように変える必要がある。

assign(A,B,T)

A から B への一方向の単一化を行う。

1. A に含まれる変数への束縛はサスペンドされる。
2. T の値は一方向の単一化が成功すれば true、失敗すれば false となり、いずれの場合も述語の実行は成功する。
3. A, B には部分項は含まれてはいけない。

［例］ assign(f(A), f(B), T). は、T = true, A = B であるが値は持たない。
assign(f(A), f(a), T). は、T, A は値を持たない。
assign(f(a), f(A), T). は、T = true, A = a となる。

3.7.3 等価性

same(X,Y)

X と Y の同一性を調べる。 同一であれば成功、そうでなければ失敗する。 変数に関して、X と Y の値は互いに単一化されない。

［例］ same({a/1, b/2}, {b/2, a/1}). は成功する。
same({a/X}, {a/1}). は失敗する。
X = Y, same(X, Y). は成功する。

dif(A,B)

A と B の値が異なるという制約宣言用述語。 A、B の中のすべての変数が具体化された時に、 その相違性を判定し等しくなければ成功、等しければ失敗する。

［例］ dif(A,B), A = 1, B = 2. は成功する。
dif(A,1), print(ok), A = 1. は 'ok' を表示後、失敗する。

equal(X,Y,C,R,_)

X と Y の等価性を調べる。 制御値 C の指定により、次のモードの動作をする。 ただし、X、Y に変数を含んでいる場合には、その変数への 束縛はサスペンドされる。

1. C が未定義の場合、X と Y が等しいかどうか調べる。 X と Y が等しければ、C、R とも true となり、述語は成功する。 X と Y が等しくなければ、C、R とも false となり、述語は成功する。
2. C = false と指定した場合, X と Y が等しくないかどうか調べる。 X と Y が等しくなければ、R は false となり、述語は成功する。 X と Y が等しければ、述語は失敗する。
3. C = true と指定した場合、X と Y を単一化する。 単一化が失敗すれば、述語は失敗する。 単一化が成功すれば、R は true となり, 述語は成功する。

[例] equal(X, Y, false, R, _), X = 1, Y = 4. は、
R = false, X = 1, Y = 4 となる。

[注] これらの述語の最後の引数はシステムが用いる制御値で、この値がセットされればその時にサスペンドされている述語は全てキャンセルされる。特にユーザが意識する必要はない。組込み述語で引数が'_'になっているのは全て同じ働きをする述語である。

t_equal(I,J,C,R,_)

equal/5 とほぼ同機能であるが、次の点で異なる。I と J は数である。また、I と J は遅延実行に関して対象でない。I が具体化されるまで中断するが、I が具体化されれば J が未定義でも I と J が単一化される。

[例] t_equal(I,J,C,R,_), I = 1. は、
I = 1, J = 1, C = true, R = true となる。

3.7.4 複写

fullCopy(X,Y)

X のデータを Y に複写する。実行時には、Y は未定義でなくてはいけない。また、X 中の変数への束縛はされず、複写後も X の値は変わらない。ただし、実装の都合上、遅延実行の制御（変数にかかったデーモン）は複写されない。

[例] fullCopy(X#{a/A}, Y), Y!a = 2. を実行しても、A は未定義のまま。
X = {a/ @print, b/X}, fullCopy(X, Y), Y!b!a = ok.
を実行しても、ok は表示されない。

3.7.5 部分項

partial(P)

P が部分項ならば成功する。

[例] partial({a/1, b/X}). は成功する。

mk_assoc(A,P)

通常項から部分項を生成する。A は (/(L,V), ...) の形式の項であり、P は生成される部分項である。部分項の表記は、この述語を用いて部分項を生成する シンタックス・シュガーである。

[例] X = ('(a/1), '(b/2)), mk_assoc(X, P). は、P = {a/1, b/2} となる。

role(L,P,V)

部分項 P のラベル L を持つ要素の値を V と単一化する。表記は P!L でよい。P 中に L というラベルを持つ要素がなければ, ラベルが L で値が V の要素 を P に追加する。

[例] X = {a/1, b/2}, role(L, X, 3), L = c. は、X = {a/1, b/2, c/3} となる。
また、 X = {a/1,b/2} , X!c =3. も上の例と同じことを示す。

locate(P,L,V)

部分項 P のラベル L を持つ要素の値を V と単一化する。P 中に L というラベルを持つ要素がなければ、述語は失敗する。

[例] locate({a/1, b/2}), b, V). は、V = 2 となる。
locate({a/1, b/2}), c, V). は、失敗する。

getRole(P,L,V)

部分項P のラベル、値をそれぞれL, V と単一化する。再実行させることでP のすべてのラベル - 値対をつぎつぎとL, V に単一化 する。

```
[例] getRole({a/1, b/2}, L, V). は、
     L = a, V = 1 ;          % ';' の入力（強制失敗、別解探索）
     L = b, V = 2   となる。
```

setOfKeys(P,Set)

部分項P 中に含まれるすべてのラベルをリストにして, Set と単 一化する。

```
[例] setOfKeys({a/1, b/2}, Set). は、Set = [a,b]. となる。
```

record(P,Record)

部分項P 中に含まれる全てのラベル - 値対をリストに変換し、Record と単一化 する。

```
[例] record({a/1, b/2}, Record). は、Record = [(a,1), (b,2)] となる。
```

buffer(P,Buffer)

リストBuffer の長さだけ部分項P をリスト（バッファ）に変換し、Buffer と単一化する。P の中の要素の数がBuffer の長さより少なければ、最後にend がはいる。Buffer が変数の間は実行は中断される。

```
[例] buffer({a/1, b/3}, Buffer), Buffer = []. は、Buffer = []   となる。
     buffer({a/1, b/3}, [X,Y,Z,W]). は、
     X = (a,1), Y = (b,3), Z = end となり、W は未定義のままである。
```

3.7.6 部分項照合操作

glue(P1,P2)

P1、P2 に共通のラベルをもつ要素同士の単一化を行う。P1、P2 とも要素の拡 張はされずに同じラベルを持つ要素の単一化が行われるだけである。

```
[例] P1 = {a/1, b/2}, P2 = {a/X, c/3}, glue(P1, P2) は、
     X = 1, P1 = {a/1, b/2}, P2 = {a/1, c/3} となる。
```

merge(P1,P2)

P1 をP2 に併合し、結果をP2 に単一化する。P1 はglue と、P2 は単一化と同様の機能を持つ。すなわち併合の結果、P2 の要素が拡張されるのに対し、P1 は拡張されずにP2 と共通のラベルをもつ要素の単一化が行われるだけである。

```
[例] P1 = {a/1, b/X}, P2 = {b/2, c/3}, merge(P1, P2). は、
     X = 2, P1 = {a/1, b/2}, P2 = {a/1, b/2, c/3} となる。
```

t_merge(P1,P2)

merge と同様の機能を持つが, 次の点で異なる。

1. P1 は全く変化しない。すなわちP1 はP2 と共通のラベルをもつ要素の単一化 も行われない。
2. 要素の値が部分項であれば再帰的にt_merge を適用する。

```
[例] t_merge({b/1, a/{c/2, b/3}}, {a/{b/Y}}). は、Y = 3 となる。
```

d_merge(P1,P2)

merge と同様の機能を持つが, 次の点で異なる。 単一化できない値が存在しても、述語は失敗せずにその対がそのまま残る。

```
[例] d_merge(P1#{a/1, b/2}, P2#{a/X, b/3}). は、
     X = 1, P1 = {a/1, b/2}, P2 = {a/1, b/3}. となる。
```

delete(L,P1,P2)

ラベル L を持つ要素を P1 から削除し（なければそのまま）、P2 に併合する。 P1, P2 に対する操作は merge と同様である。

```
[例] delete(a, {a/1, b/2}, P). は、 P = {b/2} となる。
```

masked_merge(P1,P2,P3)

P1 中の要素から P2 中の要素を削除し（なければそのまま）、P3 に併合する。

```
[例] masked_merge({a/1, b/1, c/1}, {a/_, b/_}, U#{a/2}). は、
     U = {a/2, c/1} となる。
```

subpat(P1,P2,D)

P1 が P2 のサブパターンである事を調べる。調べた結果を D に帰す。 P1 の要素の集合が P2 の要素の集合の部分集合である。（値の等価性は問わ ない） P1 のラベルとそれぞれ対応する P1, P2 の値を持つ項のリストが D に差分リストの形式で返る。

```
[例] P1 = {b/1, c/X}, P2 = {a/2, b/Y, c/2, d/2}, subpat(P1, P2, D).
   を実行すると D = [(b,1,Y), (c,X,2)|Z]-Z となる。
```

t_subpat(P1,P2)

P1 が P2 のサブパターンである事を調べる。 要素の値が部分項であれば再帰的に t_subpat を適用する。

```
[例] t_subpat({a/{b/Y}}, {b/1, a/{c/2, b/3}}). は成功する。
     t_subpat({a/{b/Y, c/U}}, {b/1, a/{a/2, b/3}}). は失敗する。
```

extend(P1,P2,D)

P1 が P2 のサブパターンになるように P2 を拡張する。 D は subpat と同様。

```
[例] P1 = {a/1}, P2 = {b/2}, extend(P1, P2, D). は、
     P1 = {a/1}, P2 = {a/X, b/2}, D = [(a,1,X)|Y]-Y となる。
```

meet(P1,P2,D)

P1 と P2 の共通パターンを作る。すなわち, P1 と P2 の共通のラベルについて, 差分リスト D を作る。

```
[例] meet({a/1, b/2}, {b/3, c/4}, '(L-□)). は、 L = [(b,2,3)] となる。
```

frontier(A,B,D)

A と B が単一化可能であるとき, そのための最低条件となる等式の集合を差分リ ストの形式で D に提示する。単一化不可能ならば、述語は失敗する。 但し、 A、 B は部分項であってはいけない。

```
[例] frontier(f(a,g(b)), f(A,B), '(L-□)). は、 L = [(a=A), (g(b)=B)] となる。
     frontier(a, b, '(L-□)). は失敗する。
```

match(A,B,D)

A と B が単一化できるための最低条件である等式の集合を差分リストの形式で D に提示する。(frontier と異なり A と B は単一化不可能でもよい) 但し、A、B は部分項であってはいけない。

[例] match(a, b, '(L-[])). は、L = [(a=b)] となる。

3.7.7 ゴール制御

solve(G)

G を実行するいわゆるメタコールである。G として記述できるのは, CIL 組込み述語、KL0 組込み述語、CIL ユーザ定義述語 (CIL 変換系でコンパイルされた述語) だけである。

[例] solve(print(ok)) は、ok が表示される。

if(T,G)

T が true または未定義ならば、G を実行する。それ以外ならば何もせず成功する。G の失敗により、述語自身も失敗する。

if(T,G1,G2)

T が true または未定義ならば、G1 を実行し、それ以外は G2 を実行する。G1 (または G2) の失敗により、述語自身も失敗する。

ifBound(X,G)

X が具体化されていれば、G を実行する。未定義ならば、何もせず成功する。G の失敗により、述語自身も失敗する。

ifUnbound(X,G)

X が未定義ならば、G を実行する。具体化されていれば、何もせず成功する。G の失敗により、述語自身も失敗する。

3.7.8 遅延実行制御

freeze(X,G)

X が未定義の場合、具体化されるまで G の実行は中断する。X が具体化されていれば、即座に G を実行する。G の失敗により、述語自身も失敗する。

freeze(X,Y,G)

X と Y がどちらもが未定義の場合、どちらかが具体化されるまで、G の実行は中断する。X と Y どちらか又は両方ともが具体化されていれば、即座に G を実行する。G の失敗により、述語自身も失敗する。

wif(T,G)

T が未定義の場合、具体化されるまで、T の評価を中断し、T が true に具体化されれば、G を実行する。それ以外の値を持てば何もせず成功する。G の失敗により、述語自身も失敗する。T には制約を記述することが出来る。

wif(T,G1,G2)

/indexwif/3 T が未定義の場合、具体化されるまで、T の評価を中断し、T が true に具体化されれば、G1 を実行する。それ以外の値を持てば G2 を実行する。G1 (または G2) の失敗により、述語自身も失敗する。T には制約を記述することが出来る。

when(A,G)

A の構造体中の変数のすべてが具体化するまで、G を中断する。ただし、A は部分項であってはならない。

```
[例] freeze(X, print(ok)), print(1), X = f(M), print(2), M = 1. は、
 '1ok2' と表示される。
 when(X, print(ok)), print(1), X = f(M), print(2), M = 1. は、
 '12ok' と表示される。
```

pv(F,SuspG)

F が未定義ならば, F を 1 にして SuspG を実行する。F が定義済ならば, SuspG は実行しない。pv は複数個のデーモンに対して、どれかつだけを実行せよという指定をする。

[例] 後で述べる遅延実行型論理演算述語 not(X,Y) は、pv/2 を用いて定義される。

```
not(X,Y):-
        freeze(X,pv(F,if(X,Y=false,Y=true))),
        freeze(Y,pv(F,if(Y,X=false,X=true))).
```

3.7.9 制約

constr(Constr,C,R)

Constr には言語仕様の制約で示した文法を用いて、変数に関する制約を記述できる。C は制約の制御値、R は制約の結果であり、未定義または論理値 (true か false) である。C が true(または false) の時、制約 Constr に表われる変数は、制約 Constr の結果が true(または false) になるように、遅延実行的に解が求められる。すなわち制約中の変数の解が唯 1 つの場合は、具体化され、結果に値が返るが（例 1、2）、解に曖昧性が残る場合には、まだ具体化されず（例 3）、他の制約などで解が唯 1 つになった段階で、具体化される（例 4）。もし、解が存在しなければ、述語が失敗する（例 5）。C が未定義ならば、制約 Constr の中の変数および C に関して遅延実行される（例 6）。

```
[例 1] constr(X=1, true, R). は、X = 1,R = true となる。
[例 2] constr(X\==2, false, R). は、X = 2, R = false となる。
[例 3] constr(X=1, false, R). は、X, R は未定義のままである。
[例 4] constr(X=1, false, R), X = 2. は、X = 2, R = false となる。
[例 5] constr(X=1, false, R), X = 1. は、失敗する。
[例 6] constr(X=1, C, R). は、X,C,R は未定義のままである。
```

引数の異なる述語も用意している。その内容は以下のとおりである。

```
1引数    constr(X) :- T? = true, constr(X, true, T).
2引数    constr(X, Y) :- constr(X, true, Y).
```

3.7.10 遅延実行型論理演算

ここで説明する論理演算に表われる変数は、論理値（true または false）である。その値は遅延実行的に決定される。。すなわち、論理関係を満たす変数が 1 組以上ある時は、その可能性が 1 つになるまで変数への値の束縛はサスペンドされる。明らかに 1 つしかない時は即座に束縛される。変数が関係を満たしていなかったり、関係を満たす変数が 1 つもなかったりした場合は失敗する。

not(T1,T2)

T1 は T2 の否定である。

and_cil(T1,T2,T3)

T3 は T1 と T2 の積である。

or_cil(T1,T2,T3)

T3 は T1 と T2 の和である。

exor(T1,T2,T3)

T3 は T1 と T2 の排他的論理和である。

nand(T1,T2,T3)

T3 は T1 と T2 の積の否定である。

nor(T1,T2,T3)

T3 は T1 と T2 の和の否定である。

imply(T1,T2,T3)

T3 は T1 と T2 の包含である。

iff(T1,T2,T3)

T3 は T1 と T2 の必要十分である。

```
[例] and_cil(X, Y, true). は、X = Y = true となる。
     or_cil(X, Y, true). は、X、Y の値はまだ決まらない。
     or_cil(X, false, true). は、X = true となる。
```

3.7.11 遅延実行型算術演算

ここでの説明では、I,J,K は数であり、C,R は true か false である。I,J は計算の入力であり、K は計算結果である。C は計算結果が正しいかどうかの制御値であり、R はその結果である。いずれの値も遅延的に決定される。すなわち、C に true が具体化すると I,J,K が四則関係を満たすように計算を行い、関係を満たす変数の組が 2 つ以上あれば値の束縛はサスペンドされ、1 つになった時に値が決まる。C が false の時は、I,J,K が四則関係を満たさないように計算を行う。

add(I,J,K,C,R,_)

K は I と J の和である。

multiply(I,J,K,C,R,_)

K は I と J の積である。

mod(I,J,K,C,R,_)

K は I と J の法である。

exp(Exp,Val,C,R,_)

Exp は、数と演算子 (+,-,*,/,mod) を組み合わせて作った数式で、Val はその計算結果で ある。全ての引数は遅延的に評価される。

evaluate(Exp,Val,C,R,_)

Exp は I+J, I-J, I*J, I/J, I mod J の形のいずれかをもち、Val はその計算結 果である。ただし、Val に関しては遅延実行の対象となるが、I と J に関しては 遅延実行制御されないので、I と J は変数であってはいけない。

3.7.12 その他

CIL 処理系では、標準的な Prolog 仕様にほぼ準拠した述語や入出力述語をいくつか提供している。また、CIL 項を含む一般項のプリティプリント用の入出力述語も用意している。

statistics(runtime,X)

実行時間を計る。

var(X)

変数 X が未定義か調べる。

nonvar(X)

変数 X が具体化しているか調べる。

\+(P)

述語 P の否定。

name(X,Y)

アトムまたは整数とネームの変換を行う。

genSym(X,Y)

アトム X をプレフィックスとするシンボルを Y につくる。副作用をもち、2回目以降の別のシンボルを作る。

see(X)

seeing(X)

seen(X)

read(X)

get(X)

tell(X)

telling(X)

told(X)

print(X)

write(X)

writeq(X)

nl

tab(X)

pp(X)

X をストリームにプリティ・プリントする。

3.8 オペレータ順位表

CIL のオペレータは ESP のオペレータに追加、または、一部修正したものである。以下に一覧を示す。オペレータの前の * は追加、または、一部修正したものを表し、括弧内は ESP での標準値を示す。

オペレータ	型	順位	ESP/CIL	備考
after	fx	1140	ESP	前デーモン
attribute	fx	1120	ESP	属性スロット
before	fx	1140	ESP	後デーモン
by	fx,yfx	1098	ESP	ユーザ定義マクロ
class	fx	1180	ESP	クラス定義
component	fx	1120	ESP	要素スロット
cspy	fx	900	-	-
* define_syntax_sugar	fx	1154	CIL	シンタックス・シュガー
div	yfx	400	ESP	整数除算
has	xfx	1160	ESP	クラス定義
instance	fx,xfx	1155	ESP	クラス定義
is	xfx	700	ESP	unify/2
local	fx,xfx	1154	ESP	クラス定義
mod	xfx	400	ESP	剰余
mode	fx	1080	-	-
nature	fx	1120	ESP	クラス定義
nospy	fy	900	-	-
public	fx	1080	CIL	コンパイラ
spy	fy	900	-	-
when	yfx	1098	ESP	マクロ定義
where	yfx	1098	ESP	マクロ定義
with	fx,yfx	1098	ESP	マクロ定義
with_macro	xfx	1140	ESP	マクロ定義
xor	yfx	500	ESP	ビット排他的論理和
;	xfx,xf	1150	CIL	クローズ内
,	xfy	1000	CIL	ゴールの区切り
:-	fx,xfx	1100	CIL	クローズ定義
?-	fx	1100	-	-
|	xfx	1150	CIL	リストの区切り
=	xfx	700	CIL	unify_cil/2, 原始制約
==	xfx	700	ESP	equal/2
\==	xfx	700	ESP	not_equal/2
			CIL	原始制約
=:=	xfx	700	ESP	identical/2
			CIL	計算制約
=\=	xfx	700	ESP	not_identical/2
			CIL	計算制約
=..	xfx	700	-	-
<	xfx	700	ESP	less_than/2
			CIL	計算制約
>	xfx	700	ESP	less_than/2
			CIL	計算制約
=<	xfx	700	ESP	not_less_than/2
>=	xfx	700	ESP	not_less_than/2
@<	xfx	700	-	-
@>	xfx	700	-	-

@=<	xfx	700	-	-
@>=	xfx	700	-	-
+	yfx	500	ESP	加算
	fx		ESP	正号
-	yfx	500	ESP	減算
	fx		ESP	負号
*	yfx	400	ESP	乗算
/	yfx	400	ESP	除算
* ->	yfx (xfy)	1040	CIL	包含制約
* <->	yfx	1040	CIL	必要十分制約
* ::	xfx	350	CIL	インヘリタンス機能
* :	fy,yfx	300	CIL	条件付項
	(fy,xfx)	(100)	ESP	メソッドコール
* @	fx,yfx	200	CIL	遅延実行型条件付項
!	yfx	200	CIL	ラベル指定値項
			ESP	スロットアクセス
* ?	xf	180	CIL	遅延実行指定変数
* ??	xf	180	CIL	遅延実行変数単一化項
#	xfx	90	CIL	同格項
	fx		ESP	クラスオブジェクト
##	xfx	50	ESP	パッケージ名指定
:=	xfx	700	ESP	スロット代入
* =>	yfx (xfx)	1099	ESP	マクロ定義
==>	xfx,xf	1099	ESP	マクロ定義
-->	xfx	1100	CIL	DCG 記法
\+	fy	900	CIL	not/1
/\	yfx	500	ESP	ビット論理積
\/	yfx	500	ESP	ビット論理和
<<	yfx	400	ESP	左シフト
>>	yfx	400	ESP	右シフト
^	xfy	200	ESP	浮動小数点
'	fx	999	ESP	マクロ抑制

3.9 プログラム例

3.9.1 discourse

このプログラムは状況を用いた談話解釈モデルを用いて同じ文が談話状況 (discourse situations) によっていろいろな解釈 (interpretation) を持つ事を示す。

```
example(Interpretation). と入力すると、実行結果が表示される。
Interpretation =>
    [soa(love,(jack,betty,loc(1)),yes),    % 文1   "I love you."
     soa(love,(betty,jack,loc(2)),yes)]    % 文2   "I love you."
```

プログラム中でのアトムおよび述語の意味は以下のとうりである。

```
sit: 状況 (situation)
soa: 事態 (state of affair)
exp: 表現 (expression)
dl : 談話時空間 (discourse location)
ip : 解釈
ag : 行為者
obj: 対象者
sp :  話者
hr :  聴者
discourse_situation: 談話状況 (DISCOURSE SITUATION) の定義
discourse_constraint: 談話制約 (DISCOURSE CONSTRAINT) の定義
member : メンバの定義
```

ソース・プログラム

```
example(Interpretation):-
       discourse_constraint(
            [{sit/ [soa(speaking,  (jack, _), yes),
                    soa(addressing,(betty, _),yes)|_],
                exp/ [i,love,you],
                dl/  loc(1)}@discourse_situation,
               {exp/ [i,love,you]}@discourse_situation],
       Interpretation).

discourse_situation({sit/S, sp/I, hr/ You, dl/ Here, exp/ Exp}):-
       member(soa(speaking, (I, Here),yes),S),
       member(soa(addressing,(You, Here),yes),S),
       member(soa(utter,(Exp, Here),yes), S).

member(X, [X|Y]).
member(X,[Y|Z]):-member(X,Z).

discourse_constraint([],[]):-!.
discourse_constraint([X],[Y]):-!,meaning(X,Y).  % Y は X の解釈である
discourse_constraint([X,Y|Z], [Mx,My|R]):-
       meaning(X,Mx),          % Mx は X の解釈である
       change_role(X, Y),      % 話者 <> 聴者
       time_precedent(X, Y), % X は Y より時間的に先行している
```

```
        discourse_constraint([Y|Z],[My|R]).

change_role({hr/X,sp/Y},{hr/Y,sp/X}@discourse_situation).

time_precedent({dl/loc(X)},{dl/loc(Y)}):-
        constr((X+1=:=Y)).    %  遅延評価のための組込述語

% 文法 - DCG(DEFINITE CLAUSE GRAMMAR) -
meaning(X#{exp/E},Y):-sentence({ip/Y,ds/X}, E, []).

sentence({ip/SOA,ds/DS})-->
        noun({ip/Ag,ds/ DS}),
        verb({ip/SOA, ds/DS, ag/Ag, obj/ Obj}),
        noun({ip/Obj, ds/ DS}).

noun({ip/jack})  --> [jack].
noun({ip/betty}) --> [betty].
noun({ip/X, ds/{sp/X}}) --> [i]
noun({ip/X, ds/{hr/X}}) --> [you].
verb({ip/ soa(love,(X, Y, Loc), yes),
          ds/ {dl/Loc}, ag/ X, obj/Y}) --> [love].
```

3.9.2 send

このプログラムは覆面算の計算を行う。

```
      SEND        答        9567
  +)  MORE            +)    1085
  ------------        -------------
      MONEY                 10652
```

を満たすような、S, E, N, D, M, O, R, Yにあてはまる数を求める。
その答えは、9, 5, 6, 7, 1, 0, 8, 2である。

```
send(M,T):- statistics(runtime,_),send(M),statistics(runtime,T).
send([[S,E,N,D],[M,O,R,E],[M,O,N,E,Y]]):-
        different([S,E,N,D,M,O,R,Y]) ,
        M? \== 0 , S? \== 0 ,
        sum(R1,0,0,M,0),
        sum(R2,S,M,O,R1),
        sum(R3,E,O,N,R2),
        sum(R4,N,R,E,R3),
        sum(O ,D,E,Y,R4).

remainder(1).    remainder(0).

digit(0). digit(1). digit(2). digit(3). digit(4).
digit(5). digit(6). digit(7). digit(8). digit(9).

different(□).
different([X|Y]):-out_of(X,Y),different(Y).
```

```
out_of(X,□).
out_of(X,[A|L]):-X? \== A?, out_of(X, L).

sum(R,X,Y,Z,R1):-
        remainder(R),
        digit(X) ,
        digit(Y) ,
    divide_with_remainder(R + X + Y, 10, R1, Z).
```

3.9.3 tomorrow

このプログラムは遅延実行のメカニズムを使って、述語の双方向性を持たせたものである。

```
tomorrow({year/ Y1, month/ M1, week/ W1, day/ D1},
         {year/ Y2, month/ M2, week/ W2, day/ D2}):-
        orderInWeek(W1,W2),
        endOfMonth(Y1,M1,L1),
        bump(D1,L1,D2,Cm),
        monthOfTomorrow(Cm,M1,M2,Cy),
        constr(Y1+Cy=:=Y2).

leapYear(X,V):-constr((X-1984) mod 4 =:= 0,_,V).

endOfMonth(Y,M,L):-wif(''(M=\=february), defaultEnd(M,L), endOfMonth1(Y,L)).

endOfMonth1(Y,L):-leapYear(Y,V), wif(V, ''(L=29), ''(L=28)).


bump(D1,L,D2,C):-
        pv(X, bump1(D1,L,D2?,C)),
        pv(X, bump1(D1?,L?,D2,C)),
        pv(X, bump1(D1?,L,D2,C?)),
        pv(X, bump1(D1,L?,D2,C?)).

bump1(X,X,1,1):-!.
bump1(X,Y,Z,0):-!, constr((X @< Y, X+1=:=Z)).


monthOfTomorrow(C,M1,M2,Cy):-
        pv(X, monthOfTomorrow1(C,M1?,M2?,Cy)),
        pv(X, monthOfTomorrow1(C?,M1?,M2,Cy)),
        pv(X, monthOfTomorrow1(C?,M1,M2?,Cy)),
        pv(X, monthOfTomorrow1(C?,M1,M2,Cy?)).


monthOfTomorrow1(0,X,X,0):-!.
monthOfTomorrow1(1,december,january,1):-!.
monthOfTomorrow1(1,X,Y,0):-!,orderMonth(X,Y).

defaultEnd(january,  31).
defaultEnd(february, 28).
```

```
defaultEnd(march,     31).
defaultEnd(april,     30).
defaultEnd(may,       31).
defaultEnd(june,      30).
defaultEnd(july,      31).
defaultEnd(august,    31).
defaultEnd(september,30).
defaultEnd(october,   31).
defaultEnd(november,  30).
defaultEnd(december,  31).


orderMonth(january,     february).
orderMonth(february,    march).
orderMonth(march,       april).
orderMonth(april,       may).
orderMonth(may,         june),
orderMonth(june,        july).
orderMonth(july,        august).
orderMonth(august,      september).
orderMonth(september,   october).
orderMonth(october,     november).
orderMonth(november,    december).
orderMonth(december,    january).


orderInWeek(X, Y):-freeze(X,Y, orderInWeek1(X,Y)).

orderInWeek1(sunday,    monday    ).
orderInWeek1(monday,    tuesday   ).
orderInWeek1(tuesday,   wednesday).
orderInWeek1(wednesday,thursday ).
orderInWeek1(thursday,  friday    ).
orderInWeek1(friday,    saturday  ).
orderInWeek1(saturday,  sunday    ).
```

第 4 章

プログラム仕様

ESP のプログラムから CIL 処理系を利用することができる。LTB の各プロセッサの CIL 処理部を構築するために、利用されているクラス群であり、LTB カーネルと呼ぶ。LTB カーネルのクラス仕様を以下に述べる。

4.1 LTB カーネル概要

1. cil_interpreter
   デバッグ環境付のインタプリタ

2. cil_syntax_sugar
   CIL のシンタックス・シュガーと ESP マクロを展開するイクスパンダ

3. cil_inspector
   実行中の CIL のゴールを調べるユーザ・インターフェイスであるインスペクタ

4. cil_term
   部分項を含む項を表示するための変換メソッドや、項中の部分項の内部形式 x(_,t(_)) を外部形式 {a/X, ···} に変換するメソッドを含むクラスである。このクラスが扱う部分項は無限構造でも可能である。

5. cil_operator
   CIL の標準オペレーターを設定するクラス

6. cil_pretty_printer
   CIL のプリティ・プリントを提供する部品クラス

7. cil_compiler
   CIL のソース・プログラムをコンパイルするクラス

8. cil_builtin
   CIL 組込述語を提供するクラス

4.2 cil_interpreter のメソッド

1. :create(#ltb_v10##cil_interpreter, ^Interpreter, ^Info)
   インスタンスを生成する。Info はデバッグ支援で必要なデバッグ情報の初期値である。デバッグ環境付で実行する場合はこの情報を引き回さなければならない。

2. :consult_on(Interpreter, File_name)
   :consult_off(Interpreter, File_name)
   プログラムのコンサルト・オン / オフする。File_name はストリングで指定する。コンパイルド・プログラムをコンサルトすることも可能である。

3. :debug_on(Interpreter, File_name)
   :debug_off(Interpreter, File_name)

プログラムのデバッグ・オン / オフする。 File_name はストリングで指定する。デバッグ用のウインドウはこの時生成 / 消去される。

4. :interpret(Interpreter, Goals, Info)
   ゴールを実行する。 Goals はシンタックス・シュガーを展開済みのゴール列とする。 Info は1の物である。

5. :execute(Interpreter, Goals, Info)
   ゴールを実行する。 Goals はシンタックス・シュガーを含むゴール列である。 Info は1の物である。実行時にシンタックス・シュガーを展開する。

6. :set_window(Interpreter, Window)
   CIL インタプリタが出力するエラー・メッセージのウインドウを設定する。ウインドウは、標準入出力でなければならない。このメソッドでウインドウを設定していない場合、エラー・メッセージはカーネルのデフォルト・ウインドウに表示される。

7. :get_compiled_instance(Interpreter, File_name, ^Instance)
   ESP プログラムから高速に CIL のコンパイルド・プログラムを呼び出すためのインターフェイスである。 File_name にはコンサルトしたコンパイルド・プログラムのファイル名をストリングで指定する。 Instance は CIL プログラムに対応するインスタンスである。このインスタンスに対して :interpret(Instance, Goal, Info) を発行することにより、ゴールを実行する。 Goal はシンタックス・シュガーを展開済みの一つのゴールである。 Info は1の物である。

8. :trace_on(Interpreter)
   :trace_off(Interpreter)
   デバッグ中の状態をそのままにして、トレースの抑制・再開を設定する。

9. :clear_program(Interpreter)
   インタープリタがコンサルトしているすべてのプログラムをコンサルト・オフ状態にする。

10. :clear_debug(Interpreter)
    デバッグ・オン状態のすべてのプログラムをデバッグ・オフ状態にする。

11. :other(Interpreter, Goal)
    ユーザ・カスタマイズ用のダミー・メソッドである。デバッグ・コマンドの other はユーザ定義用として解放されている。デバッグ・トレース中にリーシュ・コマンドとして other を選択するとユーザ定義のメソッド :other(Interpreter, Goal) を実行することができる。ユーザ定義のメソッドは cil_interpreter を継承したクラスに記述する。例えば以下のように行う。

```
class my_interpreter has
    nature
      ltb_v10##cil_interpreter;
    after:create(Class, Interpreter, Info):- init(Interpreter);
instance
    component
      other の副作用;
    :other(Interpreter, Goal):- other(Interpreter, Goal);
local
    init(Interpreter):- other の初期化定義
    other(Interpreter, Goal):-    other の定義
end.
```

4.3 cil_syntax_sugar のメソッド

1. :create(#ltb_v10##cil_syntax_sugar, ^Syntax_sugar)
   インスタンスを生成する。

2. :expand_syntax_sugar(Syntax_sugar, Key, Source, ^Expanded)
   クローズまたはゴール列のシンタックス・シュガーを展開する。ユーザ・シンタックス・シュガー, CIL 組込シンタックス・シュガー, ESP マクロが展開される。

   Key : 展開する Source の種類を指定する。
   - clause: クローズの展開をする。結果はリストで返される。
   - goals: ゴール列の展開をする。
   - head: ヘッドの展開をする。
   - body: goals と同じ。
   - clause_but_esp: clause と同じだが ESP マクロの展開は行わない。
   - goals_but_esp: goals と同じだが ESP マクロの展開は行わない。

   Source : 展開するクローズまたはゴール列.
   Expanded : 展開されたクローズまたはゴール列.

4.4 cil_inspector のメソッド

1. :create(#ltb_v10##cil_inspector, ^Inspector)
   インスタンスを生成する。

2. :inspect_goal(Inspector, Goal, Info)
   ゴールをインスペクトする。ウインドウがアクティベートされる。最初のときはポジショニングを行う。

4.5 cil_term のメソッド

1. :transform(#ltb_v10##cil_term, Term, ^New_term)
   タームを変換する。Term は変換前であり、New_term は変換後である。New_term をウインドウに:putt すれば良い。

4.6 cil_operator のメソッド

1. :declare_operator(#ltb_v10##cil_operator, Standard_io)
   SIMPOS の標準入出力のインスタンスに対してオペレーターを設定する。

2. :declare_parsing_operator(#ltb_v10##cil_operator, Standard_input)
   SIMPOS の標準入力のインスタンスに対してオペレーターを設定する。

3. :declare_unparsing_operator(#ltb_v10##cil_operator, Standard_output)
   SIMPOS の標準出力のインスタンスに対してオペレーターを設定する。

4.7 cil_pretty_printer のメソッド

このクラスは SIMPOS の標準入出力と共に継承して使う部品クラスである。

1. :pp(Pretty_printer, Term)
   :pp(Pretty_printer, Term, CVI)
   :pp(Pretty_printer, Term, CVI, ^NVI)
   Pretty_printer は cil_pretty_printer を継承したクラスのインスタンスであり, そのインスタンスに Term を表示する。変数情報付で表示する場合は、必要に応じて CVI や NVI を引き数に加える。

2. :indention_size(Pretty_printer, Size)
   プリティ・プリントの段落付けの幅を設定する。既定値は 6 である。Size は 0 以上の整数を指定する。

4.8 cil_compiler のメソッド

1. :create(#ltb_v10##cil_compiler, ^Compiler)
   インスタンスを生成する。

2. :set_state(Compiler, State)
   コンパイラの状態を設定する。State はオプションのリストで指定する。オプションには esp_source_file(ON_OFF), optimize(ON_OFF), public_declaration(ON_OFF) がある。変数 ON_OFF にはアトムの on または off を指定する。既定値は [esp_source_file(off), optimize(on), public_declaration(off)] である。

3. :compile(Compiler, File_name)
   :compile(Compiler, File_name, Package_name)
   CIL ソース・プログラムをコンパイルする。Package_name には生成したクラスを入れるパッケージ名をアトムで指定する。Package_name を指定しない場合はデフォルト・パッケージに登録される。コンパイルすると、ファイル名のタイプが cmp となったファイルが生成される。コンパイルド・プログラムを実行する時にはこのファイルを consult_on する。パッケージの情報はこのファイル内に保存されている。

4. :set_window(Compiler, Window)
   CIL のコンパイラが出力するエラー・メッセージのウインドウを設定する。ウインドウは、標準入出力でなければならない。このメソッドでウインドウを設定していない場合、エラー・メッセージはカーネルのデフォルト・ウインドウに表示される。

4.9 cil_builtin のメソッド

副作用の無い CIL 組込述語をクラス・メソッドとして提供する。メソッド・コールの一般形は
:述語名 (#ltb_v10##cil_builtin, その述語の引き数) である。
例えば unify_cil(X,Y) は :unify_cil(#ltb_v10##cil_builtin, X, Y) となる。以下のメソッドにおける引き数の仕様は組込述語と同じである。

1. :assign(#ltb_v10##cil_builtin,A,B,T)
2. :buffer(#ltb_v10##cil_builtin,P,Buffer)
3. :delete(#ltb_v10##cil_builtin,L,P1,P2)
4. :dif(#ltb_v10##cil_builtin,A,B)
5. :d_merge(#ltb_v10##cil_builtin,P1,P2)
6. :equal(#ltb_v10##cil_builtin,X,Y,V,R,W)
7. :extend(#ltb_v10##cil_builtin,P1,P2,D)
8. :frontier(#ltb_v10##cil_builtin,P1,P2,D)
9. :fullCopy(#ltb_v10##cil_builtin,X,Y)
10. :getRole(#ltb_v10##cil_builtin,P,L,V)
11. :glue(#ltb_v10##cil_builtin,P1,P2)
12. :locate(#ltb_v10##cil_builtin,P,L,V)
13. :masked_merge(#ltb_v10##cil_builtin,P1,P2,P3)
14. :match(#ltb_v10##cil_builtin,A,B,D)
15. :meet(#ltb_v10##cil_builtin,P1,P2,D)
16. :merge(#ltb_v10##cil_builtin,P1,P2)

17. :mk_assoc(#ltb_v10##cil_builtin,A,P)
18. :partial(#ltb_v10##cil_builtin,P)
19. :record(#ltb_v10##cil_builtin,P, Record)
20. :role(#ltb_v10##cil_builtin,L,P,V)
21. :same(#ltb_v10##cil_builtin,X,Y)
22. :setOfKeys(#ltb_v10##cil_builtin,P,Set)
23. :subpat(#ltb_v10##cil_builtin,P1,P2,D)
24. :t_equal(#ltb_v10##cil_builtin,I,J,C,R,W)
25. :t_merge(#ltb_v10##cil_builtin,P1, P2)
26. :t_subpat(#ltb_v10##cil_builtin,P1, P2)
27. :unify_cil(#ltb_v10##cil_builtin,X,Y)

参考文献

[1] M. Dincbas, H. Simonis, and P.V. Hentenryck. Extending equation solving and constraint handling in logic programming. Internal Report IR-LP-2203, ECRC, 1987.

[2] T. Amanuma, T. Suzuki, T. Okunishi, and K. Mukai. CIL Programming kankyô (CIL Programming Environment (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*, pages 211–214, 1988.

[3] L. Cardelli. Typechecking Dependent Types and Subtypes. Technical report, DEC Systems Research Center, 1987.

[4] K. Mukai. Horn Clause Logic with Parameterized Types for Situation Semantics Programming. Technical Report 101, ICOT, 1985.

[5] K. Mukai. Unification over Complex Indterminates in Prolog. Technical Report 113, ICOT, 1985.

[6] K. Mukai. Anadic Tuples in Prolog. Technical Report 239, ICOT, 1987.

[7] K. Mukai. A System of Logic Programming for Linguistic Analysis based on Situation Semantics. In *Proceedings of the workshop on semantic issues in human and computer languages*. CSLI, 1987.

[8] K. Mukai. Partially Specified Term in Logic Programming for Linguistic Analysis. In *Proceedings of the International Conference on FGCS '88*, 1988.

[9] K. Mukai and H. Yasukawa. *Complex Indeterminates in Prolog and its Application to Discourse Models*, volume 3, pages 441–466. OHMUSHA, ltd. and Spring Verlag, 1985.

[10] J. Reynolds. Three approaches to type structures. *Lecture Note in Computer Science*, volume 185, Springer Verlag, 1985.

[11] K. Sakai and Y. Sato. Boolean Gröbner bases. Technical Memo 488, ICOT, 1988.

[12] T. Chikayama. ESP Reference Manual. Technical Report 044, ICOT, 1984.

[13] A. Colmerauer. Prolog-II: Reference Manual and Theoretical Model. Internal report, Group Intelligence Artificielle, Université d'Aix-Marseille II, 1982.

[14] ICOT. PSI/SIMPOS Editor Manual. 1988.

[15] T. Miyazaki and K. Taki. Multi-PSI ni okeru Flat GHC no Jitsugen Hôshiki (Installation of Flat GHC on Multi-PSI (*in Japanese*)). Technical Report 190, ICOT, 1986.

[16] K. Ueda. Guarded Horn Clauses. Technical Report 103, ICOT, 1985.

[17] J. Barwise. Recent Developments in Situation Semantics. In M. Nagao, editor, *Language and Artificial Intelligence*, pages 387–399, Amsterdam, 1987. North-Holland.

[18] J. Barwise and J. Perry. *Situations and Attitudes*. MIT Press, Cambridge, 1983.

和文索引

英文索引

【特殊】

【A】

【B】

【C】

【D】

【E】

【T】

【U】

【V】

【W】

LTB 操作説明書

– LAX 編 –

(LTB1.1 版)

ICOT 第二研究室

平成元年 3 月 31 日

本説明書は形態素意味解析システム LAX(Logic based morphlogical and semantic Analizer for saX) の操作方法について記述している。

- LAX の概要
- LAX の起動と終了
- 辞書の作成手順
- 辞書エディタ
- インスペクタ
- プリティ・プリンタ

汎用日本語処理系 LTB(Language Tool Box) の説明書は次のような構成になっている。

- LTB 概要編
- LTB マスタ辞書編
- CiL 編
- LAX 編
- SAX 編
- 文生成編

目次

図目次

第 1 章

LAX の概要

本章では、

- LAX の概要、目的
- 形態素意味解析の概要
- LAX の機能概要

について述べる。

1.1 システムの概要

形態素意味解析システム LAX(Logic based morphlogical and semantic Analyzer for saX)[1] とは、分かち書きされていない日本語文から単語の切り出しを行ない、語の意味の構成を行なうプログラムとその開発環境をいう。形態素解析プログラムは、基本的には 3 型の文法を受理する非決定性有限オートマトンである。また、文法記述形式（辞書記述形式）は、いわゆる森岡の形態素文法 (3.2を参照のこと)を記述し易い形に設定した。

LAX では、この記述形式に則って書かれた辞書を、トランスレータにより形態素解析プログラムと意味構成プログラムとに変換する。その基本アルゴリズムは決定的に動作し、GHC 等の並列論理型言語での実行が可能であるような形態（レイヤード・ストリーム）を採用している。今回リリースする LAX は、この基本アルゴリズムを逐次処理向けに修正したものを用い、そのトランスレータを中心に文法開発のためのツールを備えている。LAX のツールには、文法の作成や修正を自由に行なえる辞書エディタ、形態素解析結果の分析を行なうインスペクタ、解析結果を見やすく表示するプリティ・プリンタがある。

解析結果は構文意味解析システムSAXの入力となる。また、LAX は単独のツールとしても使用でき、他のシステムの形態素意味解析部としても使用できる。

1.2 形態素意味解析の概要

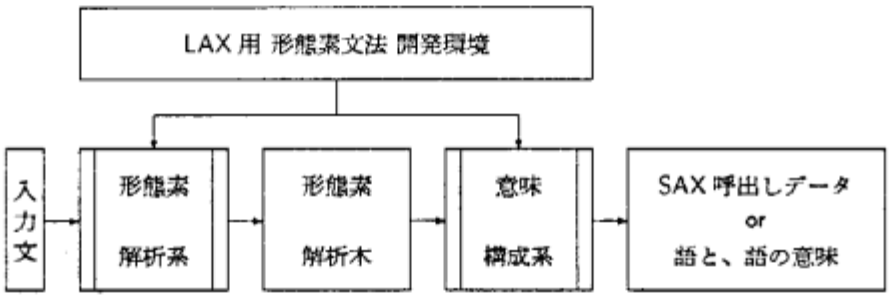


図 1.1: LAX のフェーズ

図 1.1のように LAX では解析を 2 つのフェーズで行う。一つは形態素解析部である。ここでは入力文字列を辞書エントリと見比べて形態素の候補をすべて挙げ、それらの内から可能な組み合せを調べて語の系列を作りだす。形態素解析では一般にかなりの形態的な曖昧さが生じる。もう一つは意味構成部である。形態素解析結果を元に、各語の『意味』を構成的に決定する。LAX では形態素の連接情報以外の情報を意味情報として扱うため、構文情報もこの意味構成部において処理される。意味構成に失敗した語は捨てられるので、この段階である程度の曖昧さの解消が行なわれる。

意味構造はユーザに解放されており、CILの PSTをはじめ任意のデータ構造を使用できるようになっている。解析された語とその意味の系列は、SAXの入力データの形式になり、曖昧さが反映された形でSAX に送られる。また、SAX 以外の構文解析システムを使用する場合など各自の用途に合わせてどのような構造でも出力できる。

1.3 LAX のツール構成

1.3.1 文法開発環境

図 1.2に、LAX の文法開発環境を示す。形態素意味辞書は中間形式辞書ファイルと呼ばれるいくつかのファイルに分割されて管理される。形態素意味解析用のプログラムはこの形態素意味辞書からジェネレータを用いて生成される。従って LAX における形態素意味解析プログラムの開発とは、この形態素意味辞書の開発を意味する。中間形式

[1]LAX の現バージョンでは、意味構成系に対するツールはサポートされていない

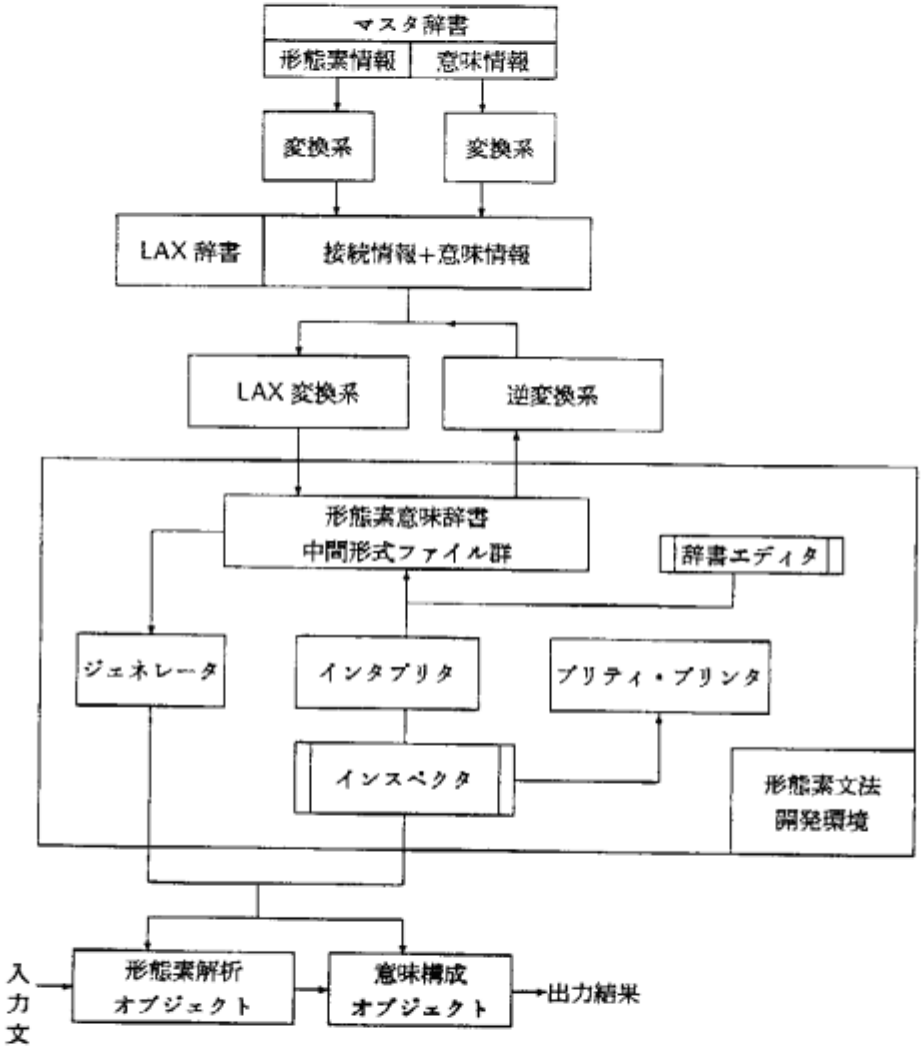


図 1.2: LAX 文法開発環境

辞書ファイルは、 LAX 辞書から変換系（トランスレータ）を用いて作成するか、または辞書エディタを用いて作成することができる。 はじめに LAX システムで使用される変換系について説明する。

1. マスタ辞書トランスレータ
   マスタ辞書に記述された形態的情報とその意味的情報を LAX 辞書の記述形式に変換するためのツール。現在、マスタ辞書に記述すべき形態的意味的情報の分析を行なっている段階なので具体的な変換系としてはまだ存在していない。次の版からリリースする予定である。

2. トランスレータ
   LAX 辞書の記述形式で書かれた辞書を、形態素意味辞書に変換する。 LAX 辞書の記述形式については 3.1の文法記述形式を参照のこと。辞書エディタを通して形態素意味辞書を作成、または修正していても、見かけ上は LAX 辞書のシンタクスでユーザは辞書を利用して、デバッグや文法の整理に役立てることである。逆トランスレートされた LAX 辞書はトランスレートすることにより形態素意味辞書に変換可能である。

3. ジェネレータ
   形態素意味辞書から形態素解析および意味構成プログラムを生成する。辞書に意味構成部の記述がなければ、形態素解析プログラムのみを生成する。辞書作成の段階ではインスペクタによるインタプリティブな解析で十分なので、ほとんど使用する必要はない。開発の最終段階にオブジェクトが必要な時に限り使用する。

1.3.2 各ツールの機能

LAX の主要なツールの機能を説明する。

1. トップ・ウインドウ
   LAX を起動直後に現われるツールで、２つの機能を有する。一つは前項で述べた各種変換系を起動する機能である。これには、最低限の辞書検索機能や解析機能、意味構成のトレース機能のほか、さらに全体の環境設定の値を変更したり、開発対象の辞書を変更したりする機能が含まれる。もう一つは以下の 2から 4までの専用ツールを起動し、制御を渡す機能である。各ツールはツールどうしでも行き来できるが、システムの終了はこのトップ・ウインドウからのみ可能である。

2. 辞書エディタ
   形態素意味辞書の内容の検索更新、削除、新規登録など、辞書のメインテナンスを行なうためのツール。基本的にカード型のデータベースを参考にデザインされている。エディット対象は仮想的に存在する LAX 辞書である。エディットの基本は、ホルダと呼ばれるカード(辞書の１エントリ、即ち１形態素を表示形式にしたもの)の集まりに対して行なう操作である。ホルダは検索により得られ、修正や削除は常にカレント・カード（エディタ・インタプリタ・ウインドウに表示されているカード）に対して行なわれる。エントリの追加は、カードにデータを記述し、追加のコマンドを発する事により実現できる。エントリの記述形式は LAX 辞書のそれと同一である。

3. インスペクタ
   形態素解析結果の簡単な表示、形態素候補やその辞書記述の表示、連接チェックなどの静的デバッグの機能を提供する。形態素の解析モードとインスペクト・モードからなる。また、インスペクタ全体を通じての解析エンジンの実行モードとして、形態素意味辞書を解釈実行するインタプリタ・モードと、コンパイルド・オブジェクトを使用するコンパイラ・モードを持つ。通常はインタプリタ・モードで実行し、前述の辞書エディタと連携して辞書の作成を行なう。

   (a) 解析モード
   入力文をキーボードから入力するか、または後述のセンテンス・セレクタを用いて入力用ウインドウに入力すると、形態素解析結果が結果表示用ウインドウに表示される。解析の可否、解析時間、入力文字数が表示され、その下に解析結果が表示される。解析結果は一行に一節が対応し、形態素の連接として表示される。形態的な曖昧さが生じている行は "/" により並立して表示される。モードの変更により、形態素の並びだけでなく、詳しい接続状況も表示できる。また接続に失敗した場合もその旨表示される。この状態で、入力ウインドウの入力文字中の一文字をマウス・クリックすることにより、次に説明するインスペクト・モードに移行する。

(b) インスペクト・モード
インスペクト・モードに移行すると、指定された文字から辞書引きされた形態素と、それらに連接する可能性がある形態素とを表示する。これらの形態素については自由に辞書記述の内容を表示できる。また特定の形態素に繋がる形態素を調べることもできる。さらに表示された形態素の任意のものを辞書エディタのホルダに積むことができ、辞書エディタに移行した時にスムーズな修正が可能である。

4. プリティ・プリンタ
形態素解析結果を，オールタナティブが分かり易いように表示するためのツール。文章の入力形式はインスペクタと同様である。

5. センテンス・セレクタ
インスペクタ、プリティ・プリンタで使用されるユーティリティ。特定のファイルに書かれたテキスト文データをセレクト・メニュー形式で選択できる。

第 2 章

LAX の操作方法

LAX の起動、終了、LAX の各ツールの操作方法を記述する。
なお、本章では、以下に示す省略表現、記号を使用している。

- 単なる“マウスで選択”は、“マウス左 1 回クリックで選択”を意味する。
- <CR> は、キャリッジ・リターンを意味する。
- <M-g> は、Meta キーを押しながら g を押下することを意味する。
- mouse#1 等の記号の意味は以下のとおりである。
  - mouse#1 : マウス左 1 回クリック
  - mouse#11 : マウス左 2 回クリック
  - mouse#m : マウス中 1 回クリック
  - mouse#mm : マウス中 2 回クリック

2.1 LAX の起動と終了

2.1.1 LAX の起動

LAX を起動する方法には 2 種類ある。

- システム・メニューから起動
- LTB シェル・ウインドウから起動

1. システム・メニューから起動

(a) LAX トップ・ウインドウの作成
マウス右 2 回クリックを行うと、システム・メニューが表示される。システム・メニューに表示された LAX を示す項目 (図 2.1の例では、"LAX") をマウス左 1 回クリックで選択する。"メニュー・ウインドウを開いて下さい" というテンポラリ・ウインドウが表示されるので、位置と大きさを適当に決め、LAX トップ・ウインドウを作成する。もし、ユーザの指定したウインドウの位置、大きさが不適当であれば、LAX が自動的にそれらを調整してウインドウを開く。ユーザは、この後もウインドウの位置と大きさを変更することができる (ただし、ウインドウをより大きくすることはできるが、より小さくはできない)。

(b) カレント辞書の選択
図 2.1に示す辞書名選択ウインドウが表示された場合、

- 既存の辞書の 1 つを使う場合は、辞書名選択ウインドウに表示された辞書の中からマウスで 1 つ選択する。
- 既存の辞書を使わないで、新しい中間形式辞書ファイル (16頁を参照のこと) を作成する場合は、辞書名選択ウインドウからマウスを外す。

なお、辞書名選択ウインドウが表示されるかどうかは、カレント環境保存ファイルに保存されている辞書選択指定の値による (19頁の環境設定を参照のこと)。辞書選択指定が "メニュー" 以外の場合は、辞書名選択ウインドウは表示されない。

(c) "- 中間辞書を読み込んでいます -" というメッセージがテンポラリ・ウインドウに表示され (解析モード (13頁を参照のこと) がインタプリタの場合)、しばらく後消えると、LAX の起動は完了する。

2.1.2 LAX の終了

LAX トップ・ウインドウの項目 "終了" をマウスで選択すると、LAX トップ・ウインドウが消え、LAX は終了する。

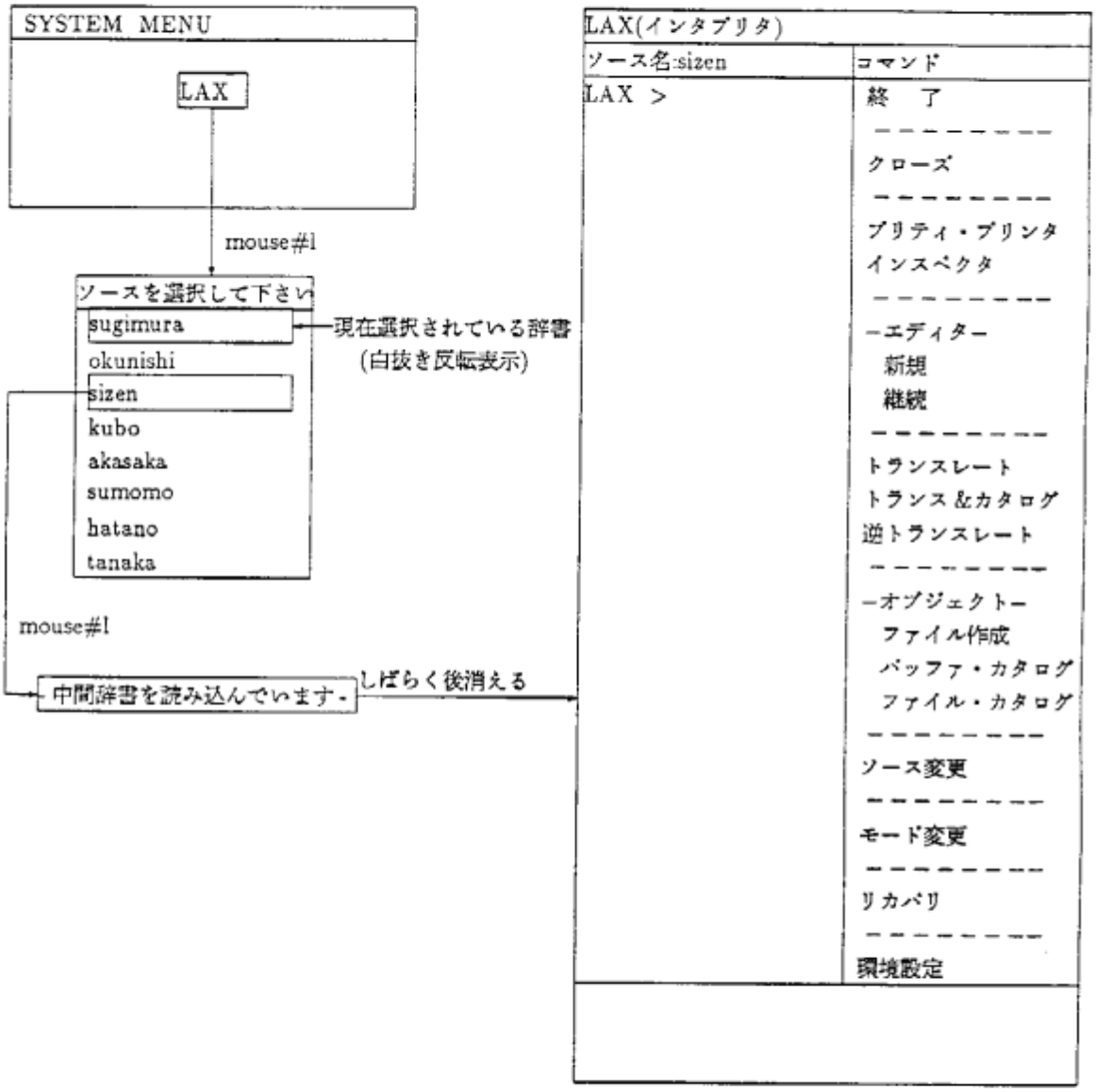


図 2.1: システム・メニューからの LAX の起動

2.2 LAX トップ・ウインドウでの操作

2.2.1 LAX トップ・ウインドウ

LAX トップ・ウインドウは、LAX を起動した時最初に表示されるウインドウである。 LAX トップ・ウインドウを図 2.2に示す。

LAX トップ・ウインドウは次の 5 つのウインドウからなる。

- ベース・ウインドウ
- 辞書名ウインドウ
- コマンド・ウインドウ
- インタプリタ・ウインドウ
- メッセージ・ウインドウ

1. ベース・ウインドウ
   ベース・ウインドウには、解析エンジン 18頁のプログラム生成を参照のこと) の動作モード(13頁のモード設定を参照のこと) が表示される。

2. 辞書名ウインドウ
   辞書名ウインドウには、カレント辞書(15頁の辞書の切り替えを参照のこと) の辞書名が表示される。

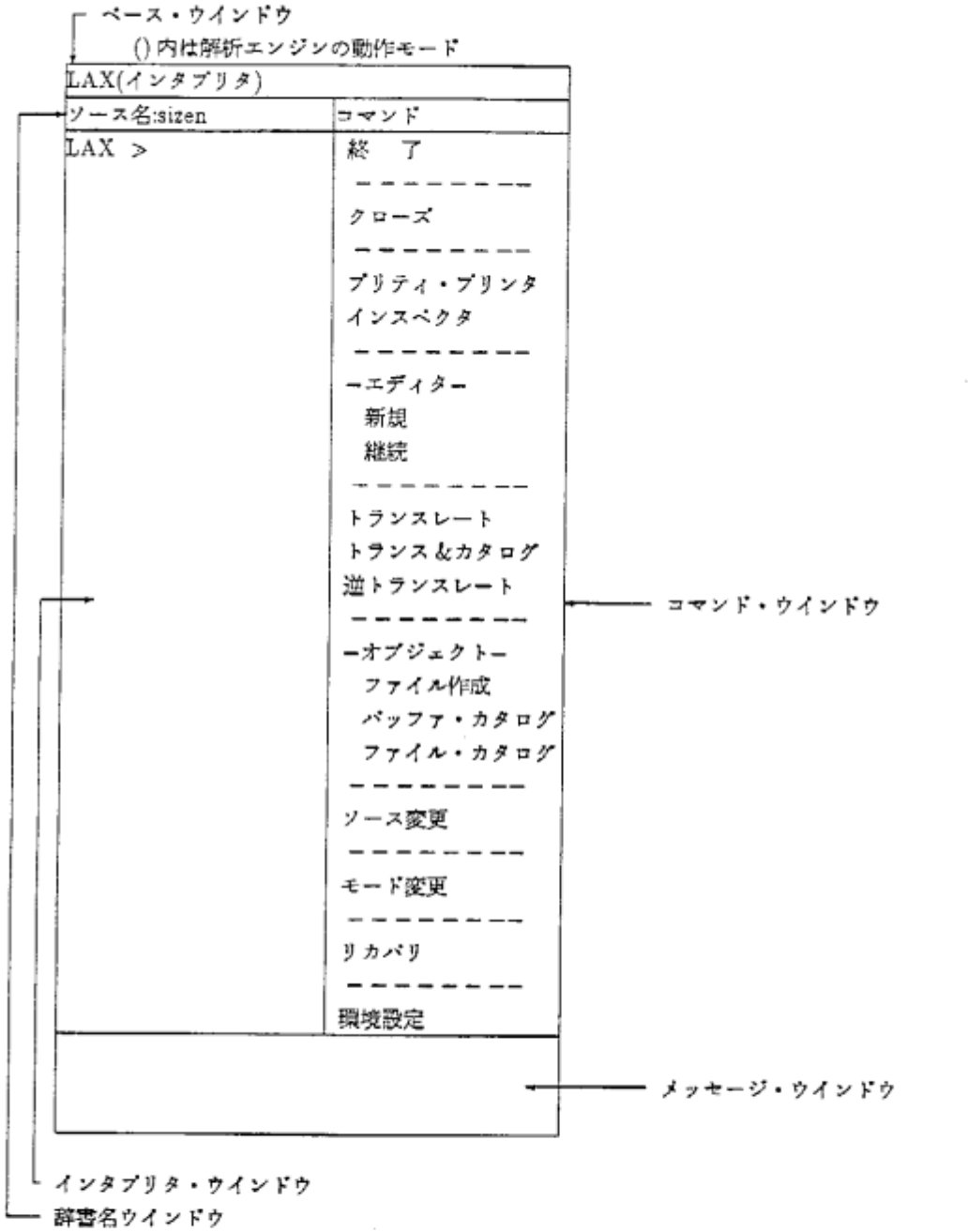
3. コマンド・ウインドウ
   コマンド・ウインドウには、トップ・ウインドウから実行可能なコマンドがメニュー項目として表示される。本ウインドウに表示された項目をマウスで選択することにより、コマンドを実行することができる。コマンドはインタプリタ・ウインドウ経由で実行される。
   なお、LAX トップ・ウインドウの大きさによっては、コマンド・ウインドウには一部のコマンドしか表示されないことがある。このときには、スクロールさせることによって、未表示コマンドを表示させることができる。

4. インタプリタ・ウインドウ
   インタプリタ・ウインドウは、コマンドの入力とその実行経過を表示するウインドウである。コマンド・ウインドウにおいて選択可能なコマンドは、対応するコマンドを本ウインドウから入力することによっても実行できる。

5. メッセージ・ウインドウ
   メッセージ・ウインドウは、コマンドの実行経過の表示と、LAX からユーザへの問い合わせ、ユーザからの応答の入力を行うウインドウである。

トップ・ウインドウからは、以下の各機能を実行することができる。

- モード設定
- 辞書の切り替え
- 辞書のトランスレート
- 辞書の逆トランスレート
- 解析エンジンの生成
- 以下の各ツールの呼び出し
  - 辞書エディタ
  - インスペクタ
  - プリティ・プリンタ



(注) ウインドウの大きさによっては、コマンド・ウインドウには一部コマンドしか表示されない。

図 2.2: LAX トップ・ウインドウ

2.2. LAX トップ・ウインドウでの操作

- その他
  - 辞書のリカバリ
  - 環境設定
  - トップ・ウインドウのアイコン化

以下では、これらの操作方法について説明する。

2.2.2 操作のしかた

モード設定

LAXの解析エンジンの動作モード(解析モード)の設定方法について述べる。解析エンジンの動作モードには、次の2種類がある。

- コンパイラ・モード
  コンパイラ・モードでは、コンパイル済みの解析エンジンが動作する。動作は、インタプリタ・モードに較べて速いが、辞書エディタで中間形式辞書ファイル(16頁を参照のこと)に修正を加えた後、本モードで解析エンジンを動作させるためには、中間形式辞書ファイルからオブジェクト・プログラム(解析エンジン)を作成し、それをコンパイルしなければならない。中間形式辞書ファイルを変更しない場合は、インタプリタ・モードよりも本モードのほうが適している。

- インタプリタ・モード
  インタプリタ・モードでは、コンパイルされていない解析エンジンがインタプリティブに動作する。動作は、コンパイラ・モードに較べて遅いが、辞書エディタで中間形式辞書ファイルに修正を加えた後でも、コンパイルする必要がないため、入力文の解析と中間形式辞書ファイルの修正を繰り返す場合は、コンパイラ・モードよりも本モードのほうが適している。

モードの設定は、以下のようにして行う。

コマンド・ウインドウで、項目“モード変更”を選択すると(又は、インタプリタ・ウインドウから同等のコマンド mode_change<CR> を入力すると)、解析モードに応じて次のように操作を行う。

1. 解析モードがインタプリタの場合

   (a) 中間形式辞書ファイルはあるが、対応するオブジェクト形式辞書ファイル(18頁を参照のこと)がない場合:
   図2.3に示す解析モード変更ウインドウが、マウス・マーカの位置に表示される。項目“ジェネレートする”を選択すると、中間形式辞書ファイルからオブジェクト・プログラムが生成され、更にコンパイルされる。終了すると、ベース・ウインドウのラベルが“コンパイラ”に変わり、メッセージ・ウインドウに“コンパイラ・モードに変わりました”というメッセージが表示される。項目“abort”を選択すると、解析モードは変わらず、メッセージ・ウインドウに“モード変更を中止します”というメッセージが表示される。

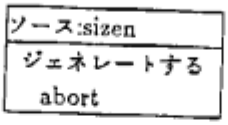


図2.3: 解析モード変更ウインドウ

   (b) 中間形式辞書ファイル、オブジェクト形式辞書ファイルが共にある場合:
   オブジェクト形式辞書ファイルには、解析エンジンを構成する複数のクラスが書かれている。

      i. クラスがカタログ済みの場合
      クラスが未ロードであればロードされる。ベース・ウインドウのラベルが“コンパイラ”に変わり、メッセージ・ウインドウに“コンパイラ・モードに変わりました”というメッセージが表示される。

      ii. クラスが未カタログの場合
      図2.4に示すカタログ指示ウインドウが表示される。項目“カタログする”を選択すると、クラスがコンパイル、カタログされ、終了するとベース・ウインドウのラベルが“コンパイラ”に変わり、メッセージ・ウインドウに“コンパイラ・モードに変わりました”というメッセージが表示される。
      項目“abort”を選択すると、解析モードは変わらず、メッセージ・ウインドウに“モード変更を中止します”というメッセージが表示される。

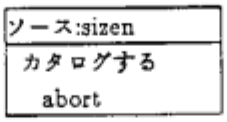


図 2.4: カタログ指示ウインドウ

2. 解析モードがコンパイラの場合
   解析モードがインタプリタに変わる。 ベース・ウインドウのラベルが“インタプリタ”に変わり、メッセージ・ウインドウに“インタプリタ・モードに変わりました”というメッセージが表示される。

3. 中間形式辞書ファイルが存在しない場合
   解析モードは変わらず、メッセージ・ウインドウに“モード変更を中止します”というメッセージが表示される。

辞書の切り替え

LAXが扱う辞書の切り替え方法について述べる(辞書の形式、ファイルのパス名については3.1.1を参照のこと)。ユーザは、LAXに複数の辞書を登録することができるが、インスペクタ等のLAXのモジュールは、一時点においてはそれらのうちのユーザが指定した1辞書を対象として動作する。この辞書をカレント辞書といい、辞書名表示ウインドウに表示されている。ユーザは、カレント辞書を現在割り当てられている辞書から別の辞書に切り替えることができる。

辞書の切り替えは、以下のようにして行う。

コマンド・ウインドウで、項目“ソース変更”を選択すると(又は、インタプリタ・ウインドウから同等のコマンド file_name<CR> を入力すると)、図2.5に示す辞書名選択ウインドウがマウス・マーカの位置に表示される。辞書名選択ウインドウでは、カレント辞書が反転表示されている。ただし、LAXに辞書が全く登録されていない場合は、“中間ファイルを持ったソースが存在しません”というメッセージがテンポラリ・ウインドウに表示される。

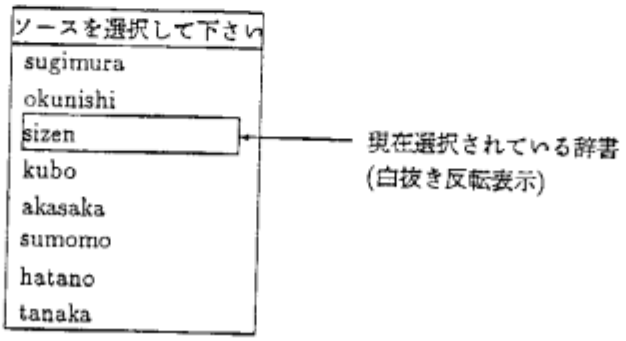


図 2.5: 辞書名選択ウインドウ

1. カレント辞書以外を選択した場合
   カレント辞書の辞書環境が、辞書環境保存ファイルに保存される。

   (a) 変更に成功した場合
   辞書名ウインドウに、選択された辞書の名前が表示され、メッセージ・ウインドウに“ソース xxx を設定しました”というメッセージが表示される。(xxx は選択された辞書名)。

   (b) 変更に失敗した場合
   何らかの理由で、中間形式辞書ファイルが壊れている場合、変更に失敗する。この時は、カレント辞書は変更されず、メッセージ・ウインドウに“設定に失敗しました。もとに戻します”というメッセージが表示される。

2. カレント辞書を選択した場合
   カレント辞書は変わらず、メッセージ・ウインドウに“変更がありません”というメッセージが表示される。

3. マウス・マーカを辞書名変更ウインドウから外した場合
   カレント辞書は変わらず、メッセージ・ウインドウに“変更がありません”というメッセージが表示される。

トランスレートと逆トランスレート

辞書のトランスレートと逆トランスレートの操作方法について述べる。 LAX が取り扱う辞書の形式には、次の 3 種類がある (3.1.1を参照のこと)。

- ソース形式辞書ファイル
- 中間形式辞書ファイル
- オブジェクト形式辞書ファイル

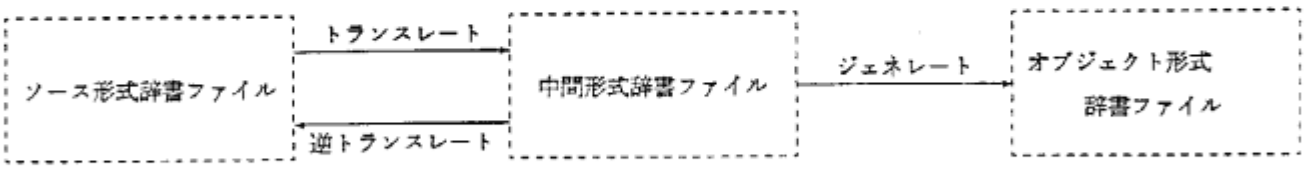


図 2.6: LAX の扱う辞書の形式

図 2.6に示すように、トランスレートは、ソース形式辞書ファイルから中間形式辞書ファイルを生成する。逆トランスレートは、トランスレートと反対に中間形式辞書ファイルからソース形式辞書ファイルを生成する。辞書のトランスレートと逆トランスレートは、以下のようにして行う。

なお、トランスレートにはトランスレートのみを行うコマンドとトランスレートとオブジェクト形式辞書ファイルのコンパイル、カタログを連続して行うコマンドがある。

1. トランスレート
   コマンド・ウインドウの項目 " トランスレート" をマウスで選択する (または、インタプリタ・ウインドウから同等のコマンド translate<CR> を入力すると)、図 2.7に示すように、LAX はユーザにメッセージ・ウインドウを使って " 入力ファイル名?: xxx" と尋ねてくる。ここで xxx には現在選択されている辞書名が表示されるので、ユーザはこの辞書をトランスレートする場合は単に <CR> を押下し、別の辞書をトランスレートする場合は、 xxx をその辞書名に変更して <CR> を押下する。トランスレートが終了すると、 " トランスレート終了しました " というメッセージが表示され、 LAX はユーザからの次の指示を待つ。

```
ファイルをクローズします
.....
入力ファイル名?: xxx<CR>
.....
トランスレート終了しました。
```

図 2.7: トランスレート

2. トランスレートとコンパイル、カタログ
   コマンド・ウインドウの項目 " トランスレート &カタログ" をマウスで選択する (または、インタプリタ・ウインドウから同等のコマンド trans_generate<CR> を入力すると)、図 2.8に示すように、LAX はユーザにメッセージ・ウインドウを使って " 入力ファイル名?: xxx" と尋ねてくる。ここで xxx には現在選択されている辞書名が表示されるので、ユーザはこの辞書をトランスレートする場合は単に <CR> を押下し、別の辞書をトランスレートする場合は、 xxx をその辞書名に変更して <CR> を押下する。コンパイルが終了すると、 " コンパイル終了しました " というメッセージが表示され、 LAX はユーザからの次の指示を待つ。

```
ファイルをクローズします
.....
入力ファイル名?: xxx<CR>
.....
コンパイル終了しました。
```

図 2.8: トランスレート、コンパイル、カタログ

3. 逆トランスレート
   コマンド・ウインドウの項目“逆トランスレート”をマウスで選択する(または、インタプリタ・ウインドウから同等のコマンド rev_translate<CR> を入力すると)、図 2.9に示すように、LAX はユーザにメッセージ・ウインドウを使って“出力パス名:LAX:SOURCE>xxx.lax”と尋ねてくる(LAX:SOURCE はディレクトリ名)。出力パス名とは、逆トランスレートによって作成されるソース形式辞書ファイルの名称である。xxx は現在選択されている辞書の名称である。ユーザは作成されるソース形式辞書ファイルのパス名がこれでよい場合は単に <CR> を押下し、別のパス名にする場合は、出力パス名を変更して <CR> を押下する。逆トランスレートが終了すると、“辞書逆変換を終了しました”というメッセージが表示され、LAX はユーザからの次の指示を待つ。

```
出力パス名: LAX:SOURCE>xxx.lax<CR>
辞書逆変換を開始します。
.....
辞書逆変換を終了しました。
```

図 2.9: 逆トランスレート

プログラム生成

オブジェクト・プログラムの生成方法について述べる。ここでいうオブジェクト・プログラムは、解析エンジンと同義である。オブジェクト・プログラムが収められているファイルをオブジェクト形式辞書ファイルという。 LAX が取り扱う辞書の形式には、次の 3 種類がある。

- ソース形式辞書ファイル
- 中間形式辞書ファイル
- オブジェクト形式辞書ファイル

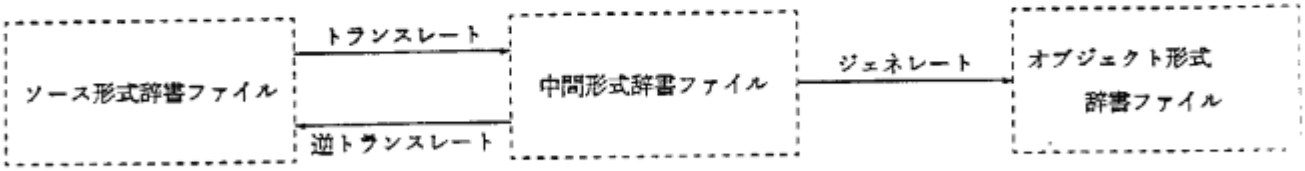


図 2.10: LAX の扱う辞書の形式

図 2.10に示すように、ジェネレータは、中間形式辞書ファイルからオブジェクト形式辞書ファイルを生成する。オブジェクト・プログラムの生成の方法には、次の 3 種類がある。

1. 中間形式辞書ファイルからオブジェクト形式辞書ファイル (内容はオブジェクト・プログラム、即ち解析エンジンのソース・プログラム) を生成
   コマンド・ウインドウの項目 “ファイル作成” をマウスで選択する (又は、インタプリタ・ウインドウから同等のコマンド `generate_file<CR>` を入力する) と、メッセージ・ウインドウに処理経過を示すメッセージが表示され、終了すると LAX はユーザからの次の指示を待つ。

2. 中間形式辞書ファイルから、オブジェクト・プログラムを主記憶上に生成し、コンパイル、カタログ (オブジェクト形式辞書ファイルは生成されない)
   コマンド・ウインドウの項目 “バッファ・カタログ” をマウスで選択する (又は、インタプリタ・ウインドウから同等のコマンド `generate<CR>` を入力する) と、メッセージ・ウインドウに処理経過を示すメッセージが表示され、終了すると LAX はユーザからの次の指示を待つ。

3. オブジェクト形式辞書ファイルをコンパイル、カタログ
   コマンド・ウインドウの項目 “ファイル・カタログ” をマウスで選択する (又は、インタプリタ・ウインドウから同等のコマンド `file_compile<CR>` を入力する) と、メッセージ・ウインドウに処理経過を示すメッセージが表示され、終了すると LAX はユーザからの次の指示を待つ。

その他の機能

1. 環境設定
   ここでいう環境設定とは、次のような項目の値を変更することである。

   (a) 表層表示選択
   辞書エディタのウインドウに形態素の表層を表示する時、トークンも共に表示するか、表層のみ表示するかの別を設定する項目である。表層とトークンが同じ場合は、表層のみ表示すればよい。

   (b) オブジェクト・プログラムのセーブ時期
   コンパイル、カタログされたオブジェクト・プログラム (解析エンジン) をどの時点で (SIMPOS のライブラリアンの機能を使用して) セーブするかを設定する項目である。次の 2 種類の値を取り得る。一般に、一回の LAX 起動から終了の間に複数の辞書からオブジェクト・プログラムを作成する場合は、カタログする毎にセーブするほうがよい。1 つの辞書のみ取り扱う場合は、LAX 終了時にセーブすればよい。

   - カタログ直後
     オブジェクト・プログラムをカタログした後、引き続いてセーブ
   - LAX 終了時
     ユーザが、コマンド・ウインドウの項目“終了”を選択したとき

   (c) バックアップ作成
   中間形式辞書ファイルのバックアップ・ファイルを作成するか否かを設定する項目である。バックアップを取ることにすると、LAX 終了時 (コマンド・ウインドウの項目“終了”が選択されたとき) に中間形式辞書ファイルのコピー・ファイルが生成される。

   (d) 辞書選択指定
   ユーザは、LAX に複数個の辞書を登録できる。 LAX が起動された時、最初に取り扱う辞書をどのようにして指定するかを設定する項目である。次の 3 通りの指定方法がある。

   - LAX に登録された複数個の辞書の名前をメニュー・ウインドウに表示してそのなかから 1 つ選択する
   - 最も最近使った辞書とする
   - 既存の辞書を使わない

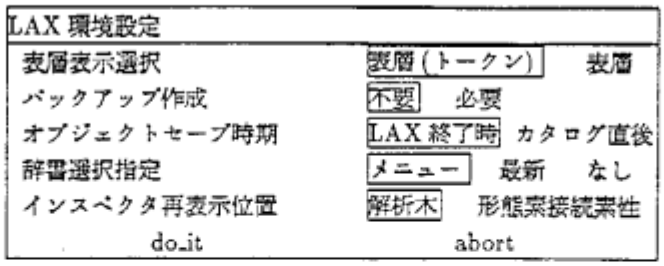
   (e) インスペクタ再表示位置
   インスペクタから辞書エディタを呼び出して、再びインスペクタに戻る時、インスペクタのどのウインドウに戻るかを設定する項目である。次の 2 通りがある。

   - 解析木表示ウインドウに戻る
   - 形態素接続情報表示ウインドウに戻る

   これらの環境値は、LAX 終了時の値がカレント環境保存ファイルに自動保存されるので、次回 LAX を起動する時にそのまま使用される。
   LAX をインストールした直後の環境値は、LAX が省略値として採用した値が設定されている。

   環境設定は、次のようにして行う。
   コマンド・ウインドウの項目“環境設定”をマウスで選択 (又は、インタプリタ・ウインドウから同等のコマンド `laxenv_mode<CR>` を入力) すると、図 2.11に示す環境設定ウインドウが表示される。このウインドウでは、選択されている項目値が白抜き反転表示されている。マウスで項目値をクリックすると、その値が白抜き反転表示される。この後、マウスで項目“do_it”を選択すると、ウインドウが消え、環境設定が終了する。マウスで項目“abort”を選択すると、値は変更されない。

2. 辞書のリカバリ
   環境設定で“バックアップ作成”を“必要”に設定しておくと、LAX 終了時、自動的に中間形式辞書ファイルのコピー・ファイルが作られる。辞書のリカバリでは、コピー・ファイルを利用して中間形式辞書ファイルを復元する。
   辞書のリカバリは、コマンド・ウインドウの項目“リカバリ”をマウスで選択 (又は、インタプリタ・ウインドウから同等のコマンド `recovery<CR>` を入力) して、次のように行う。



(注) [ ] は、白抜き反転表示されていて、現在の環境値を示す。

図 2.11: 環境設定ウインドウ

- コピー・ファイルが存在する場合
  リカバリが正常に終了した場合は、メッセージ・ウインドウに“リカバリされました”というメッセージが表示される。
- コピー・ファイルが存在しない場合
  メッセージ・ウインドウに“バックアップ・ファイルが存在しません”というメッセージが表示され、リカバリはなされない。

3. LAX トップ・ウインドウのアイコン化
   コマンド・ウインドウの項目“クローズ”をマウスで選択する (又は、インタプリタ・ウインドウから同等のコマンド `icon<CR>` を入力する) と、LAX トップ・ウインドウはアイコン表示される。

各ツールの呼び出し

LAX トップ・ウインドウから、以下のツールを呼び出すことができる。

1. 辞書エディタの呼び出し (⇒ 2.3.1)
2. インスペクタの呼び出し (⇒ 2.3.2)
3. プリティ・プリンタの呼び出し (⇒ 2.3.3)

2.3 各ツールの操作

2.3.1 辞書エディタ

辞書エディタの操作方法を説明する。辞書エディタは中間形式辞書ファイルを処理の対象とする辞書編集ツールである。辞書エディタは、既存の中間形式辞書ファイルを更新することも、新しく中間形式辞書ファイルを作成することもできるが、両者ではその操作方法に幾分の違いがある。以下では、操作方法に違いがある所では、その違いを明記する。

起動方法

辞書エディタの起動方法には、LAX トップ・ウインドウから起動する方法と、インスペクタから起動する方法がある。

1. LAX トップ・ウインドウからの起動
   既存の中間形式辞書ファイルを更新するか、新しい中間形式辞書ファイルを作成するかによって起動方法が異なる。

   (a) 既存の中間形式辞書ファイルの更新(図 2.12を参照のこと)
   既存の中間形式辞書ファイルを更新する場合は、以下のようにして辞書エディタを起動する。
      i. LAX トップ・ウインドウ (図 2.2) で、項目“エディタ”の下の項目“継続”をマウス左1回クリックで選択する。
      ii. “エディタ・ウインドウを開いて下さい”というテンポラリ・ウインドウが表示される[1]ので、ウインドウの位置と大きさを適当に決めウインドウを開く。もし、ユーザの指定したウインドウの位置、大きさが不適当であれば、辞書エディタが自動的にそれらを調整してウインドウを開く。ユーザは、辞書エディタ起動中いつでも、ウインドウの位置と大きさを変更することができる (ただし、ウインドウをより大きくすることはできるが、より小さくはできない)。
      iii. 次に、“エディタ準備中です。しばらくお待ち下さい”というテンポラリ・ウインドウが表示され、このウインドウが消えると辞書エディタの初期ウインドウが表示される。

   しかし、ユーザが指定した辞書が中間形式辞書ファイルを持っていない場合、LAX トップ・ウインドウの一部であるメッセージ・ウインドウに、“中間ファイルを持ったソースを指定して下さい”というメッセージが表示され、辞書エディタは起動されない。ユーザは、LAX トップ・ウインドウの項目“ソース変更”を選択することによって、中間形式辞書ファイルを持った辞書名を指定した後、再び項目“継続”を選択する。

   また、カレント辞書が設定されていない場合、辞書名選択ウインドウ (図 2.5) が、表示されるので辞書を一つ選択する (上記 1(a)ii, 1(a)iiiに続く)。

   (b) 新しい中間形式辞書ファイルの作成(図 2.13を参照のこと)
   新しい中間形式辞書ファイルを作成する場合は、以下のようにして辞書エディタを起動する。
      i. LAX トップ・ウインドウ (図 2.2) で、項目“エディタ”の下の項目“新規”をマウス左1回クリックで選択する。
      ii. “既存ファイルをクローズしています”というメッセージがテンポラリ・ウインドウに表示され、しばらくして消える。
      iii. “エディタ・ウインドウを開いて下さい”というテンポラリ・ウインドウが表示される[2]ので、ウインドウの位置と大きさを適当に決めウインドウを開く。もし、ユーザの指定したウインドウの位置、大きさが不適当であれば、辞書エディタが自動的にそれらを調整してウインドウを開く。ユーザは、辞書エディタ起動中いつでも、ウインドウの位置と大きさを変更することができる (ただし、ウインドウをより大きくすることはできるが、より小さくはできない)。
      iv. 次に、“エディタ準備中です。しばらくお待ち下さい”というテンポラリ・ウインドウが表示され、このウインドウが消えると辞書エディタの初期ウインドウが表示される。

[1] LAX 起動後、最初に辞書エディタを呼び出した時のみ。既に、辞書エディタを起動・終了した後ならば、その時作られたウインドウが使われるので新たにウインドウを開く必要はない

[2] 同

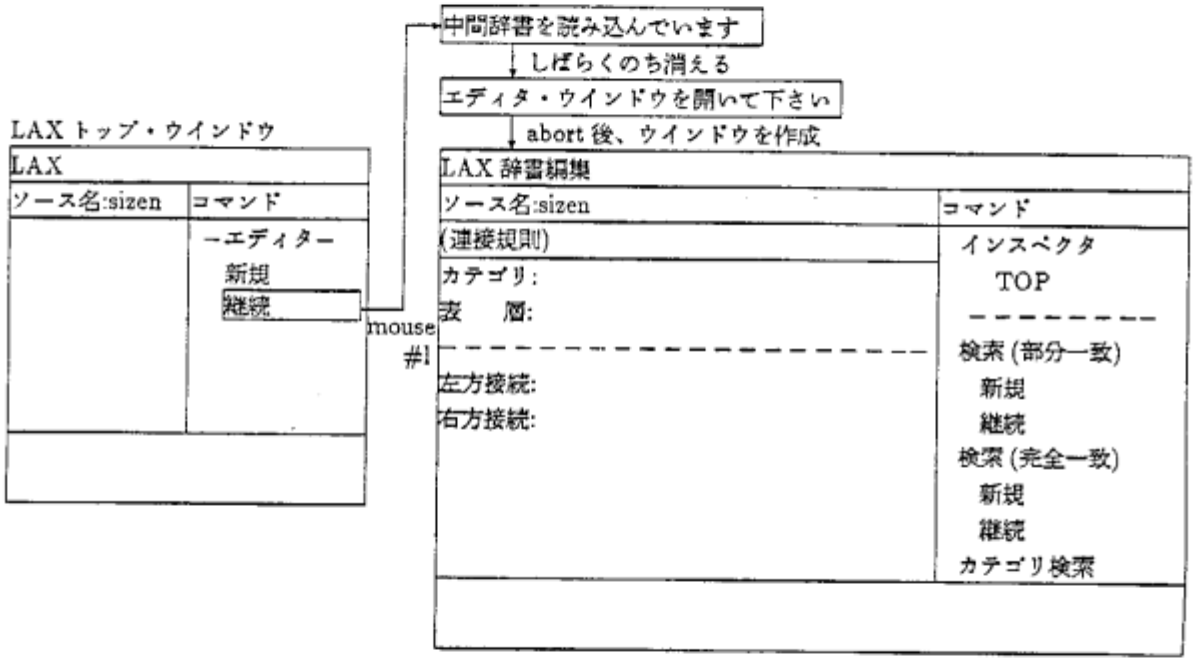


図 2.12: 既存の中間形式辞書ファイル更新時の辞書エディタの初期ウインドウ

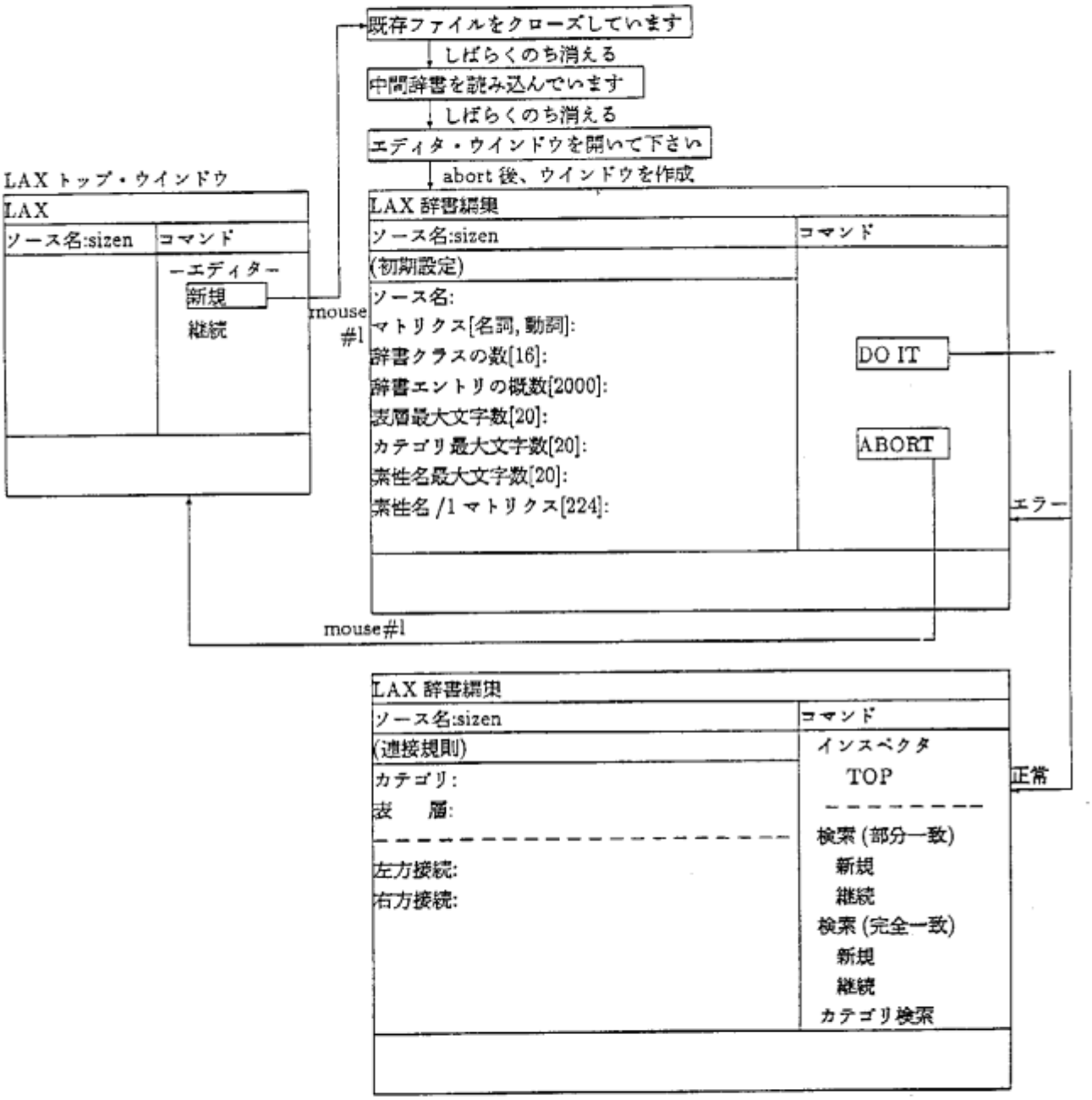


図 2.13: 新しい中間形式辞書ファイル作成時の辞書エディタの初期ウインドウ

v. 中間形式辞書ファイルの初期設定

新しく辞書ファイルを作成する場合、辞書環境の初期設定を行う必要がある。以下にその方法を述べる。初期設定のためのウインドウは図 2.13に示すとおりである。このウインドウには、初期設定を要する項目が省略値([ ]内) と共に表示されている。ユーザは、:の後に値を入力することにより、以下に示す辞書環境項目に値を設定する。なお、省略値のとおりでよいなら項目 "DO IT" をマウスで選択するだけでよい。以下の値の設定で、<CR>は必要ない。

- ソース名
  辞書名(3.1.1を参照のこと)を文字列で指定する。辞書名はLAXに未登録のものでなければならない。
- マトリクス
  マトリクスをプロローグ・リストの形式で指定する (例:[派生, 陳述])。
- 辞書クラスの数
  オブジェクト・プログラムを構成するクラス数を正の整数で指定する。
- 辞書エントリの概数
  辞書に登録しようとする形態素の大まかな個数を正の整数で指定する。
- 表層最大文字数
  辞書に登録しようとする形態素の表層の最大長 (文字数) と等しいかより大きい正の整数を指定する。
- カテゴリ最大文字数
  辞書に登録しようとする形態素のカテゴリの最大長 (文字数) と等しいかより大きい正の整数を指定する。
- 素性名最大文字数
  辞書に登録しようとする形態素の素性名の最大長 (文字数) と等しいかより大きい正の整数を指定する。
- 素性数/1マトリクス
  1マトリックスに属する素性の最大数と等しいかより大きい正の整数を指定する。

以上の項目に値を設定した後、

A. 項目 "DO IT" を選択すると、
   - ユーザの設定値が正しい場合
     エディタ・メッセージ・ウインドウに " ソース名: xxx 設定終了しました。 " というメッセージが表示され、初期設定が終了し、辞書環境設定ウインドウに換わってエディタ・インタプリタ・ウインドウが表示される。
   - ユーザの設定値に誤りのある場合
     エディタ・メッセージ・ウインドウに、エラー・メッセージが表示され再度入力待ちになる。

B. 項目 "ABORT" を選択すると、初期設定は中止され辞書エディタのウインドウが消える。

(c) インスペクタからの起動(図 2.14を参照のこと)

インスペクタ起動中に、動的に辞書エディタを起動することができる。

i. インスペクタの形態素接続情報表示ウインドウ(左方または右方) に表示されている形態素をマウス中2回クリックで選択する。詳細は、 2.3.2を参照のこと。

ii. " エディタ・ウインドウを開いて下さい " というテンポラリ・ウインドウが表示されるので、ウインドウの位置と大きさを適当に決めウインドウを開く。もし、ユーザの指定したウインドウの位置、大きさが不適当であれば、辞書エディタが自動的にそれらを調整してウインドウを開く。ユーザは、辞書エディタ起動中いつでも、ウインドウの位置と大きさを変更することができる (ただし、ウインドウをより大きくすることはできるが、より小さくはできない)。

iii. 次に、" エディタ準備中です。しばらくお待ち下さい " というテンポラリ・ウインドウが表示され、このウインドウが消えると辞書エディタの初期ウインドウが表示される。

iv. インスペクタから受け渡されたホルダ(37頁を参照のこと) に登録されている形態素の1つが、辞書エディタのウインドウに表示されているので、ユーザは中間形式辞書ファイルの検索を行うことなく、形態素の修正等を行うことができる。

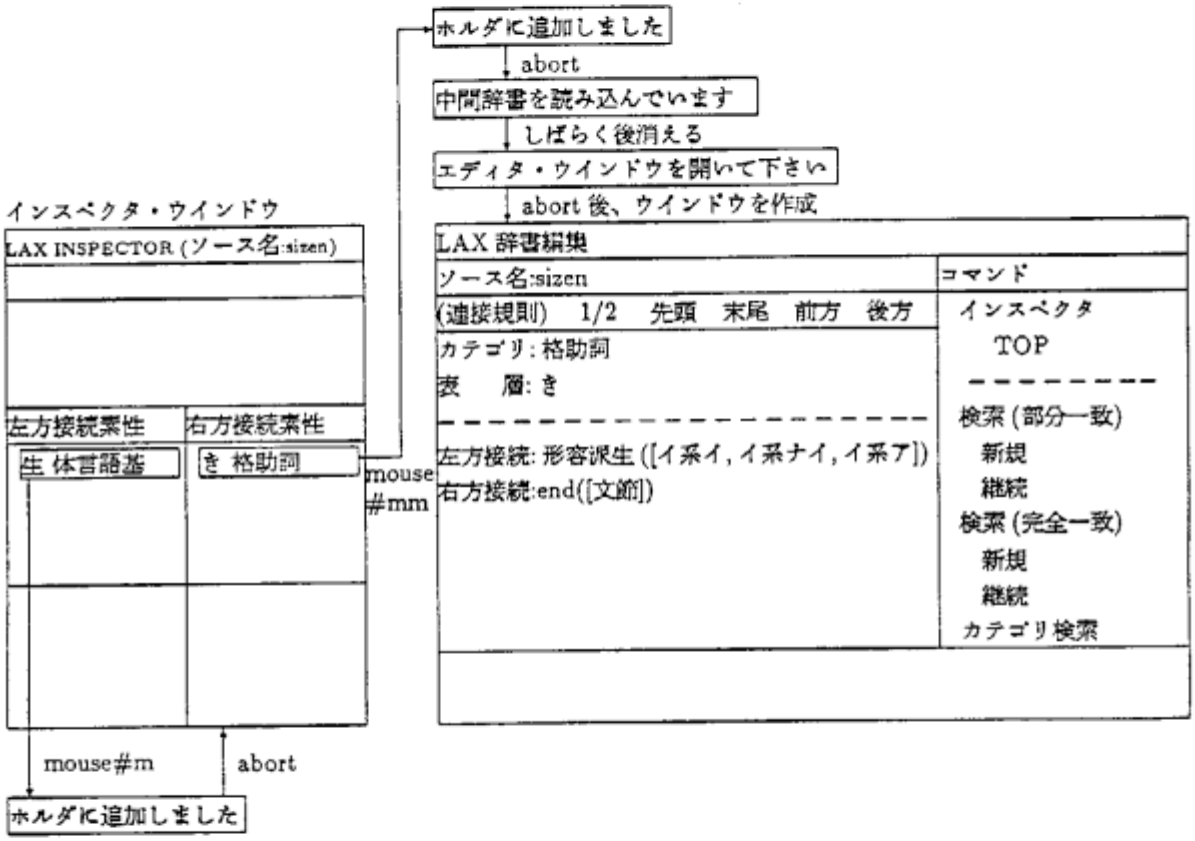


図 2.14: インスペクタから呼び出した辞書エディタの初期ウインドウ

辞書エディタのウインドウ

辞書エディタのウインドウは、1 つのメイン・ウインドウと 3 つのサブ・ウインドウから成る。

1. メイン・ウインドウ
   図 2.15にメイン・ウインドウを示す。

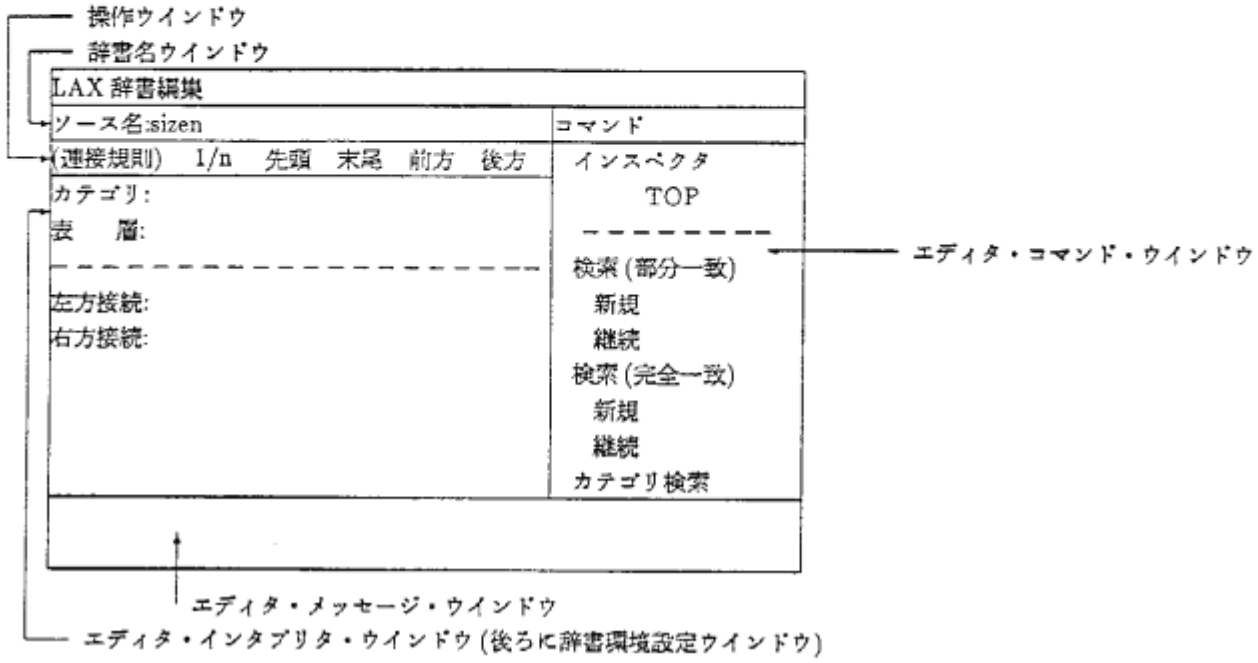


図 2.15: 辞書エディタのメイン・ウインドウ

メイン・ウインドウは実際には 6 つのウインドウから成る。

- 辞書名ウインドウ
- 操作ウインドウ
- エディタ・コマンド・ウインドウ
- 辞書環境設定ウインドウ
- エディタ・インタプリタ・ウインドウ
- エディタ・メッセージ・ウインドウ

これらのうち、エディタ・インタプリタ・ウインドウと辞書環境設定ウインドウは、同時に表示されることはなく、どちらか一方が表示される。

2. サブ・ウインドウ
   メイン・ウインドウ以外に次の 3 つのウインドウがある。

- カテゴリ選択ウインドウ
- 表層選択ウインドウ
- 操作コマンド付き表層選択ウインドウ

メイン・ウインドウ　メイン・ウインドウを構成する各ウインドウについて述べる。

辞書名ウインドウ　本ウインドウには、現在処理対象となっている辞書の名称が表示される。

操作ウインドウ　図 2.16に操作ウインドウを示す。

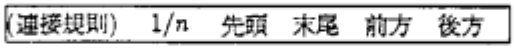


図 2.16: 操作ウインドウ

ユーザは、本ウインドウからエディタ・インタプリタ・ウインドウに表示されている形態素を操作する。

1. "(連接規則)"
   現在、辞書エディタが連接規則を処理していることを示している。入力項目ではない。

2. i/n
   nは、現在ホルダ(37頁を参照のこと)に登録されている形態素の個数。iは、現在エディタ・インタプリタ・ウインドウにそのうち何番目が表示されているかを示している。入力項目ではない。

3. 前方
   本項目をクリックすると、$i-1$番目の形態素がエディタ・インタプリタ・ウインドウに表示される。ただし、1番目の形態素が表示されている時に、本項目をクリックするとn番目の形態素がエディタ・インタプリタ・ウインドウに表示される。

4. 後方
   本項目をクリックすると、$i+1$番目の形態素がエディタ・インタプリタ・ウインドウに表示される。但し、n番目の形態素が表示されている時に、本項目をクリックすると1番目の形態素がエディタ・インタプリタ・ウインドウに表示される。

5. 先頭
   本項目をクリックすると、1番目の形態素がエディタ・インタプリタ・ウインドウに表示される。

6. 末尾
   本項目をクリックすると、n番目の形態素がエディタ・インタプリタ・ウインドウに表示される。

エディタ・コマンド・ウインドウ　ユーザは辞書エディタに対する指示の大部分を、エディタ・コマンド・ウインドウに表示されたコマンドをマウスでクリックすることにより行う。エディタ・コマンド・ウインドウの表示形式は、

- 辞書環境の初期設定時および変更時
- 中間形式辞書ファイルの更新時
- 接続素性書き換え時

のそれぞれで変化する。その詳細は以下のとおりである。

1. 辞書環境の初期設定時および更新時のエディタ・コマンド・ウインドウ
   辞書環境の初期設定時および変更時におけるエディタ・コマンド・ウインドウの形式を図 2.17に示す。ユーザは本ウインドウの項目をマウスで選択することにより、辞書環境の初期設定または変更を行う。

   (a) 項目 "DO IT"
   辞書環境設定ウインドウで指定した値で、辞書環境を初期設定又は変更する。

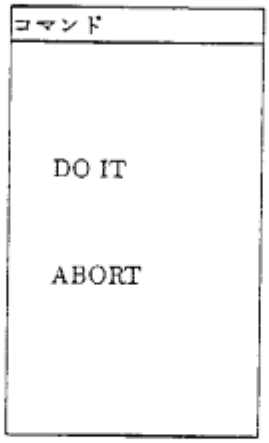


図 2.17: 辞書環境の初期設定時及び更新時のエディタ・コマンド・ウインドウ

(b) 項目 "ABORT"
辞書環境の初期設定又は変更を取り止める。

2. 中間形式辞書ファイルの更新時
中間形式辞書ファイルの更新時のエディタ・コマンド・ウインドウを図 2.18に示す。このウインドウは表示されるべきメニュー項目を一度に表示することができず、その一部が表示されており、ウインドウをスクロールさせることにより、隠されているメニュー項目を表示させることができる。ユーザは、項目をマウスで選択することにより、辞書エディタに指示を与える。詳細は、操作方法 (37ページを参照のこと) に記述してある。

(a) 項目 " インスペクタ "
辞書エディタを終了する。

(b) 項目 "TOP"
辞書エディタを終了する。項目 "TOP" は、LAX トップ・ウインドウから呼び出された場合のみ表示される。

(c) 項目 " 検索 (部分一致)" 及び項目 " 検索 (完全一致)"
エディタ・インタプリタ・ウインドウから検索条件を与えて検索する。 次の 2 つの下位項目から成る。

- 項目 " 新規 "
  中間形式辞書ファイルから条件に合致する形態素を取り出す。
- 項目 " 継続 "
  前回の検索の結果取り出された形態素の集まりから条件に合致する形態素を取り出す。

(d) 項目 " カテゴリ検索 "
カテゴリ又は表層を検索キーとして中間形式辞書ファイルを検索する。次の 2 つの下位項目から成る。

- 項目 " カテゴリ⇒ホルダ "
  ある特定のカテゴリを持つすべての形態素を取り出す。
- 項目 " 表層⇒ホルダ "
  ある特定のカテゴリと表層を持つすべての形態素を取り出す。

(e) 項目 " ホルダ選択 "
ホルダ選択は、ホルダ (37頁を参照のこと) に登録されている形態素をウインドウに表示し、取捨選択して絞り込み、結果をホルダに登録する。次の 2 つの下位項目から成る。

- 項目 " バッファ "

図 2.18: 中間形式辞書ファイルの更新時のエディタ・コマンド・ウインドウ

- 項目“メニュー”

(f) 項目“ホルダ・スタック”
ホルダのホルダ・スタックへの登録とホルダ・スタックからの取り出しを行う。次の2つの下位項目から成る。

- 項目“PUSH”
  ホルダをホルダ・スタックに登録する。
- 項目“POP”
  ホルダをホルダ・スタックから取り出す。

(g) 項目“クリア”
ホルダ及びエディタ・インタプリタ・ウインドウの初期設定を行う。次の2つの下位項目から成る。

- 項目“ALL”
  ホルダを空にし、エディタ・インタプリタ・ウインドウのキー部とデータ部をクリアし、見出しの表示だけにする。
- 項目“データ部”
  エディタ・インタプリタ・ウインドウのデータ部をクリアし、見出しの表示だけにする。エディタ・インタプリタ・ウインドウのキー部は変わらない。

(h) 項目“追加”
形態素を中間形式辞書ファイルに追加する。

(i) 項目“削除”
形態素を中間形式辞書ファイルから削除する。次の3つの下位項目から成る。

- 項目“バッファ”
- 項目“ホルダ”
- 項目“キー検索”

(j) 項目“素性置換”
形態素の左方接続素性、右方接続素性を変更する(カテゴリ、表層は変更しない)。次の3つの下位項目がある。

- 項目“バッファ”
- 項目“ホルダ”
- 項目“キー検索”

(k) 項目“カテゴリ置換”
形態素のカテゴリを変更する(表層、左方及び右方接続素性は変更しない)。次の3つの下位項目がある。

- 項目“バッファ”
- 項目“ホルダ”
- 項目“キー検索”

(l) 項目“初期値設定”
辞書環境項目の値を変更する。

(m) 項目“ソース変更”
カレント辞書を別の辞書に変更する。

(n) 項目“保存(ソース)”
逆トランスレートを起動して、中間形式辞書ファイルからソース形式辞書ファイルを作成する。

3. 接続素性書き換え時のエディタ・コマンド・ウインドウ
接続素性の書き換え(53頁を参照のこと)時のエディタ・コマンド・ウインドウの形式を図 2.19に示す。ユーザは本ウインドウの項目をマウスで選択することにより、接続素性の書き換えに関する指示を行う。

(a) 項目“DO IT”
エディタ・インタプリタ・ウインドウに表示されている形態素の接続素性を書き換える。

(b) 項目 "SKIP"
エディタ・インタプリタ・ウインドウに表示されている形態素の接続素性を書き換えは行わず、ホルダに登録されている次の形態素をエディタ・インタプリタ・ウインドウに表示する。

(c) 項目 "ABORT"
接続素性の書き換えを取り止める。

図 2.19: 接続素性書き換え時のエディタ・コマンド・ウインドウ

エディタ・インタプリタ・ウインドウ　エディタ・インタプリタ・ウインドウは、形態素の入力及び表示を行うウインドウである。通常 (辞書環境の初期設定、変更を行う場合以外)、本ウインドウが表示される。 図 2.20にエディタ・インタプリタ・ウインドウを示す。

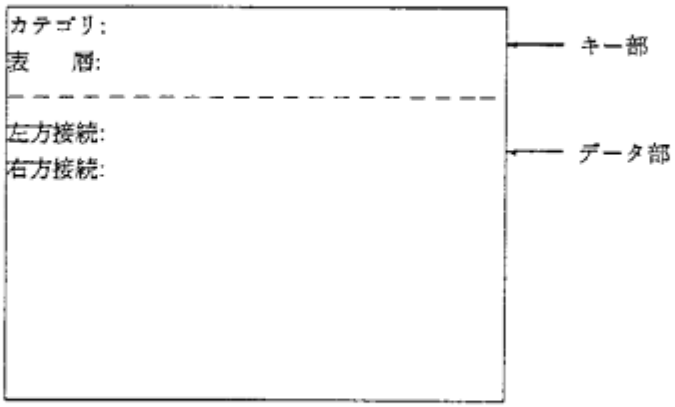


図 2.20: エディタ・インタプリタ・ウインドウ

エディタ・インタプリタ・ウインドウは、キー部とデータ部から成る。キー部、データ部とも、見出しと表示・入力域から成る。

1. キー部
   キー部は、カテゴリと表層の表示・入力域から成る。ユーザは、":" に続く部分に入力する。また、検索結果等の表示も同部分になされる。

2. データ部
   データ部は、左方接続実性と右方接続実性の表示・入力域から成る。ユーザは、":" に続く部分に入力する。また、検索結果等の表示も同部分になされる。

辞書環境設定ウインドウ　辞書環境設定ウインドウは、辞書の環境項目の初期設定、変更を行うウインドウである。本ウインドウは、LAX トップ・ウインドウの項目 "新規" が選択された時 (辞書環境の初期設定)、エディタ・コマンド・ウインドウの項目 "初期値設定" が選択された時 (辞書環境の変更) に表示される。辞書環境の初期設定、変更が終了するとエディタ・インタプリタ・ウインドウが表示される。 図 2.20に辞書環境設定ウインドウを示す。

辞書環境設定ウインドウは、見出しと表示・入力域から成る。見出しのうち、[ ]内に表示されているのは、初期設定時には省略値であり、変更時には現在値である。ユーザは、":" に続く部分に入力する。辞書環境項目には、以下のものがある。

1. ソース名
   カレント辞書の辞書名 (3.1.1を参照のこと) を文字列で入力する。本項目は、初期設定時にのみ表示され、更新時には表示されない。

2. マトリクス
   マトリクスをプロローグ・リストの形式で指定する。

3. 辞書クラスの数
   オブジェクト・プログラムを構成するクラス数を正の整数で指定する。

4. 辞書エントリの概数
   辞書に登録しようとする形態素の大まかな個数を正の整数で指定する。

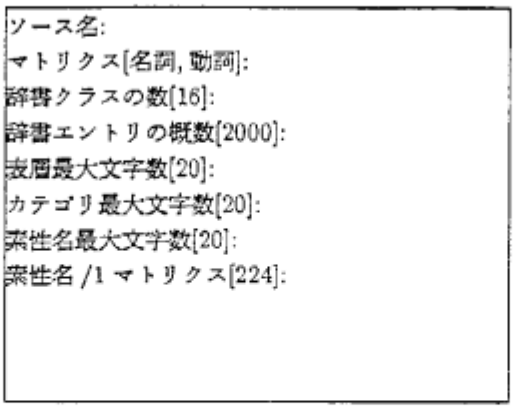


図 2.21: 辞書環境設定ウインドウ

5. 表層最大文字数
   辞書に登録しようとする形態素の表層の最大長 (文字数) と等しいかより大きい正の整数を指定する。

6. カテゴリ最大文字数
   辞書に登録しようとする形態素のカテゴリの最大長 (文字数) と等しいかより大きい正の整数を指定する。

7. 素性名最大文字数
   辞書に登録しようとする形態素の素性名の最大長 (文字数) と等しいかより大きい正の整数を指定する。

8. 素性数 /1 マトリクス
   1 マトリクスに属する素性の最大数と等しいかより大きい正の整数を指定する。

エディタ・メッセージ・ウインドウ エディタ・メッセージ・ウインドウは、辞書エディタの処理経過の表示、辞書エディタからユーザへの問い合わせの表示、それに対するユーザの応答の入力がなされるウインドウである。本ウインドウへの入力はキー・ボードから行う。

カテゴリ選択ウインドウ カテゴリ選択ウインドウは、カテゴリ名の一覧を表示・選択するスクロール機能付きのウインドウである。 ユーザは、カテゴリを 1 つマウスで選択する。カテゴリが選択されると本ウインドウは消える。マウス・マーカを本ウインドウから外すと本ウインドウは消え、カテゴリは選択されない。図 2.22にカテゴリ選択ウインドウを示す。

表層選択ウインドウ 表層選択ウインドウは、ある特定のカテゴリを持つ表層名の一覧を表示・選択するスクロール機能付きのウインドウである。本ウインドウの表層の表示形式は 2 種類ある。1 つは表層のみ表示する形式であり、他方は表層とトークン (3.1.1を参照のこと) を共に表示する形式である。ユーザは、LAX トップ・ウインドウの項目 “環境設定”(19頁を参照) で、どちらの表示形式を採用するか選択できる。表層とトークンが同じ場合は、表層のみ表示すればよい。 ユーザは、本ウインドウの表層を 1 つマウスで選択する。マウス・マーカを本ウインドウから外すと本ウインドウは消え、表層は選択されない。図 2.23に表層選択ウインドウを示す。本ウインドウのラベルに表示されているのはカテゴリ名である。

操作コマンド付き表層選択ウインドウ 操作コマンド付き表層選択ウインドウは、表層一覧とそれらに対する操作コマンドを表示・選択するスクロール機能付きのメニュー・ウインドウである。選択された表層を持つ形態素はホルダに登録される。本ウインドウの表層の表示形式は 2 種類ある。1 つは表層のみ表示する形式であり、他方は表層

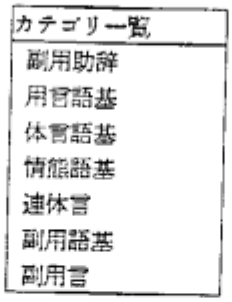
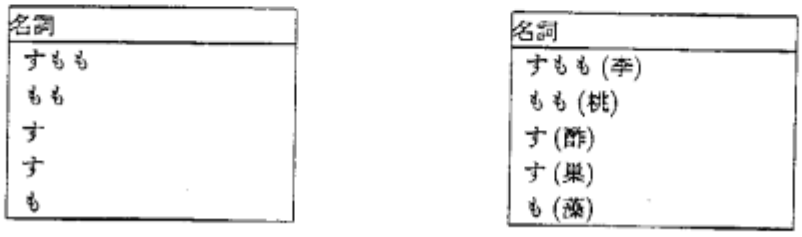


図 2.22: カテゴリ選択ウインドウ



(a) 表層のみ　　(b) 表層 (トークン)

図 2.23: 表層選択ウインドウ

とトークン (3.1.1を参照のこと) を共に表示する形式である。ユーザは、LAX トップ・ウインドウの項目 " 環境設定"(19頁を参照) で、どちらの表示形式を採用するか選択できる。表層とトークンが同じ場合は、表層のみ表示すればよい。図 2.24に操作コマンド付き表層選択ウインドウを示す。

1. 表層の選択
   本ウインドウでは複数の表層を選択できる。表層をマウスで選択すると、白抜き反転表示する。選択されない表層は元のままである。白抜き反転表示された表層をマウスで選択すると通常表示に変わる。

2. 操作コマンド
   操作コマンドの 1 つをマウスで選択する。

   (a) `Abort`
   表層の選択は無効になり本ウインドウは消える。

   (b) `All`
   本ウインドウは消え、ホルダの内容は変わらない。

   (c) `Select`
   白抜き反転表示の表層を持つ形態素がホルダに登録される。

   (d) `NoSelect`
   通常表示の表層を持つ形態素がホルダに登録される。

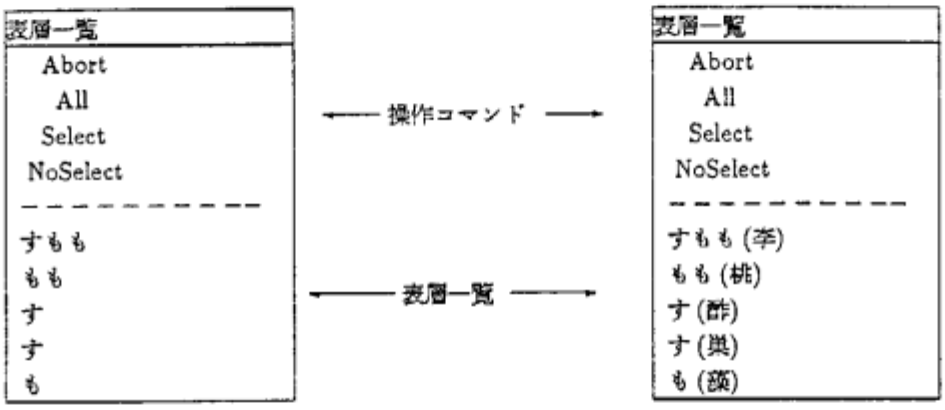


図 2.24: 操作コマンド付き表層選択ウインドウ

操作方法

起動方法の項で記述したように、辞書エディタの起動方法には 3 種類あるが、その後の操作方法にも若干違いがある。

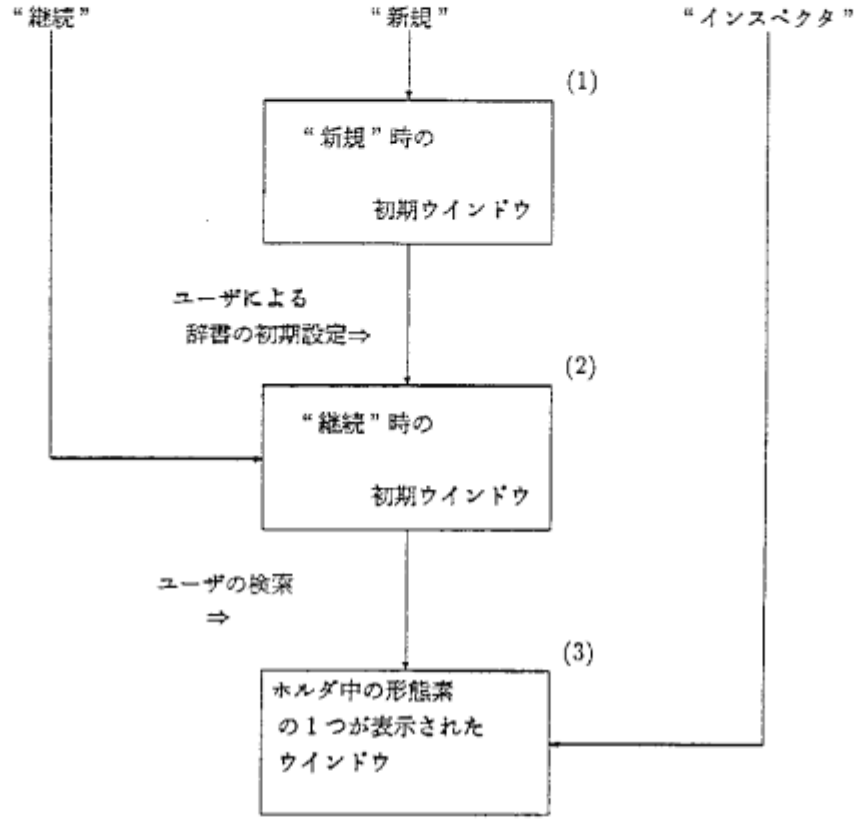


図 2.25: 起動方法による初期ウインドウの違い

図 2.25の (1) から (2) への操作方法は、起動方法の項 (22頁) に記述したのでそちらを参照されたい。以下には、図 2.25の (2) より下の操作を記述している。

形態素の検索　中間形式辞書ファイルを検索して条件に合致した形態素を取り出す方法について述べる。

1. 形態素の検索とホルダ、ホルダ・スタック

ユーザが中間形式辞書ファイルを更新する場合、中間形式辞書ファイル全体を対象とすることは稀で、ほとんどの場合、何らかの条件で絞り込まれた形態素を対象とする。この絞り込まれた形態素を登録しておく器がホルダである。ホルダは、辞書エディタに１つしか存在せず、常に最後になされた絞り込みの結果が登録されている。ユーザがホルダの内容を保存することができるのが、ホルダ・スタックである。ホルダをホルダ・スタックに登録することをプッシュ、ホルダ・スタックからホルダを取り出すことをポップという。プッシュは、ホルダをホルダ・スタックの末尾に追加し、ポップはホルダ・スタックから最も新しく登録されたホルダを取り出す。

ホルダに形態素を登録する操作には、以下のものがある。

- インスペクタの形態素接続情報表示ウインドウで、マウス中１回クリック又はマウス中２回クリックで形態素を選択する。

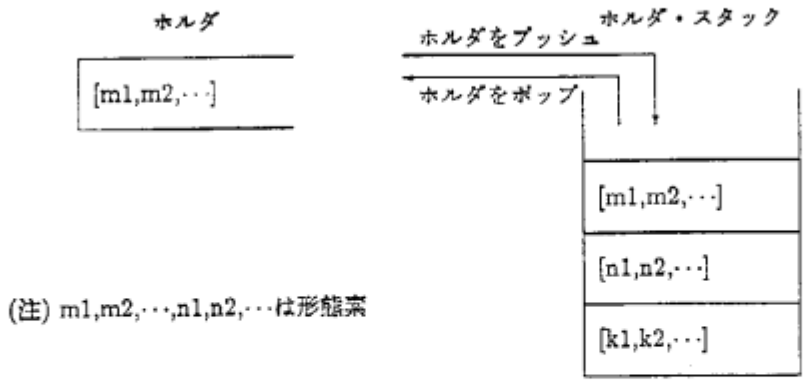


図 2.26: ホルダとホルダ・スタック

- エディタ・コマンド・ウインドウの以下の項目を選択する (79頁を参照のこと)。
  - 検索 (部分一致) の下位項目
  - 検索 (完全一致) の下位項目
  - カテゴリ検索の下位項目
  - ホルダ選択の下位項目
  - ホルダ・スタックの下位項目 "POP"

ホルダを空にするには、エディタ・コマンド・ウインドウの項目“クリア”の下位項目“ALL”をマウスで選択する。

ホルダをホルダ・スタックに登録するには、エディタ・コマンド・ウインドウの項目“ホルダ・スタック”の下位の項目“PUSH”をマウスで選択する。ホルダがホルダ・スタックに登録された場合は、エディタ・メッセージ・ウインドウに“ホルダをスタックに追加しました”というメッセージが表示される。ホルダが空の場合は、エディタ・メッセージ・ウインドウに、“追加するホルダがありません”というメッセージが表示され登録されない。

ホルダをホルダ・スタックから取り出すには、エディタ・コマンド・ウインドウの項目“ホルダ・スタック”の下位の項目“POP”をマウスで選択する。ホルダがホルダ・スタックから取り出された場合、取り出されたホルダに登録されている形態素のうちの1つがエディタ・インタプリタ・ウインドウに表示され、エディタ・メッセージ・ウインドウに、“ホルダを取り出しました”というメッセージが表示される。ホルダ・スタックが空の場合は、エディタ・メッセージ・ウインドウに、“スタックが空です”というメッセージが表示される。

2. 検索キーと検索条件の与え方

(a) 検索キーと検索条件
検索キーは次の4種類である。検索条件は、検索キーに検索値を与えることにより設定する。検索値を与えなかった検索キーは、検索条件として採用されない。これらの検索キーは全て論理積で結合される。

- カテゴリ
  1つのカテゴリを文字列で指定する。
- 表層
  1つの表層を文字列で指定する。
- 左方接続素性
  以下の形式で入力する。

<左方接続素性>::= "["<要素> {","<要素>}"]"
最初の[と最後の]は省略してもよい。(例: 派生([ウ系強変化ラ]))
<要素>::= <マトリクス>"([" <接続素性>{","<接続素性>}"])"
<マトリクス>::= 文字列
<接続素性>::= 文字列

- 右方接続素性
以下の形式で入力する。
<右方接続素性>::= "["<要素> {","<要素>}"]"
最初の[と最後の]は省略してもよい。
<要素>::= <マトリクス>"([" <接続素性>{","<接続素性>}"])"
<マトリクス>::= 文字列
<接続素性>::= 文字列

(b) 検索条件の与え方
検索条件の与え方は、次の4種類ある。

- エディタ・インタプリタ・ウインドウから与える
エディタ・インタプリタ・ウインドウのカテゴリ、表層、左方接続素性、右方接続素性の少なくとも1つに対し値をキー・ボードから設定する。ただし、既にエディタ・インタプリタ・ウインドウに表示されている値をそのまま採用することも、これらを一部変更して使うこともできる。
- カテゴリの一覧を表示したメニュー・ウインドウから選ぶ
5を参照のこと。
- 表層の一覧を表示したメニュー・ウインドウから選ぶ
5を参照のこと。
- ホルダに登録されている形態素を、1つずつ順次ウインドウに表示して取捨選択する
6を参照のこと。

3. エディタ・インタプリタ・ウインドウの操作
キー・ボードからカテゴリ、表層等に値を設定できる他に、エディタ・インタプリタ・ウインドウに対しては次のような操作が可能である。

(a) 形態素表示
エディタ・インタプリタ・ウインドウには、ホルダに登録されている形態素のうちの1つが表示されている。操作ウインドウの項目をマウスで選択することにより、次の操作ができる。現在、ホルダにn個の形態素が登録されていて、そのうちのi番目 (iはn以下) の形態素がエディタ・インタプリタ・ウインドウに表示されているとする。

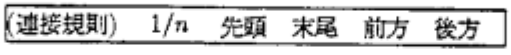


図 2.27: 操作ウインドウ

i. 項目"前方"
$i-1$番目の形態素が表示される。但し、1番目の形態素が表示されている場合は、n番目が表示される。

ii. 項目"後方"
$i+1$番目の形態素が表示される。但し、n番目の形態素が表示されている場合は、1番目が表示される。

iii. 項目"先頭"
1番目の形態素が表示される。

iv. 項目"末尾"
n番目の形態素が表示される。

(b) エディタ・インタプリタ・ウインドウのクリア
エディタ・コマンド・ウインドウの項目“クリア”の下位項目をマウスで選択することにより、エディタ・インタプリタ・ウインドウをクリアする(カテゴリ、表層等の見出しのみの表示にすること)ことができる。

i. 項目“ALL”
キー部、データ部を共にクリアする。更に、ホルダを空にする。

ii. 項目“データ部”
左方接続素性、右方接続素性を見出しのみの表示にする。

4. 検索条件をエディタ・インタプリタ・ウインドウから与える検索(図 2.28を参照のこと)

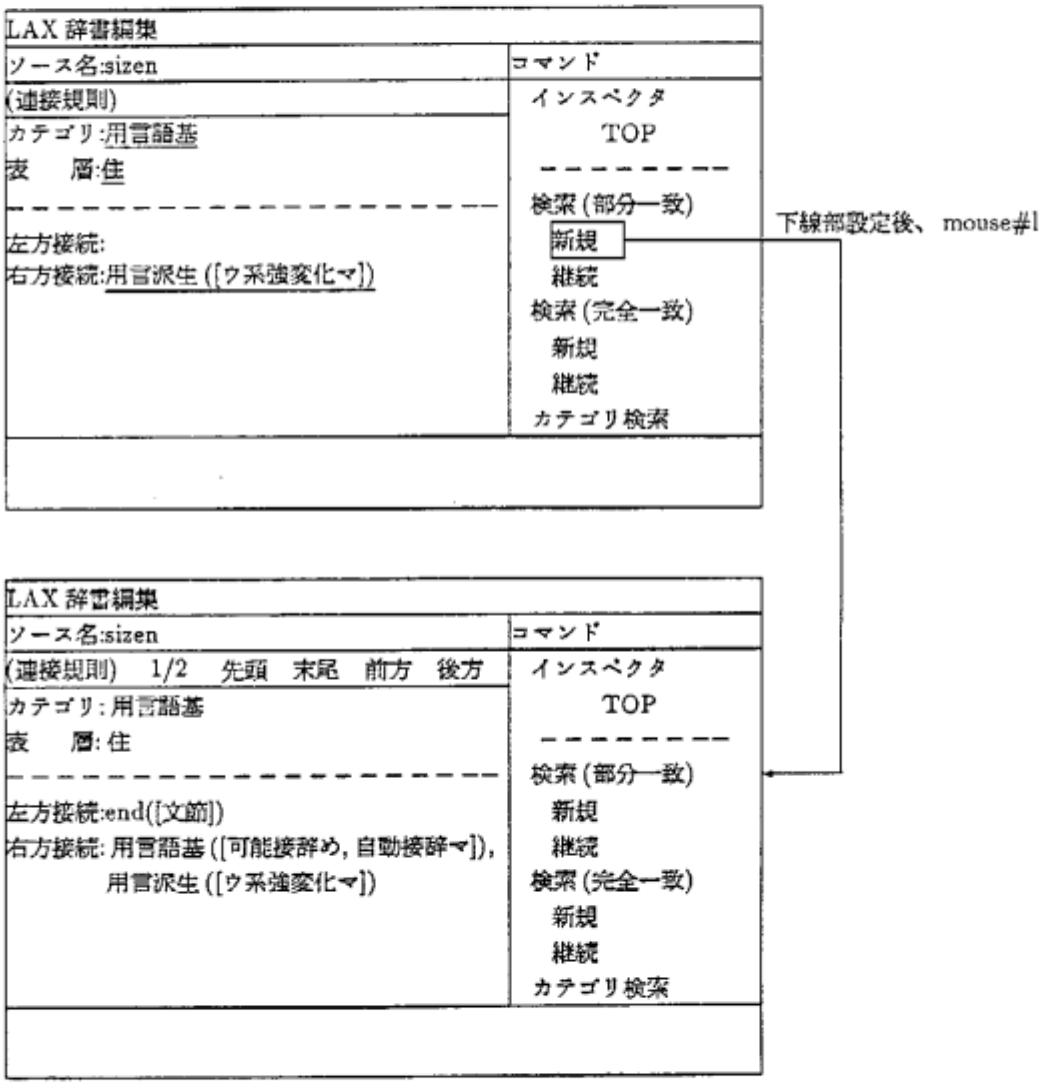


図 2.28: 項目“検索”による検索

(a) “部分一致”と“完全一致”
“部分一致”の場合、表層とカテゴリについては与えられた値が完全に一致し、かつ左方接続素性と右方

接続素性については、検索値として与えられた接続素性を含む形態素が取り出される。一方、“完全一致”の場合、表層、カテゴリ、左方接続素性、右方接続素性の全てについて、与えられた値と完全に一致する形態素が取り出される。“部分一致”、“完全一致”とも次項で述べる“新規”と“継続”がある。

(b) “新規”と“継続”
“新規”の場合、検索の対象となるのは中間形式辞書ファイルである。この場合、これ以前になされた検索の影響は全く受けない。一方、“継続”の場合、検索の対象となるのはホルダである。“継続”は、直前になされた検索の結果を更に別の検索条件で絞り込むのに使われる。なお、ホルダが空の場合は、“新規”と全く同じである。

(c) 検索の操作
エディタ・インタプリタ・ウインドウのカテゴリ、表層、左方接続素性、右方接続素性の少なくとも1つに値を与えた後、項目“新規”または“継続”をマウスで選択する。

(d) 検索結果の表示
検索の結果、n個取り出されたとすると、これらn個の形態素はホルダに登録され、そのうちの1つがエディタ・インタプリタ・ウインドウに表示される。別の形態素を表示したい場合は、操作ウインドウの項目をマウスで選択する(3を参照のこと)。検索条件に合致する形態素が1つもない場合は、エディタ・メッセージ・ウインドウに“指定されたエントリはありません”というメッセージが表示される。また,指定された検索条件が正しくない場合は、検索は行われず、エディタ・メッセージ・ウインドウにエラー・メッセージが表示される。

5. カテゴリ又は表層による検索

これは、カテゴリ又はカテゴリと表層両方を検索キーとして中間形式辞書ファイルを検索する方法である。検索結果は、ホルダに登録され、そのうち1つがエディタ・インタプリタ・ウインドウに表示される。ユーザはエディタ・コマンド・ウインドウの項目“カテゴリ検索”の下位項目“カテゴリ⇒ホルダ”又は“表層⇒ホルダ”をマウスで選択する。

(a) 項目“カテゴリ⇒ホルダ”(図 2.29を参照のこと)
ある特定のカテゴリを持つ全ての形態素を取り出す検索である。本項目をマウスで選択すると、カテゴリ選択ウインドウが表示される。このウインドウに表示されたカテゴリをマウスで選択する。マウス・マーカをこのウインドウから外すと、ウインドウは消え選択はされない。

(b) 項目“表層⇒ホルダ”(図 2.30を参照のこと)
ある特定のカテゴリと表層を持つ全ての形態素を取り出す検索である。本項目をマウスで選択すると、カテゴリ選択ウインドウが表示される。このウインドウに表示されたカテゴリをマウスで選択すると、表層選択ウインドウが表示される。このウインドウに表示された表層をマウスで選択する。マウス・マーカをこのウインドウから外すと、ウインドウは消え選択はされない。

6. ホルダに登録されている形態素の絞り込み

ホルダに登録されている形態素を1つずつエディタ・インタプリタ・ウインドウに表示し、取捨選択して絞り込み、結果をホルダに登録する方法である。2種類の方法がある。ユーザは、エディタ・コマンド・ウインドウの項目“ホルダ検索”の下位項目“バッファ”又は“メニュー”をマウスで選択する。

(a) 項目“バッファ”(図 2.31を参照のこと)

形態素を1つずつエディタ・インタプリタ・ウインドウに表示して取捨選択する。項目“バッファ”をマウスで選択すると、

i. ホルダに登録されている形態素の1つがエディタ・インタプリタ・ウインドウに表示され、エディタ・メッセージ・ウインドウに“バッファに表示された形態素についての指示を選択して下さい。選択する(y)、しない(n)、終了(e) ?”というメッセージが表示され入力待ちになる。ホルダに形態素が登録されていない場合は、エディタ・メッセージ・ウインドウに“ホルダは空です。処理を終了します”というメッセージが表示される。

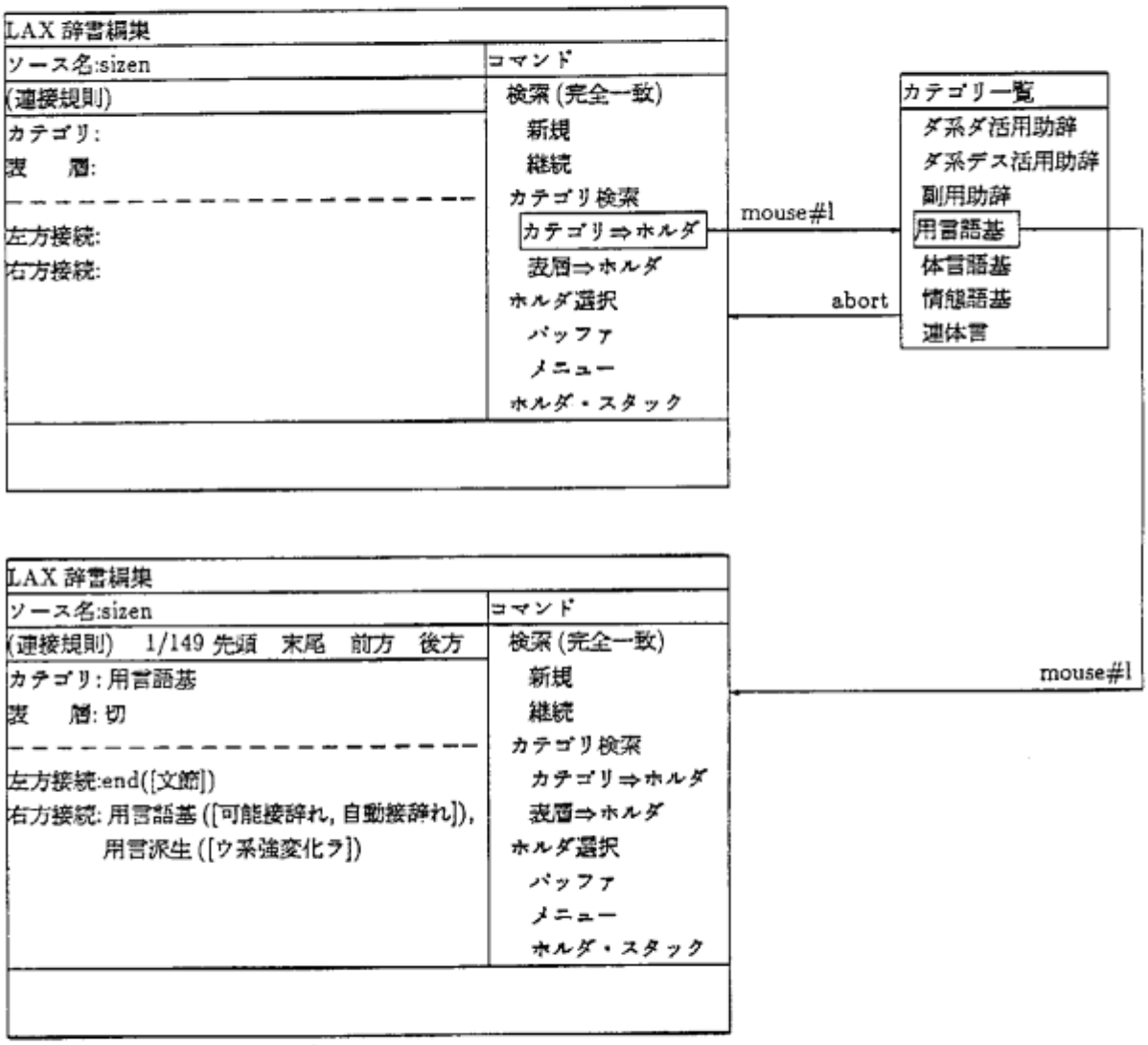


図 2.29: カテゴリによる検索

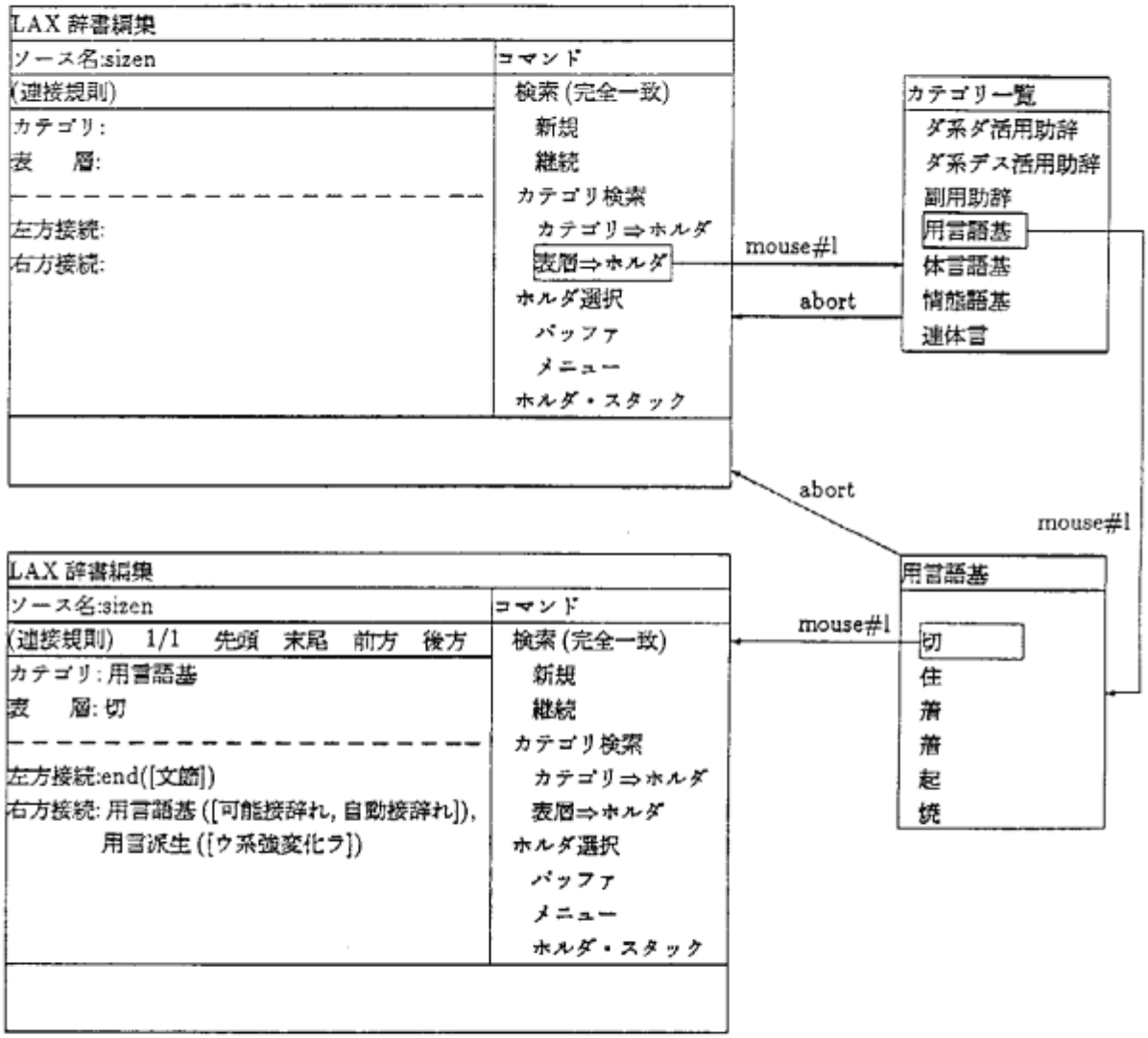


図 2.30: カテゴリと表層による検索

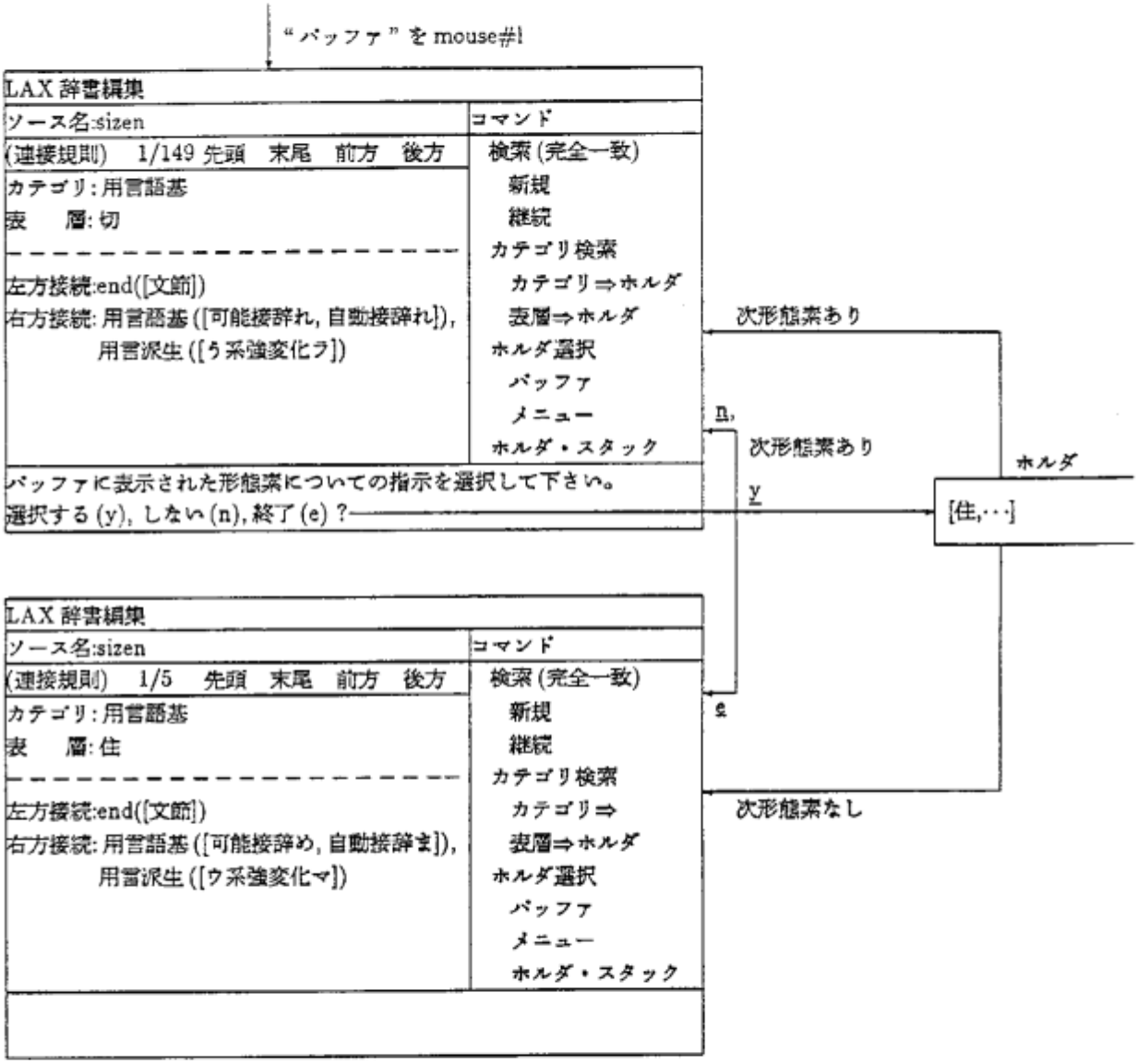


図 2.31: ホルダ選択 (項目“バッファ”)

A. "y" を入力すると、この形態素はホルダに登録される。

B. "n" を入力すると、この形態素はホルダに登録されない。

C. "e" を入力すると、エディタ・メッセージ・ウインドウに"処理を終了します"というメッセージが表示され、ホルダ選択は終了する。

ii. ホルダに次の形態素があれば、上記の 6(a)iが繰り返される。なければ、エディタ・メッセージ・ウインドウに"処理を終了します"というメッセージが表示され、ホルダ検索は終了する。

なお、ホルダ検索の結果、ホルダが空になった場合、エディタ・メッセージ・ウインドウに"指定されたエントリはありません"というメッセージが表示される。

(b) 項目"メニュー"(図 2.32を参照のこと)

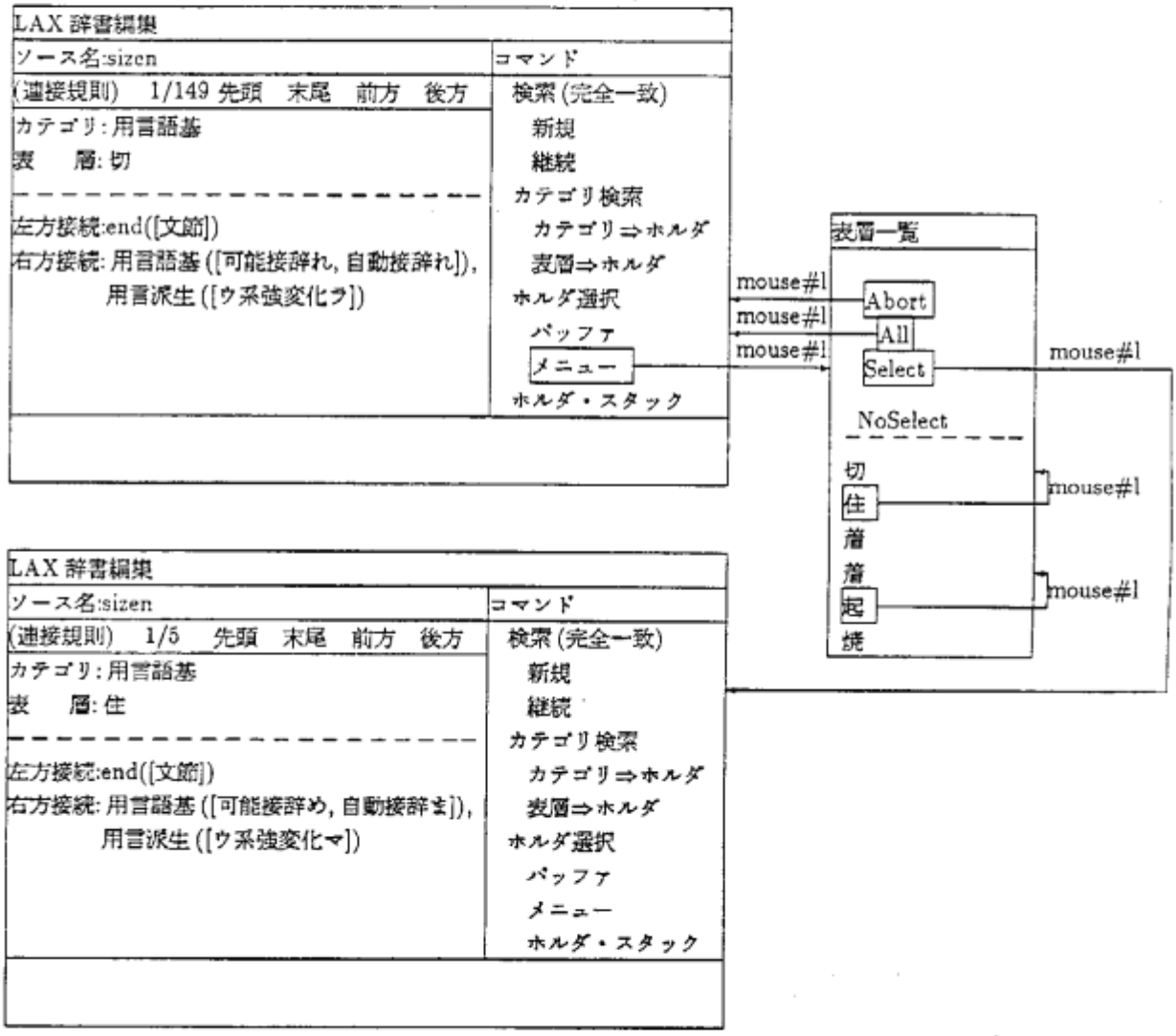


図 2.32: ホルダ選択(項目"メニュー")

ホルダに登録されている形態素の表層一覧を操作コマンド付き表層選択ウインドウに表示し取捨選択する。項目"メニュー"をマウスで選択すると、

i. 操作コマンド付き表層選択ウインドウが表示される。このウインドウの表層をマウスで選択する (複数選択可) と、白抜き表示される。この後、このウインドウの操作コマンドをマウスで選択する。

A. 項目 "Abort"
表層の選択が無効になり、エディタ・メッセージ・ウインドウに "中止します。処理を終了します" というメッセージが表示され、ホルダ選択は終了する。

B. 項目 "All"
ホルダの内容は変わらない。

C. 項目 "Select"
白抜き表示されている形態素がホルダに登録される (ホルダの内容が変わる)。

D. 項目 "NoSelect"
通常表示されている形態素がホルダに登録される (ホルダの内容が変わる)。

ii. ホルダに登録されている形態素の 1 つがエディタ・インタプリタ・ウインドウに表示される。

形態素の追加　形態素を中間形式辞書ファイルに追加する方法について述べる (図 2.33を参照のこと)。

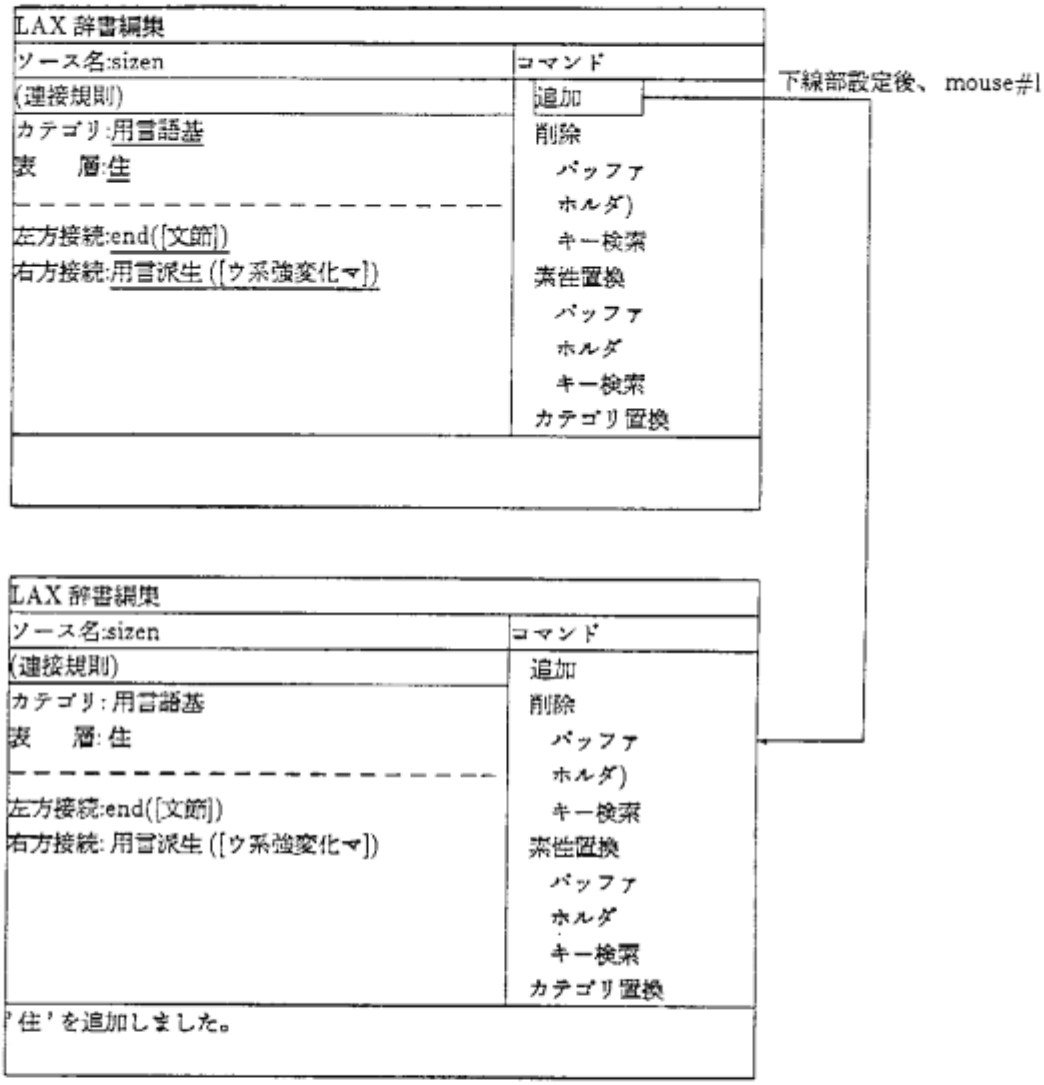


図 2.33: 形態素の追加

ユーザは、エディタ・インタプリタ・ウインドウに値を設定した後, エディタ・コマンド・ウインドウの項目 "追加" をマウスで選択する。エディタ・インタプリタ・ウインドウのカテゴリ、表層、左方接続素性、右方接続素性の全ての項目に対して値を設定しなければならない。形態素の追加が終了すると、エディタ・メッセージ・ウインドウに "xxx を追加しました" というメッセージが表示される。ここで、xxx は表層である。エディタ・インタプリタ・ウインドウに設定されている内容が次のような場合、エディタ・メッセージ・ウインドウにエラー・メッセージが表示され、形態素は追加されない。

- " カテゴリ、表層、左方 / 右方形態素のどれかに誤りがあります "
  値が設定されていない項目があるか、または (と)、[と]の対応が取れていない場合に表示される。
- "xxx は二重登録です "
  追加しようとした形態素と全く同じ形態素が既に中間形式辞書ファイルにある場合に表示される。
- " 中止します "

ユーザがエディタ・インタプリタ・ウインドウの見出しを変更した場合に表示される。

- “文法の誤りです”
  左方 / 右方接続素性の記述が正しくない場合に表示される。

形態素の削除　形態素を中間形式辞書ファイルから削除する方法について述べる。削除の操作方法は 3 種類ある。

- 現在、エディタ・インタプリタ・ウインドウに表示されている形態素を削除する
- ホルダに登録されている形態素の全体または一部を削除する
- 検索を行って、削除対象形態素を取り出し削除する

以下、これらの操作方法について説明する。

1. 項目“バッファ”(図 2.34を参照のこと)
   ホルダに登録されている形態素のうち、現在エディタ・インタプリタ・ウインドウに表示されている形態素を削除する。エディタ・インタプリタ・ウインドウの項目“削除”の下位項目“バッファ”をマウスで選択すると、形態素が削除され、エディタ・メッセージ・ウインドウに、“xxx を削除しました。処理を終了します”というメッセージが表示される。この形態素は、ホルダからも削除される。ホルダが空の場合は、エディタ・メッセージ・ウインドウに“指定されたエントリはありません。処理を終了します”というメッセージが表示され、形態素は削除されない。
2. 項目“ホルダ”(図 2.35を参照のこと)

   ホルダに登録されている形態素の全体または一部を削除する。エディタ・コマンド・ウインドウの項目“削除”の下位項目“ホルダ”をマウスで選択すると、

   (a) ホルダに 2 個以上の形態素がある場合
   エディタ・メッセージ・ウインドウに“削除方法を選択してください。一括 (a)、バッファ表示 (b)、メニュー選択 (m)、キャンセル (c) ?”というメッセージが表示され、入力待ちになる。ユーザは、次のどれかを入力する。

   i. a
   ホルダに登録されている全ての形態素を削除する。“a”を入力すると、エディタ・メッセージ・ウインドウに、“xxx を削除しました”という処理経過を示すメッセージ (xxx は表層) が表示され、削除が完了すると“処理を終了します”というメッセージが表示される。

   ii. b
   ホルダに登録されている形態素を 1 つずつエディタ・インタプリタ・ウインドウに表示して、その 1 つずつに対し、削除するかどうかを指定する方法である。“b”を入力すると、

   A. ホルダに登録されている形態素の 1 つがエディタ・インタプリタ・ウインドウに表示され、エディタ・メッセージ・ウインドウに“バッファに表示された形態素についての指示を選択して下さい。削除する (y)、削除しない (n)、中止する (c) ?”というメッセージが表示され、入力待ちになる。

   B. これに対し、
   - この形態素を削除する場合は、“y”を入力する。エディタ・メッセージ・ウインドウに、“xxx を削除しました”というメッセージ (xxx は表層) が表示される。
   - この形態素を削除しない場合は、“n”を入力する。
   - このホルダに登録されている形態素の削除を止める場合は、“c”を入力する。
   - “y”、“n”、“c”以外を入力した場合は、再度同じメッセージが表示され入力待ちになる。

   C. ホルダに次の形態素がある場合は、上記 (2(a)iiA, 2(a)iiB) が繰り返され、ない場合はエディタ・メッセージ・ウインドウに“処理を終了します”というメッセージが表示され削除は終了する。

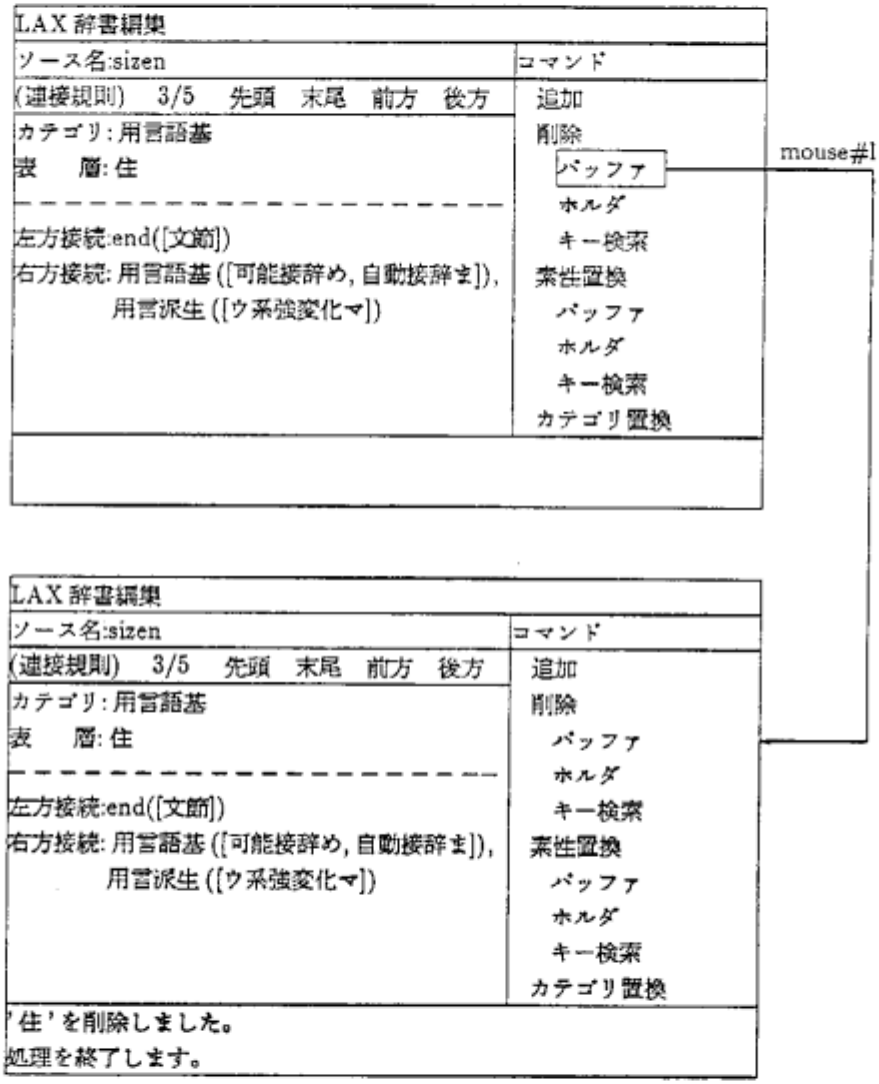


図 2.34: 形態素の削除 (項目 "バッファ")

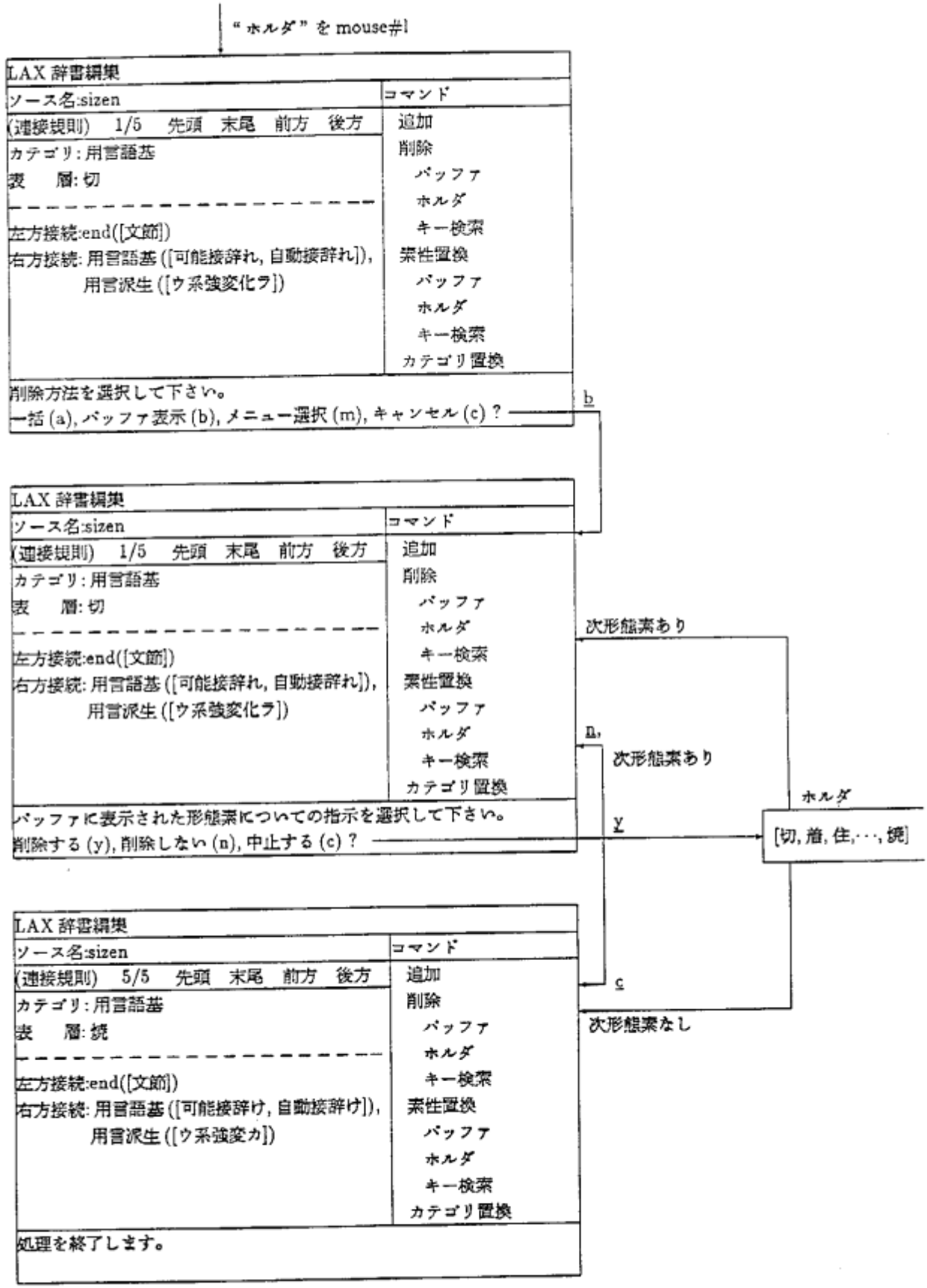


図 2.35: 形態素の削除(項目“ホルダ”)

iii. m

ホルダに登録されている形態素の表層一覧を操作コマンド付き表層選択ウインドウに表示し、この中から改めて削除対象形態素を選ぶ。“m”を入力すると、

A. 操作コマンド付き表層選択ウインドウが表示される。このウインドウの表層をマウスで選択する(複数選択可)と、白抜き表示される。この後、このウインドウの操作コマンドをマウスで選択する。

- 項目“Abort”
  表層の選択が無効になり、エディタ・メッセージ・ウインドウに“中止します。処理を終了します。”というメッセージが表示され、削除処理は終了する。
- 項目“All”
  ホルダの内容は変わらない。
- 項目“Select”
  白抜き表示されている形態素がホルダに登録される(ホルダの内容が変わる)。
- 項目“NoSelect”
  通常表示されている形態素がホルダに登録される(ホルダの内容が変わる)。

B. 以下の操作は、“b”を選択した場合と同じである。

iv. c

“c”を入力すると、エディタ・メッセージ・ウインドウに“中止します。処理を終了します。”というメッセージが表示され、削除が終了する。

v. “a”、“b”、“m”、“c”以外を入力した場合は、再度同じメッセージが表示され入力待ちになる。

(b) ホルダに含まれる形態素が1個だけの場合
ホルダに2個以上形態素がある場合に“b”を入力したときと同じである。

(c) ホルダが空の場合
エディタ・メッセージ・ウインドウに“中止します”というメッセージが表示され、削除は終了する。

3. 項目“キー検索”(図2.36を参照のこと)
カテゴリ、表層をキーにして検索を行い、結果をホルダに登録した後削除を行う。

(a) エディタ・インタプリタ・ウインドウのカテゴリ又は表層に値を設定し、エディタ・インタプリタ・ウインドウの項目“削除”の下位項目“キー検索”をマウスで選択する。

(b) 検索の結果、n個の形態素が取り出されると,エディタ・メッセージ・ウインドウに“対象エントリはn件あります。”というメッセージが表示され、形態素はホルダに登録される。
検索の結果、該当する形態素が1個もなかった場合は、エディタ・メッセージ・ウインドウに“指定されたエントリはありません。対象エントリはありません。処理を終了します”というメッセージが表示され削除は終了する。

(c) この後の操作は、項目“ホルダ”を選択した場合と同じである。

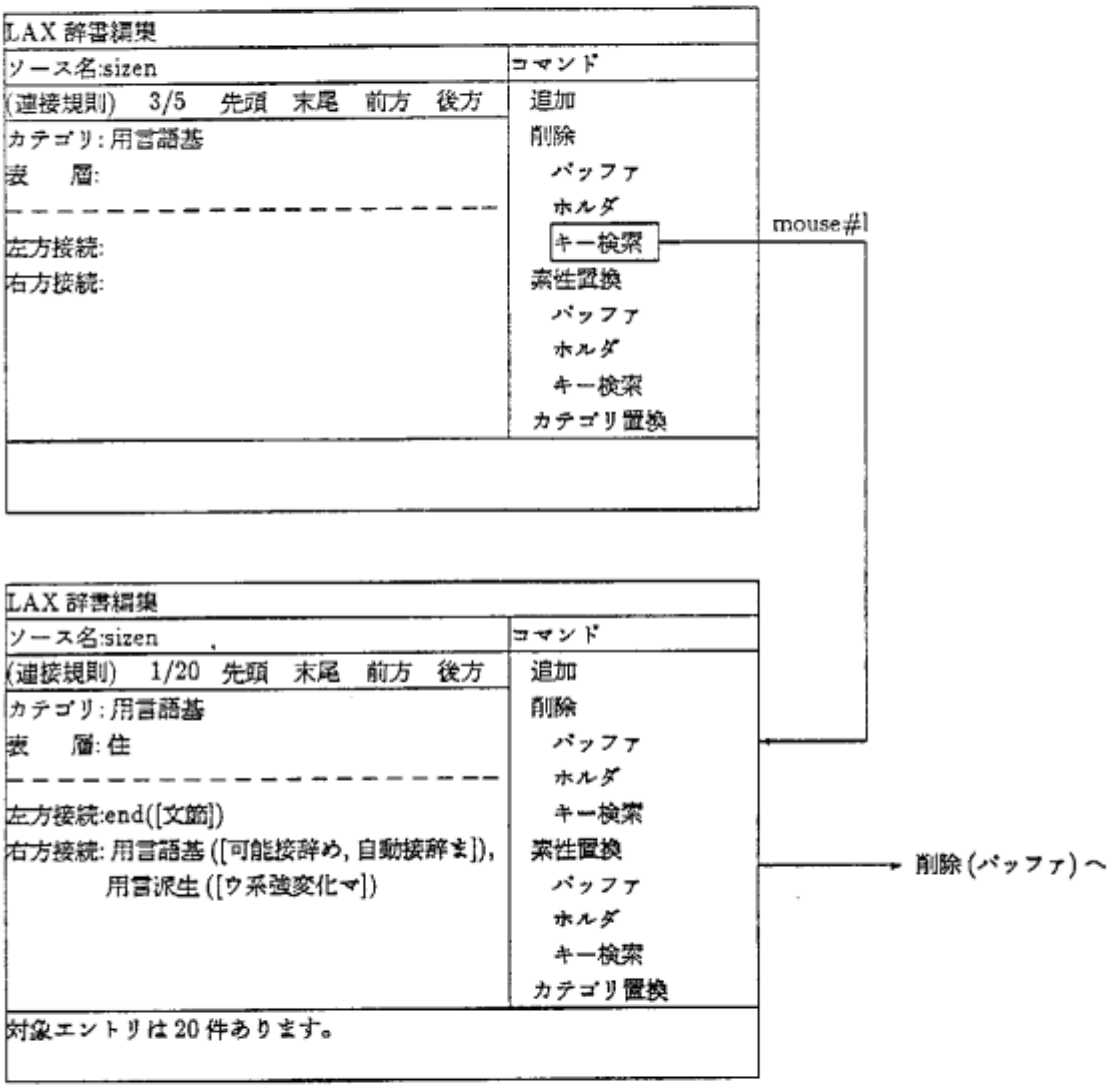


図 2.36: 形態素の削除 (項目 " キー検索 ")

形態素の書き換え　中間形式辞書ファイルに登録されている形態素の内容を書き換える方法について述べる。書き換えることができるのは、カテゴリ、左方接続素性、右方接続素性の3つで、表層を書き変えることはできない。

1. 接続素性の書き換え

形態素の左方接続素性、右方接続素性を変更する (カテゴリ、表層は変更しない) について述べる。操作方法は、次の3種類がある。

- 現在、エディタ・インタプリタ・ウインドウに表示されている形態素の左方または右方接続素性を変更する。
- ホルダに登録されている形態素の左方または右方接続素性を変更する。
- 検索を行って、変更対象形態素を取り出し、左方または右方接続素性を変更する。

以下、これらの操作方法について説明する。

(a) 項目 "バッファ"(図 2.37を参照のこと)

現在、エディタ・インタプリタ・ウインドウに表示されている形態素の左方または右方接続素性をキー・ボードで修正した後、エディタ・コマンド・ウインドウの項目 "素性置換" の下位項目 "バッファ" をマウスで選択する。カテゴリと表層は書き換えてはならない。

i. 形態素の変更がなされた場合は、エディタ・メッセージ・ウインドウに "xxx を書き換えました。処理を終了します" というメッセージが表示される。

ii. 変更の結果、中間形式辞書ファイルに既に登録されている形態素と全く同じになった場合は、エディタ・メッセージ・ウインドウに "xxx は二重登録です。処理を終了します" というメッセージが表示され、変更はなされない。

iii. カテゴリまたは表層が書き換えられた場合は、エディタ・メッセージ・ウインドウに "カテゴリ、表層、左方 / 右方接続素性のどれかに誤りがあります。処理を終了します" というメッセージが表示され、変更はなされない。

(b) 項目 "ホルダ"(図 2.38を参照のこと)
ホルダに登録されている形態素を順次エディタ・インタプリタ・ウインドウに表示し、その左方または右方接続素性を変更する。

i. エディタ・メッセージ・ウインドウの項目 "素性置換" の下位項目 "ホルダ" をマウスで選択すると、ホルダが空でなければ、次のメッセージが表示される。ホルダが空であれば、エディタ・メッセージ・ウインドウに "書き換え対象エントリを検索により指定して下さい。処理を終了します" というメッセージが表示され、素性置換は終了する。

A. ホルダに登録されている形態素の1つがエディタ・インタプリタ・ウインドウに表示され、

B. エディタ・メッセージ・ウインドウに "バッファに表示された形態素についての指示をマウスで選択して下さい" というメッセージが表示され、

C. エディタ・コマンド・ウインドウが図 2.39に示す表示に変わる。

ii. 必要なら、エディタ・インタプリタ・ウインドウの左方または右方接続素性をキー・ボードで修正する。カテゴリと表層は書き換えてはならない。

iii. 次にマウスで、

- 項目 "DO IT" を選択すると、この形態素の左方または右方接続素性は変更される。形態素に誤りがある場合は、エラー・メッセージがエディタ・メッセージ・ウインドウに表示され、変更されない。
- 項目 "SKIP" を選択すると、変更されない。
- 項目 "ABORT" を選択すると、素性置換は終了する。

iv. ホルダに次の形態素がある場合は、その形態素が表示されるので、上記を繰り返す。キューに次の形態素がない場合は、エディタ・メッセージ・ウインドウに "処理を終了します" というメッセージが表示され素性置換は終了する。

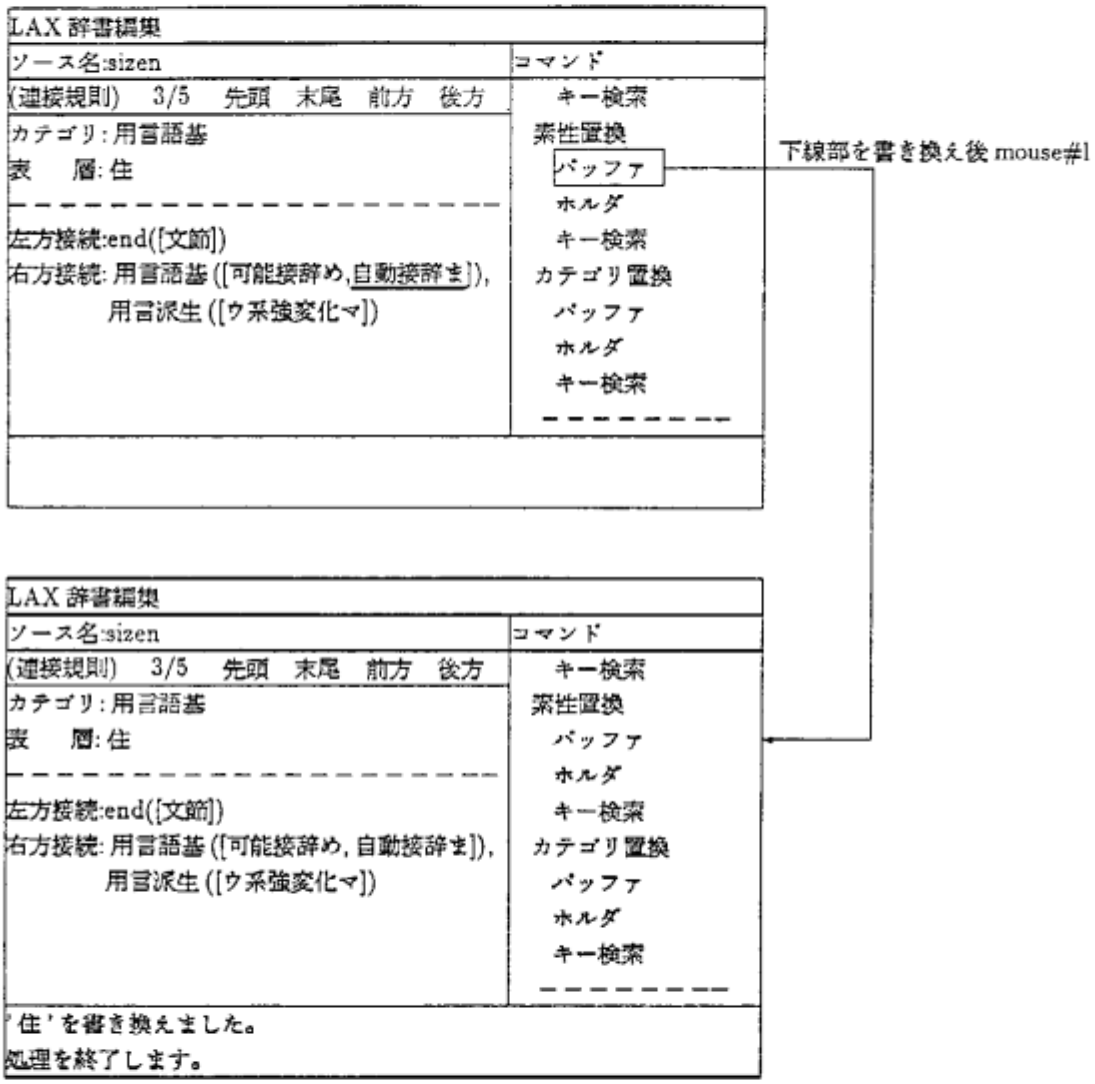


図 2.37: 左方または右方接続素性の置換 (項目“バッファ”)

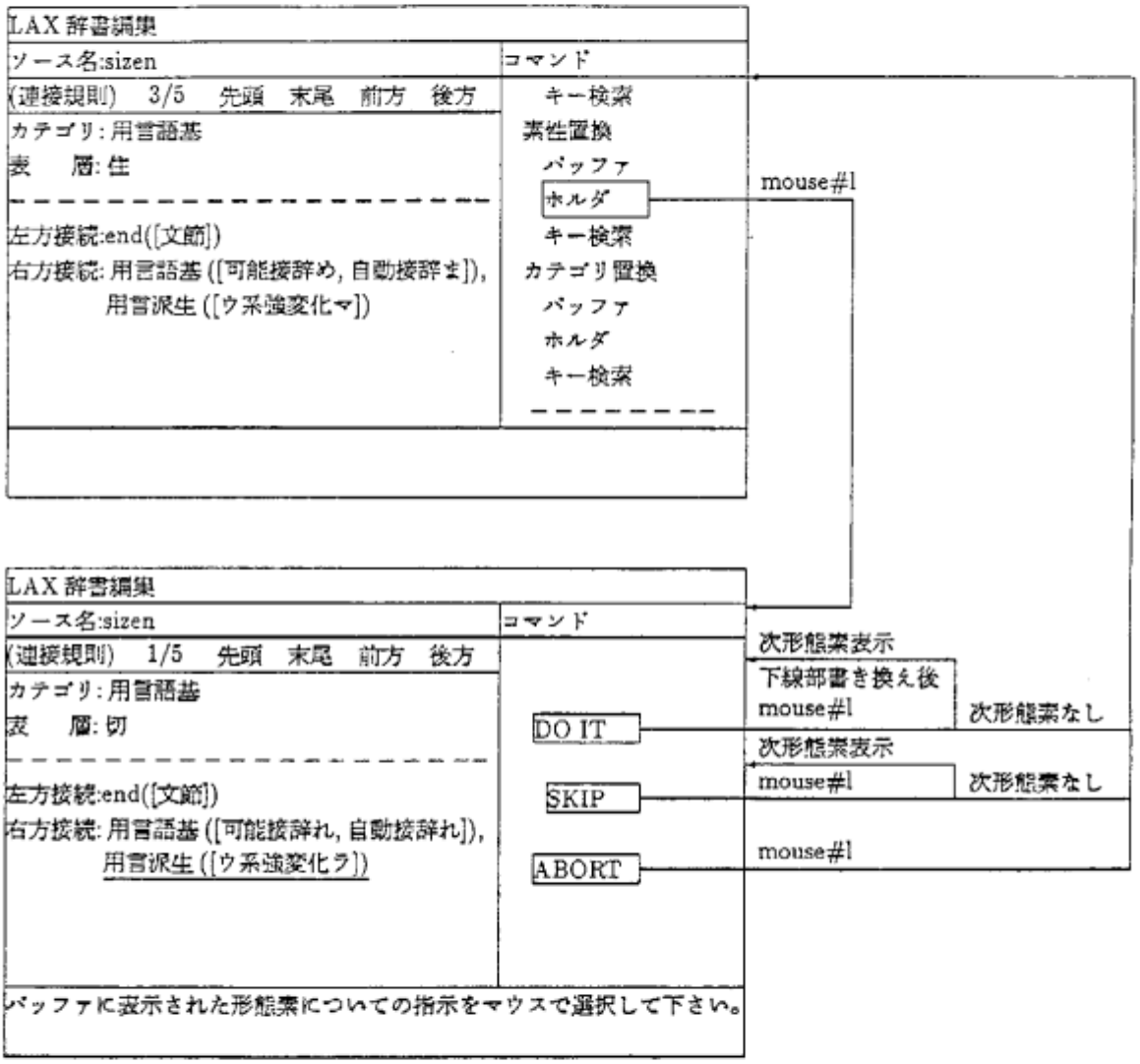


図 2.38: 左方または右方接続素性の置換 (項目“ホルダ”)

図 2.39: 接続素性変更時のエディタ・コマンド・ウインドウ

(c) 項目“キー検索”(図 2.40を参照のこと)
カテゴリ又は表層をキーにして検索を行って、左方または右方接続素性を修正したい形態素をホルダに登録した後、登録された形態素を順次エディタ・インタプリタ・ウインドウに表示して素性置換を行う。

i. エディタ・インタプリタ・ウインドウのカテゴリ又は表層に値を設定する。
ii. エディタ・コマンド・ウインドウの項目“素性置換”の下位項目“キー検索”をマウスで選択する。
iii. 形態素が取り出されれば、その 1 つがエディタ・インタプリタ・ウインドウに表示される。この後の操作は、項目“ホルダ”をマウスで選択した場合と同じである。
形態素がない場合は、エディタ・メッセージ・ウインドウに“指定されたエントリはありません。対象エントリはありません。処理を終了します”というメッセージが表示される。

2. カテゴリの書き換え

形態素のカテゴリを変更する (表層、左方及び右方接続素性は変更しない)。操作方法は、次の 3 種類がある。

- 現在、エディタ・インタプリタ・ウインドウに表示されている形態素のカテゴリを変更する
- ホルダに登録されている形態素のカテゴリを変更する
- 検索を行って、変更対象形態素を取り出しカテゴリを変更する

(a) 項目“バッファ”(図 2.41を参照のこと)

エディタ・コマンド・ウインドウの項目“カテゴリ置換”の下位項目“バッファ”をマウスで選択すると、エディタ・メッセージ・ウインドウに、“カテゴリを入力して下さい。:”というメッセージが表示され入力待ちになる。カテゴリを文字列で入力すると (xxxxxx<CR>)、

i. 形態素の変更がなされた場合は、エディタ・メッセージ・ウインドウに“xxx を書き換えました。処理を終了します”というメッセージ (xxx は表層) が表示される。
ii. 変更の結果、中間形式辞書ファイルに既に登録されている形態素と全く同じになった場合は、エディタ・メッセージ・ウインドウに“xxx は二重登録です。処理を終了します”というメッセージ (xxx は表層) が表示され、変更はなされない。

カテゴリの変更を取り止める場合は、M−g を押下する。“処理を終了します”というメッセージが表示され、変更はなされない。

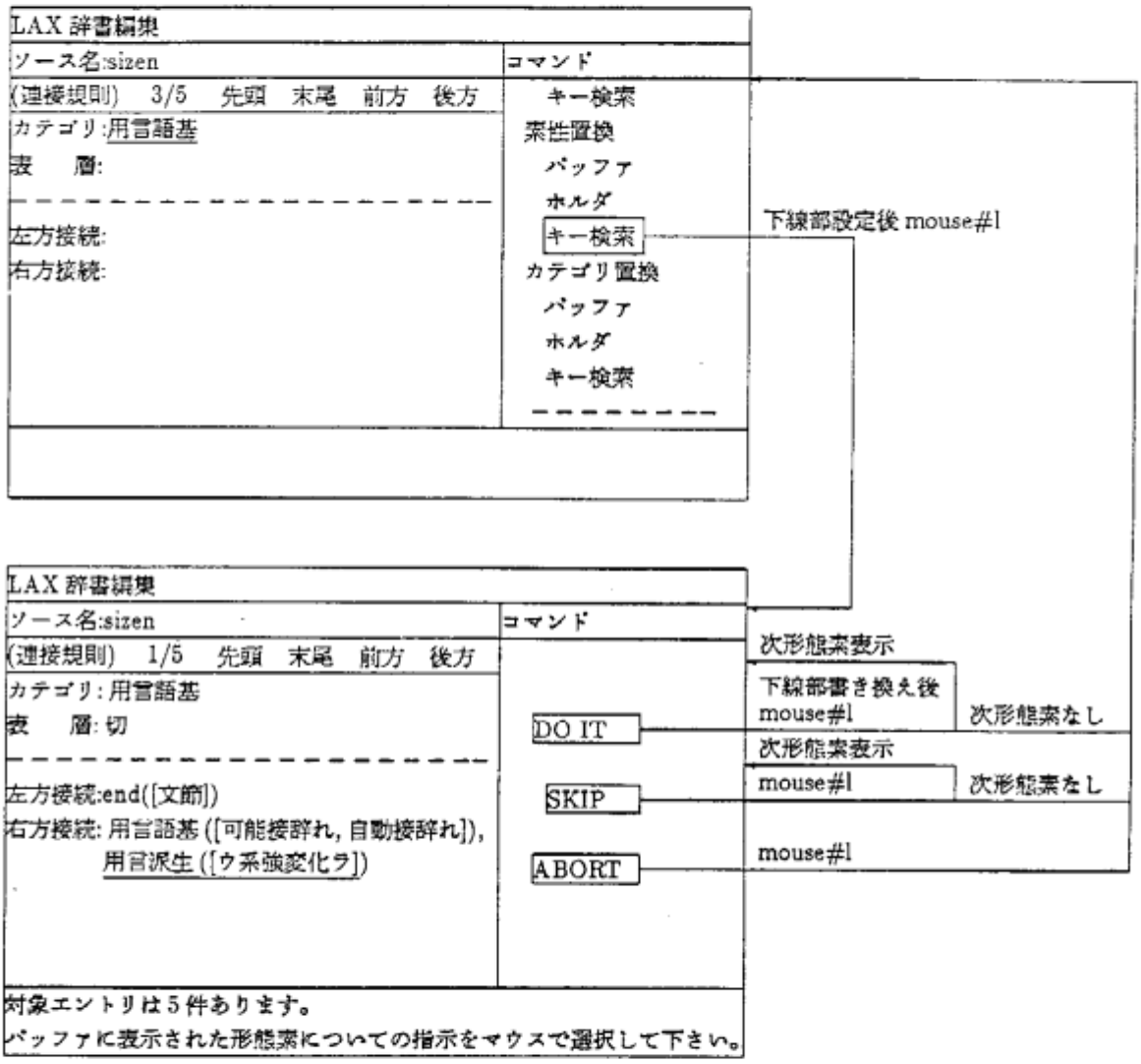


図 2.40: 左方または右方接続素性の置換 (項目 “ キー検索 ”)

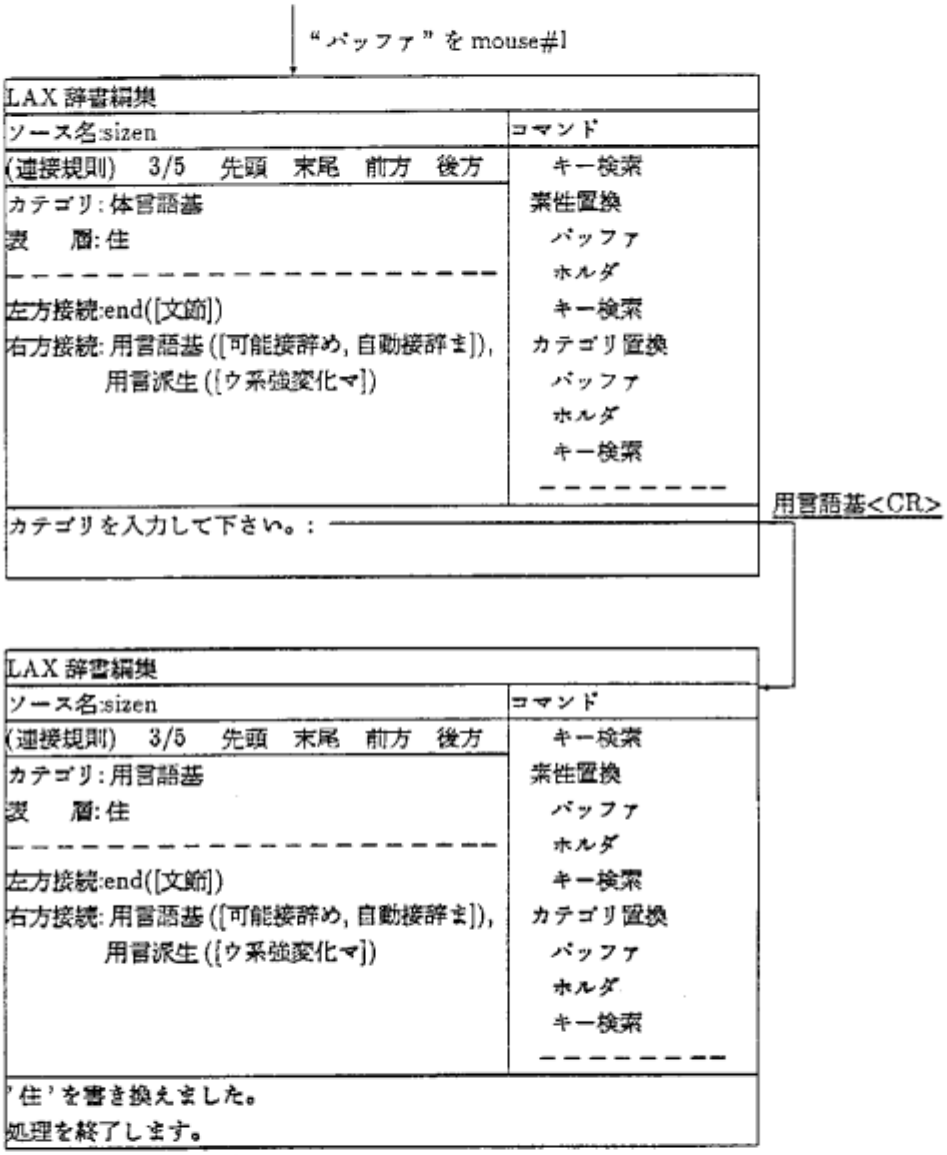


図 2.41: カテゴリの置換 (項目“バッファ”)

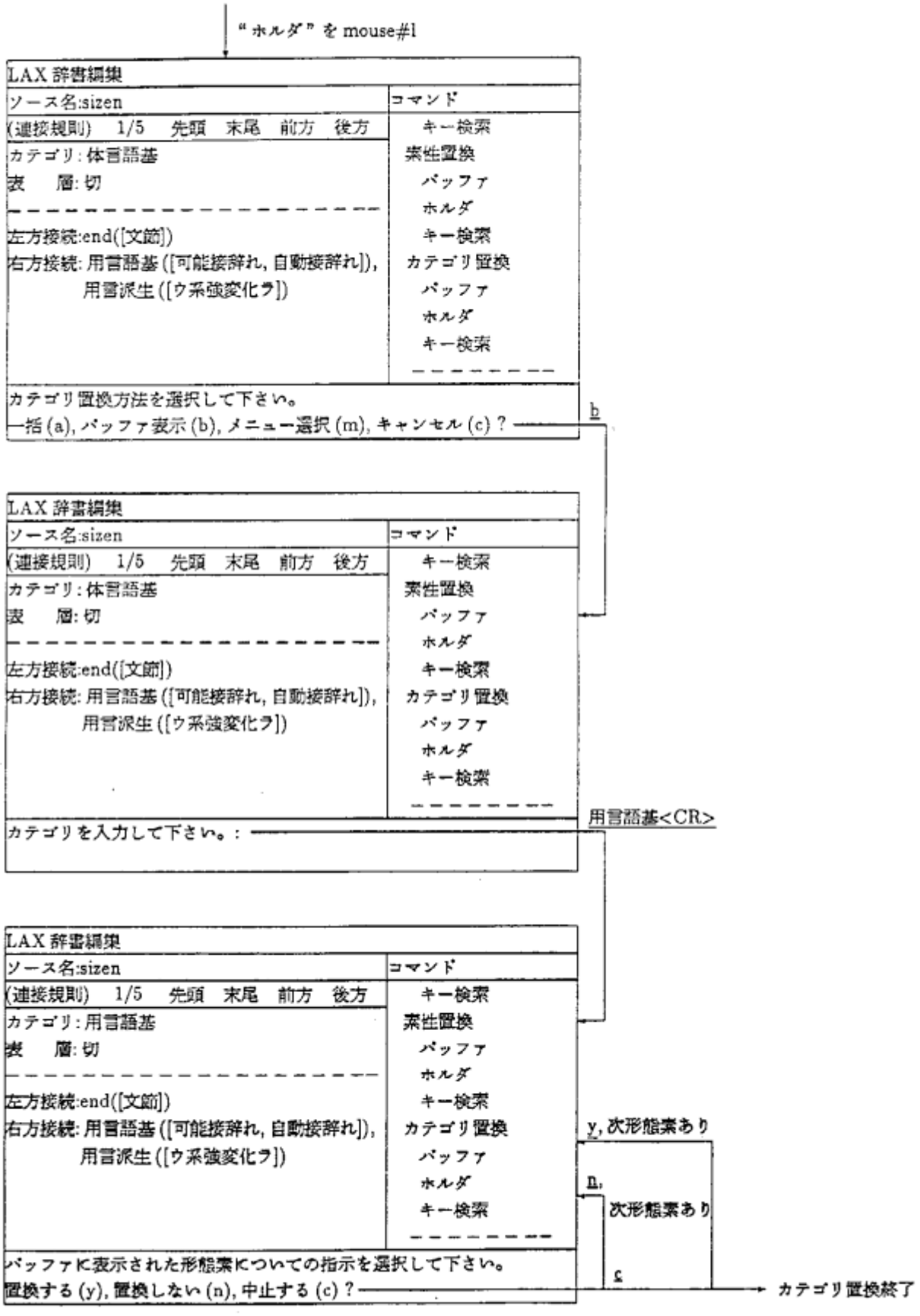


図 2.42: カテゴリの置換 (項目 "ホルダ")

(b) 項目“ホルダ”(図 2.42を参照のこと)

エディタ・コマンド・ウインドウの項目“カテゴリ置換”の下位項目“ホルダ”をマウスで選択すると、

i. ホルダに2個以上の形態素が登録されている場合は、エディタ・メッセージ・ウインドウに“カテゴリ置換の方法を選択して下さい。一括(a)、バッファ表示(b)、メニュー選択(m)、キャンセル(c)?”というメッセージが表示され入力待ちになる。

A. a

ホルダに登録されている形態素のカテゴリを、全て同じカテゴリに変更する方法である。“a”を入力すると、エディタ・メッセージ・ウインドウに、“カテゴリを入力して下さい。:”というメッセージが表示されるので、カテゴリを文字列で入力する。エディタ・メッセージ・ウインドウに変更の経過が表示され、ホルダに登録されている全ての形態素のカテゴリの変更が終了すると、“処理を終了します”というメッセージが表示されカテゴリ置換が終了する。カテゴリの書き換えによって、中間形式辞書ファイルに既に登録されている形態素と全く同じになる場合は、エディタ・メッセージ・ウインドウに“xxx は二重登録です”というメッセージ(xxx は表層)が表示され、この形態素については変更はなされない。

B. b

ホルダに登録されている形態素を1つずつ順次エディタ・インタプリタ・ウインドウに表示して、カテゴリの書き換えを行うかどうかを指示する方法である。“b”を入力すると、エディタ・メッセージ・ウインドウに“カテゴリを入力して下さい。:”というメッセージが表示されるので、カテゴリを文字列で入力する。エディタ・インタプリタ・ウインドウに、ホルダに登録されている形態素1つが表示され、エディタ・メッセージ・ウインドウに“表示された形態素についての指示を選択して下さい。置換する(y)、置換しない(n)、中止する(c)?”というメッセージが表示され入力待ちになる。

- y
  カテゴリを書き換える場合は“y”を入力する。ホルダに次の形態素がある場合は、エディタ・インタプリタ・ウインドウにそれが表示され、再びエディタ・メッセージ・ウインドウで入力待ちになる。ない場合は、エディタ・インタプリタ・ウインドウに“処理を終了します”というメッセージが表示されカテゴリ置換は終了する。中間形式辞書ファイルに既に登録されている形態素と全く同じものになった場合は、エディタ・メッセージ・ウインドウに“xxx は二重登録です”というメッセージ(xxx は表層)が表示され変更は行われない。
- n
  カテゴリを書き換えない場合は“n”を入力する。ホルダに次の形態素がある場合は、エディタ・インタプリタ・ウインドウにそれが表示され、再びエディタ・メッセージ・ウインドウで入力待ちになる。ない場合は、エディタ・インタプリタ・ウインドウに“処理を終了します”というメッセージが表示されカテゴリ置換は終了する。
- c
  カテゴリ置換を終了する場合は“c”を入力する。エディタ・メッセージ・ウインドウに“処理を終了します”というメッセージが表示される。
- その他の入力
  “y”、“n”、“c”以外の文字列が入力された場合は、再度同じメッセージが表示され入力待ちになる。

C. m

ホルダに登録されている形態素の表層一覧を操作コマンド付き表層選択ウインドウに表示し、この中から改めて置換対象形態素を選ぶ。“m”を入力すると、

- 操作コマンド付き表層選択ウインドウが表示される。このウインドウの表層をマウスで選択する(複数選択可)と、白抜き表示される。この後、このウインドウの操作命令をマウスで選択する。
  - 項目“Abort”
    表層の選択が無効になり、エディタ・メッセージ・ウインドウに“中止します。処理を終了します。”というメッセージが表示され、カテゴリ置換は終了する。

- 項目 “All”
  ホルダの内容は変わらない。
- 項目 “Select”
  白抜き表示されている形態素がホルダに登録される (ホルダの内容が変わる)。
- 項目 “NoSelect”
  通常表示されている形態素がホルダに登録される (ホルダの内容が変わる)。

• 以下の操作は、 “b” を入力した場合と同じである。

D. c
“c” を入力すると、エディタ・メッセージ・ウインドウに “中止します。処理を終了します” というメッセージが表示され、カテゴリ置換が終了する。

E. “a”、 “b”、 “m”、 “c” 以外を入力した場合は、再度同じメッセージが表示され入力待ちになる。

ii. ホルダに登録されている形態素が1個の場合は、項目 “バッファ” をマウスで選択した場合と同じである。

iii. ホルダが空の場合、エディタ・メッセージ・ウインドウに “処理を終了します” というメッセージが表示され、変更はなされない。

(c) 項目 “キー検索”(図 2.43を参照のこと)
カテゴリ、表層をキーにして検索を行って結果をホルダに登録した後カテゴリ置換を行う。

i. エディタ・インタプリタ・ウインドウのカテゴリまたは表層にキー・ボードで値を設定し、エディタ・コマンド・ウインドウの項目 “カテゴリ置換” の下位項目 “キー検索” をマウスで選択する。

ii. 検索の結果、n個の形態素が取り出されると, エディタ・メッセージ・ウインドウに “対象エントリはn件あります。” というメッセージが表示され、形態素はホルダに登録される。
検索の結果、該当する形態素が1個もなかった場合は、エディタ・メッセージ・ウインドウに “指定されたエントリはありません。対象エントリはありません。処理を終了します” というメッセージが表示されカテゴリ置換は終了する。

iii. この後の操作は、項目 “ホルダ” を選択した場合と同じである。

辞書環境の変更 以下のような辞書環境項目を変更する方法について述べる。

- マトリクス
- 辞書クラスの数
  オブジェクト・プログラムを構成するクラス数。
- 辞書エントリの概数
  辞書に登録しようとする形態素の大まかな個数。
- 表層最大文字数
  辞書に登録しようとする形態素の表層の最大長 (文字数)。
- カテゴリ最大文字数
  辞書に登録しようとする形態素のカテゴリの最大長 (文字数)。
- 素性名最大文字数
  辞書に登録しようとする形態素の素性名の最大長 (文字数)。
- 素性数/1マトリクス
  1マトリックスに属する素性の最大数。

LAXトップ・ウインドウで項目 “新規” を選択した場合の初期設定と異なり、ソース名の変更ができない。

辞書環境を変更すると、自動的に逆トランスレータ及びトランスレータが呼び出され、辞書は再生成される。辞書の再生成には次の2つの方法があり、ユーザはどちらかを選択できる。

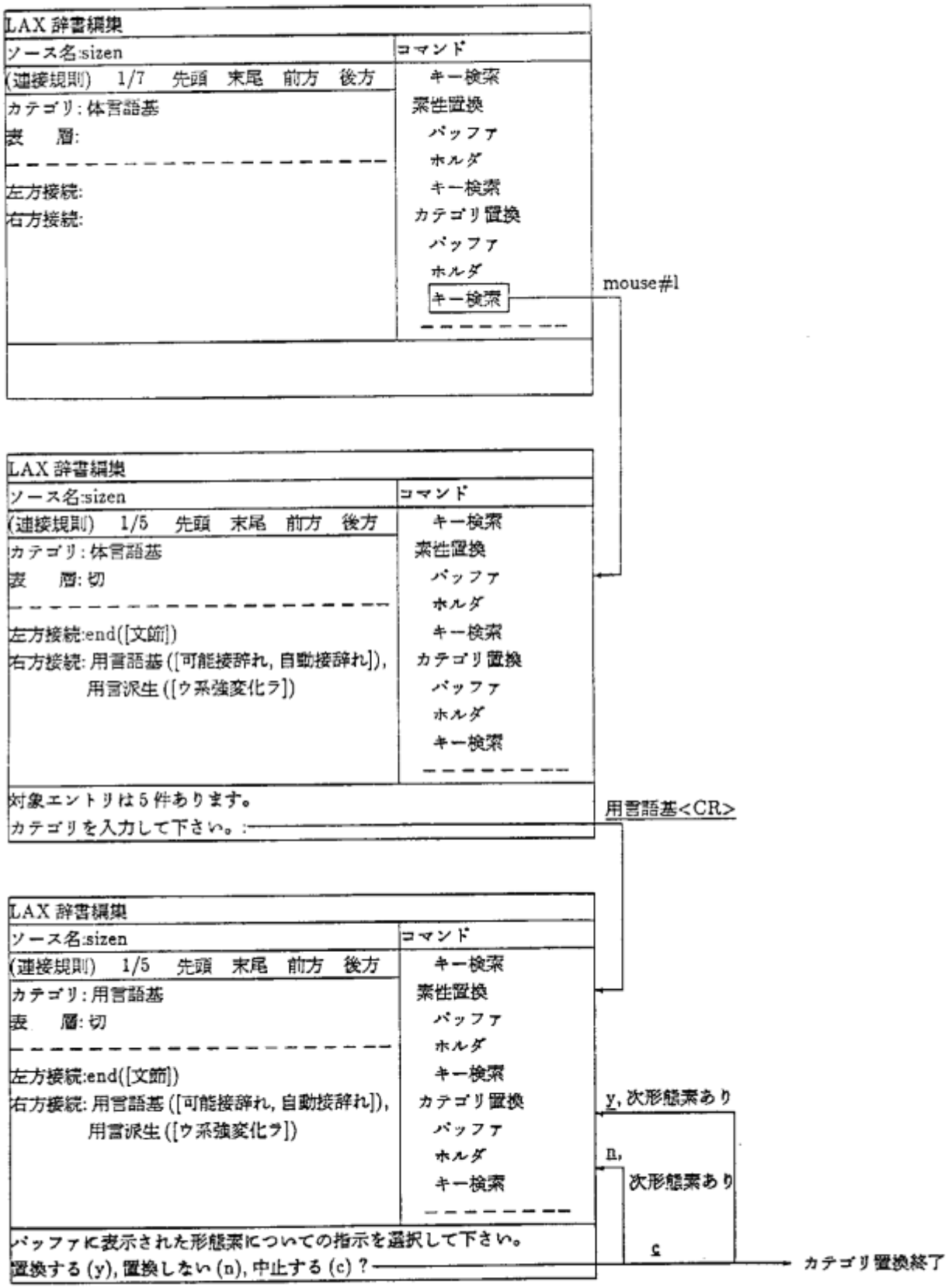


図 2.43: カテゴリの置換(項目“キー検索”)

- ソース形式辞書ファイルと中間形式辞書ファイルを共に再生成する
- 中間形式辞書ファイルのみ再生成する

エディタ・コマンド・ウインドウの項目“初期値設定”をマウスで選択すると、図 2.44が表示される。

1. 辞書環境設定ウインドウの辞書環境項目に値を設定する。辞書環境設定ウインドウには、項目名が見出しとして表示され、その後ろの[ ]内に、現在の値が表示されている。値を変更したい場合は、:の後に値を設定する。

   (a) マトリクス
   マトリクスをプロローグ・リストの形式で指定する。

   (b) 辞書クラスの数
   オブジェクト・プログラムを構成するクラス数を正の整数で指定する。

   (c) 辞書エントリの概数
   辞書に登録しようとする形態素の大まかな個数を正の整数で指定する。

   (d) 表層最大文字数
   辞書に登録しようとする形態素の表層の最大長(文字数)と等しいかより大きい正の整数を指定する。

   (e) カテゴリ最大文字数
   辞書に登録しようとする形態素のカテゴリの最大長(文字数)と等しいかより大きい正の整数を指定する。

   (f) 素性名最大文字数
   辞書に登録しようとする形態素の素性名の最大長(文字数)と等しいかより大きい正の整数を指定する。

   (g) 素性数/1マトリクス
   1マトリックスに属する素性の最大数と等しいかより大きい正の整数を指定する。

2. DO IT又はABORTをマウスで選択する。

   (a) DO IT
   辞書環境項目に変更値を与えた場合は、エディタ・メッセージ・ウインドウに変更項目名と値が表示された後、“ファイルを再作成します。ソース・ファイルを作成しますか? [y/n]:”というメッセージが表示され、入力待ちになる。ここで、

   i. ソース形式辞書ファイルと中間形式辞書ファイルを共に再生成する場合、“y”を入力する
      - エディタ・メッセージ・ウインドウに“辞書逆変換を開始しました。出力パス名:xxxxx”と表示される。xxxxxは、ソース形式辞書ファイルのパス名であり、ユーザはパス名を変更する必要があれば変更後、このままでよければ単に<CR>を押下する。
      - エディタ・メッセージ・ウインドウに、逆トランスレータ、トランスレータの処理経過が表示され、最後に“変更しました”というメッセージが表示され辞書環境の変更が終了する。
      - 辞書環境の変更を取り止める場合は、M−gを押下する。

   ii. 中間形式辞書ファイルのみを再生成する場合、“y”以外の文字を入力する
      - エディタ・メッセージ・ウインドウに、逆トランスレータ、トランスレータの処理経過が表示され、最後に“変更しました”というメッセージが表示され辞書環境の変更が終了する。

   辞書環境項目に変更値を与えなかった場合は、エディタ・メッセージ・ウインドウに“変更ありません”というメッセージが表示され、辞書環境は変更されない。辞書環境設定ウインドウはエディタ・インタプリタ・ウインドウに変わる。

   (b) ABORT
   エディタ・メッセージ・ウインドウに“変更を中止します”というメッセージが表示され、辞書環境は変更されない。辞書環境設定ウインドウはエディタ・インタプリタ・ウインドウに変わる。

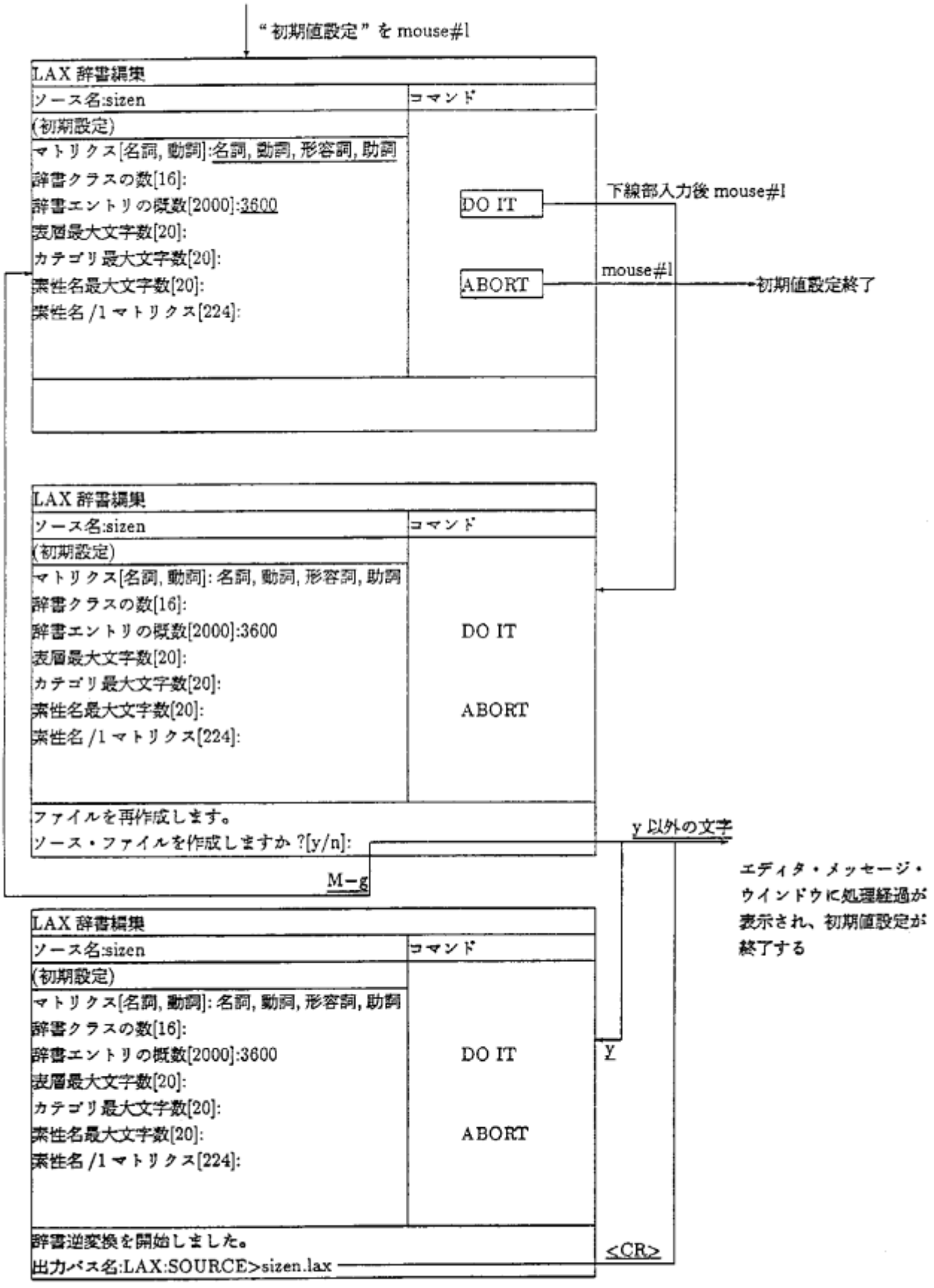


図 2.44: 辞書環境の変更

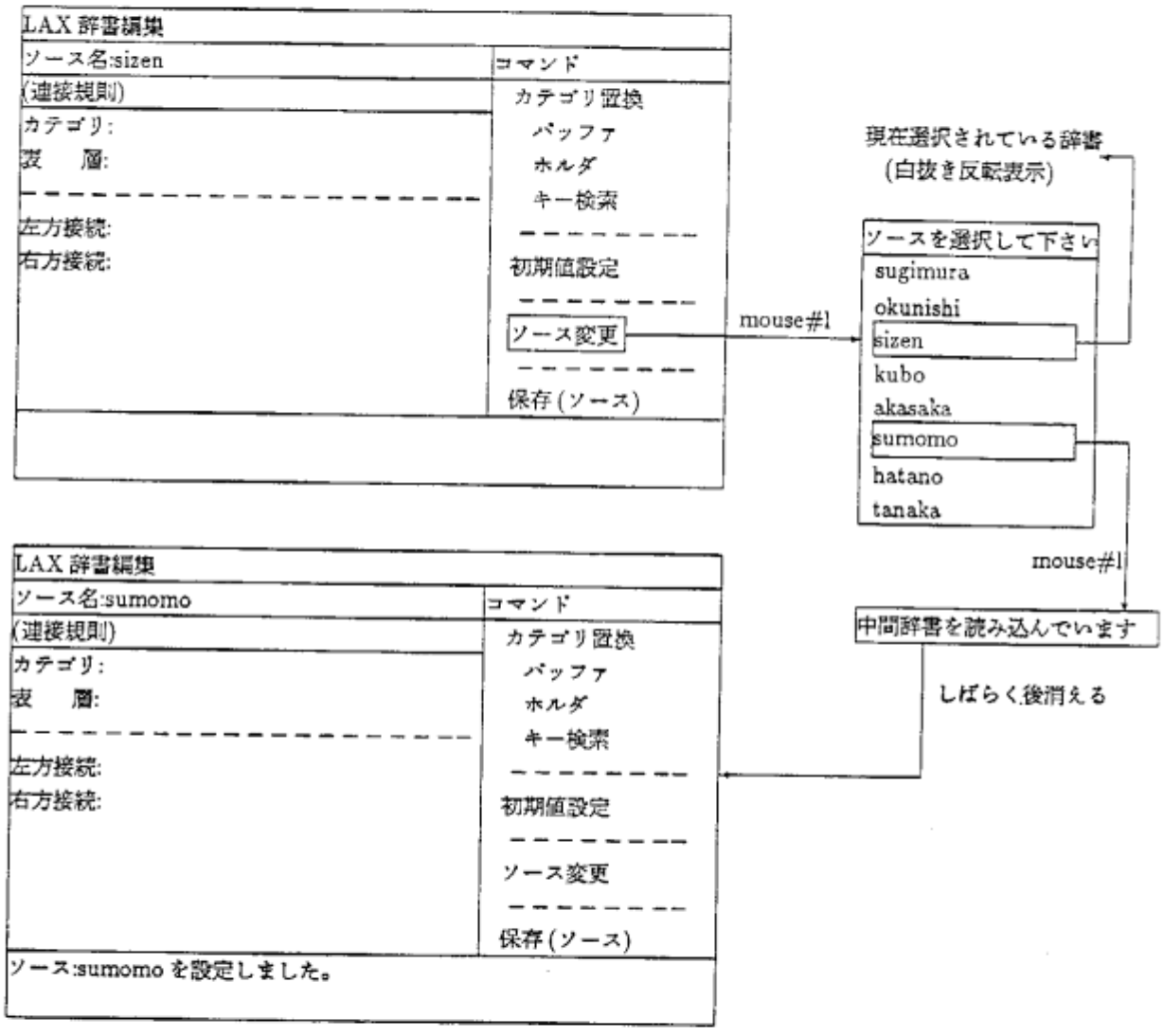

LAX 辞書編集
ソース名:sizen
コマンド
(連接規則)
カテゴリ:
表　層:
左方接続:
右方接続:
カテゴリ置換
バッファ
ホルダ
キー検索
初期値設定
ソース変更
保存(ソース)
mouse#1
現在選択されている辞書
(白抜き反転表示)
ソースを選択して下さい
sugimura
okunishi
sizen
kubo
akasaka
sumomo
hatano
tanaka
mouse#1
中間辞書を読み込んでいます
しばらく後消える
LAX 辞書編集
ソース名:sumomo
コマンド
(連接規則)
カテゴリ:
表　層:
左方接続:
右方接続:
カテゴリ置換
バッファ
ホルダ
キー検索
初期値設定
ソース変更
保存(ソース)
ソース:sumomo を設定しました。

図 2.45: ソース変更

カレント辞書の変更　カレント辞書をLAXに登録されている他の辞書に変更する方法について述べる。エディタ・コマンド・ウインドウの項目“ソース変更”をマウスで選択すると、辞書名選択ウインドウ(図 2.45)が表示される。

辞書名選択ウインドウには、LAXに登録されている辞書の名前がメニュー項目として表示される。カレント辞書(現在選択されている辞書)の名前は白抜き反転表示されている。辞書名をマウスで選択すると、

1. エディタ・メッセージ・ウインドウに“中間辞書をクローズします”というメッセージが表示され、カレント辞書の中間形式辞書ファイルがクローズされる。クローズが終了すると、“クローズが終了しました”というメッセージが表示される。
2. “中間辞書を読み込んでいます”というメッセージがテンポラリ・ウインドウに表示される。このウインドウが消えると、
   (a) カレント辞書が変更された場合、エディタ・メッセージ・ウインドウに“ソース:xxxを設定しました”というメッセージが表示され、エディタ・インタプリタ・ウインドウが表示される。
   (b) 指定された辞書の中間形式辞書ファイルが壊れている等の理由で、変更に失敗した場合、エディタ・インタプリタ・ウインドウに“設定に失敗しました。もとに戻します”というメッセージが表示され、元のエディタ・インタプリタ・ウインドウが表示される。

辞書名選択ウインドウで現在選択されている辞書を選択した場合は、エディタ・メッセージ・ウインドウに“変更ありません”というメッセージが表示される。マウス・マーカをこのウインドウから外した場合は、“変更を中止します”というメッセージが表示される。

ソース形式辞書ファイルの作成　(図 2.46を参照のこと)

逆トランスレートを起動して、中間形式辞書ファイルからソース形式辞書ファイルを作成する方法について述べる。エディタ・コマンド・ウインドウの項目“保存(ソース)”をマウスで選択すると、

1. エディタ・メッセージ・ウインドウに“出力パス名:xxxxx”と表示され入力待ちになる。ここに,xxxxxはソース形式辞書ファイルのパス名(省略値)である。ユーザは、別のパス名に変えたければ、書き換えた後<CR>を押下する。省略値のままでよければ単に<CR>を押下する。ソース形式辞書ファイルの作成を取り止める場合は、M−gを押下する。
2. 逆トランスレータが起動され、エディタ・メッセージ・ウインドウに、“辞書逆変換を開始します”等のメッセージが表示され、逆トランスレータが終了すると、“辞書逆変換を終了しました”というメッセージが表示される。

辞書エディタの終了　辞書エディタを終了させる方法について述べる。

1. 項目“インスペクタ”
   エディタ・コマンド・ウインドウの項目“インスペクタ”をマウスで選択すると、
   (a) 辞書エディタがLAXトップ・ウインドウから呼び出された場合
   辞書エディタのウインドウが消え、辞書エディタは終了した後、インスペクタが起動され、インスペクタの初期ウインドウが表示される。
   (b) 辞書エディタがインスペクタから呼び出された場合
   辞書エディタのウインドウが消え、辞書エディタは終了した後、インスペクタが起動され、インスペクタのウインドウが表示される。インスペクタのどのウインドウが表示されるかは、インスペクタ再表示指定(19頁を参照のこと)の値により異なる。
2. 項目“TOP”
   エディタ・コマンド・ウインドウの項目“TOP”は、LAXトップ・ウインドウから呼び出された場合のみ表示される。本項目をマウスで選択すると、辞書エディタのウインドウが消え、辞書エディタは終了する。ユーザは、LAXトップ・ウインドウから次の指示をLAXに与える。

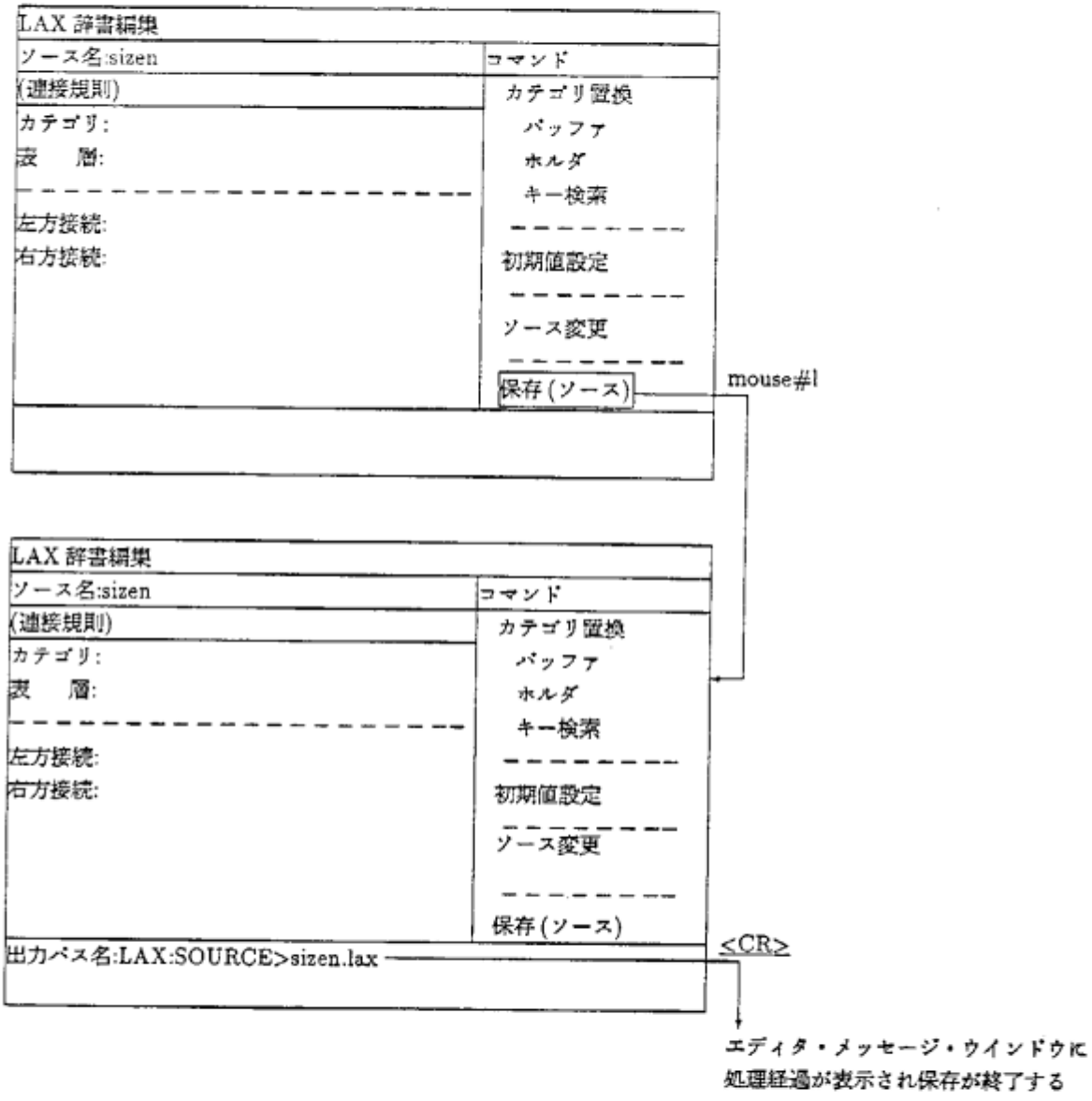


図 2.46: ソース形式辞書ファイルの作成

2.3.2 インスペクタ

インスペクタの操作方法を説明する。

起動方法

LAX トップ・ウインドウ (図 2.2) で、項目 "インスペクタ" をマウス左 1 回クリックで選択してインスペクタ・ウインドウ (初期ウインドウ) 図 2.47を作る[3]。カレント辞書が設定されていない場合、辞書名選択ウインドウ (図 2.5) が、表示されるので辞書を一つ選択する。

インスペクタのウインドウ

インスペクタ・ウインドウを図 2.47に示す。

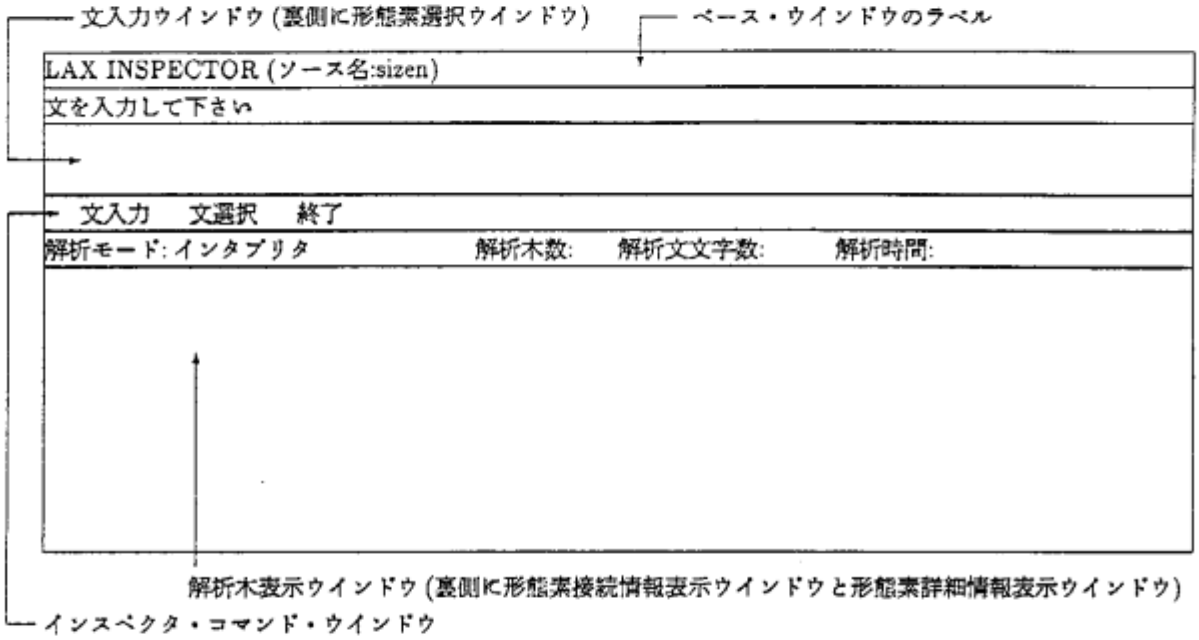


図 2.47: インスペクタ・ウインドウ

インスペクタ・ウインドウは、次の 9 つのウインドウから成る。

- ベース・ウインドウ
- 文入力ウインドウ
- 形態素選択ウインドウ
- インスペクタ・コマンド・ウインドウ
- 解析木表示ウインドウ
- 形態素接続情報表示ウインドウ (左方形態素接続情報表示ウインドウと右方形態素接続情報表示ウインドウ)
- 形態素詳細情報表示ウインドウ (左方形態素詳細情報表示ウインドウと右方形態素詳細情報表示ウインドウ)

[3]LAX 起動後、最初にインスペクタを呼び出した時のみ。既に、インスペクタを起動・終了した後ならば、その時作られたウインドウが使われるので新たにウインドウを開く必要はない

これらのウインドウのうち、ベース・ウインドウはラベルを除いて常に他のウインドウの下に隠されており、ユーザの目に触れることはない。ベース・ウインドウを除く8つのウインドウは、ベース・ウインドウの上に表示されるが、常に表示されているわけではない。

以下に、各ウインドウの説明を行う。

ベース・ウインドウ　ユーザが見ることのできるのは、ラベルだけで現在使用している辞書名が表示される。

インスペクタ・コマンド・ウインドウ　本ウインドウは、4種類の表示形式がある。

1. 初期ウインドウでのインスペクタ・コマンド・ウインドウ
   インスペクタを起動すると、本ウインドウが表示される(図 2.48)。

図 2.48: 初期ウインドウでのインスペクタ・コマンド・ウインドウ

   (a) 文入力
   項目“文入力”を選択すると、文入力ウインドウから入力できるようになる。
   (b) 文選択
   項目“文選択”を選択すると、文選択ウインドウが表示され文選択が可能になる(2.3.4を参照)。
   (c) 終了
   項目“終了”を選択すると、インスペクタは終了する。

2. 文入力待ち時のインスペクタ・コマンド・ウインドウ
   文入力待ち時のインスペクタ・コマンド・ウインドウは図 2.49に示すとおりで、入力はできない(文入力ウインドウからのみ入力できる)。

図 2.49: 文入力待ち時のインスペクタ・コマンド・ウインドウ

3. 解析木表示時のインスペクタ・コマンド・ウインドウ
   解析木表示時のインスペクタ・コマンド・ウインドウを図 2.50に示す。

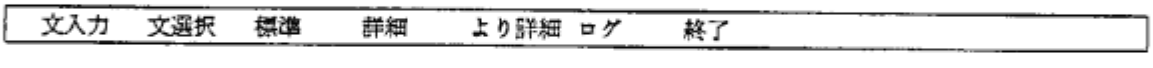


図 2.50: 解析木表示時のインスペクタ・コマンド・ウインドウ

   (a) 文入力
   項目“文入力”を選択すると、文入力ウインドウが表示され入力できるようになる。
   (b) 文選択
   項目“文選択”を選択すると、文選択ウインドウが表示され文選択が可能になる(2.3.4を参照)。
   (c) 標準
   項目“標準”を選択すると、解析木表示ウインドウに入力文の解析木が標準モードで表示される。標準モードでは、“トークン”のみが表示される。

(d) 詳細
項目“詳細”を選択すると、解析木表示ウインドウに入力文の解析木が詳細モードで表示される。詳細モードでは、“トークン”と“カテゴリ”が表示される。

(e) より詳細
項目“より詳細”を選択すると、解析木表示ウインドウに入力文の解析木がより詳細モードで表示される。より詳細モードでは、“トークン”、“カテゴリ”、“マトリクス”、“接続素性”が表示される。

(f) ログ
項目“ログ”を選択すると、入力文と解析木表示ウインドウに表示されている情報が、“INSLOG”という名のエディタ・テキスト・バッファに書き込まれる(アペンド)。ユーザは、この内容をSIMPOSのサブ・システムPmacsを使って参照することができる。

(g) 終了
項目“終了”を選択すると、インスペクタは終了する。

4. 形態素接続情報表示時のインスペクタ・コマンド・ウインドウ
形態素接続情報表示時のインスペクタ・コマンド・ウインドウを図2.51に示す。

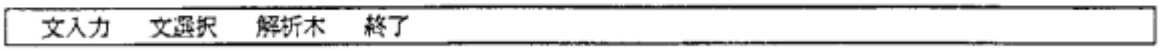


図 2.51: 形態素接続情報表示時のインスペクタ・コマンド・ウインドウ

(a) 文入力
項目“文入力”を選択すると、文入力ウインドウが表示され入力できるようになる。

(b) 文選択
項目“文選択”を選択すると、文選択ウインドウが表示され文選択が可能になる(2.3.4を参照)。

(c) 解析木
項目“解析木”を選択すると、解析木表示ウインドウが表示される。

(d) 終了
項目“終了”を選択すると、インスペクタは終了する。

文入力ウインドウ 文入力ウインドウを図2.52に示す。 文入力ウインドウからは、解析しようとするテキストを入力する。入力形式は、 テキスト<CR>である。<CR>のみを入力すると、解析は行われず直前に表示されていたウインドウが表示される。

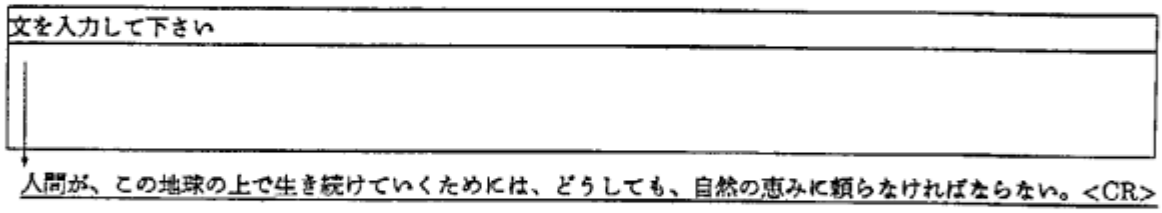


図 2.52: 文入力ウインドウ

形態素選択ウインドウ 形態素選択ウインドウを図2.53に示す。 形態素選択ウインドウに表示されている入力文の1文字を、マウス左1回クリックで指定することにより、接続情報を知りたい形態素(指定した1文字を先頭文字とする全ての形態素とそれらのすぐ左方にある形態素)を、形態素接続情報表示ウインドウに表示することができる。

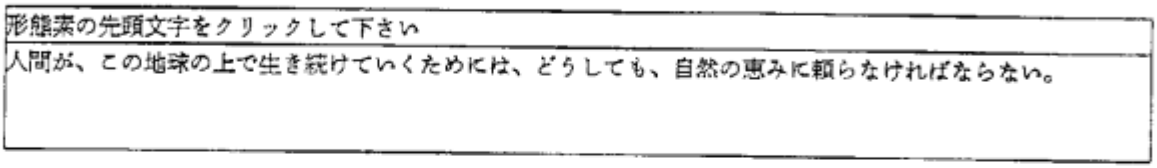


図 2.53: 形態素選択ウインドウ

解析木表示ウインドウ　解析木表示ウインドウを図 2.54に示す。
解析木表示ウインドウのラベルには、次の情報が表示される。

- 解析モード
  解析エンジンの動作モード(コンパイラ又はインタプリタ)(13頁を参照のこと)。
- 解析木数
  入力文解析結果のオールタナティブの数。
- 解析文文字数
  入力文の文字数。
- 解析時間
  解析エンジンが入力文の解析に要した時間。単位は、ミリ秒。

解析木表示ウインドウには、入力文の解析結果が解析木の形で表示される。
“・”は形態素の区切り、改行は文節の区切り、“/”は解析結果にオールタナティブが存在することを表す。形態素は、“トークン(カテゴリ,マトリクス,接続素性)”の形式で表示される。トークンが未登録語の場合は、トークンの後に“@”印が、左方形態素と接続しなかった場合は、トークンの後に“*”印が付けられる。

1. 解析木表示形式
   解析木表示形式には、次の3種類がある。
   (a) 標準
   “トークン”のみが表示される。
   (b) 詳細
   “トークン”、“カテゴリ”が表示される。
   (c) より詳細
   “トークン”、“カテゴリ”、“マトリクス”、“接続素性”が表示される。
2. 画面のスクロール
   解析木表示ウインドウの大きさに制限があるため、1解析木を1画面に表示できない場合がある。このような時、マウス・マーカをウインドウの左側いっぱいに移動させ、マウス左1回クリックすると、ウインドウは上方向にスクロールし、マウス中1回クリックするとウインドウは下方向にスクロールする。

形態素接続情報表示ウインドウ　形態素接続情報表示ウインドウを図 2.55に示す。 形態素接続情報表示ウインドウには、左方形態素接続情報表示ウインドウと右方形態素接続情報表示ウインドウとがある。右方形態素接続情報表示ウインドウに表示されるのは、形態素選択ウインドウで指定された文字を先頭に持つ形態素であり、通常複数個ある。左方形態素接続情報表示ウインドウに表示されるのは、右方形態素接続情報表示ウインドウに表示された形態素のすぐ左方にある全ての形態素である。形態素接続情報表示ウインドウは、スクロール機能を持ったメニュー・ウインドウであり、表示された形態素をマウスにより選択することにより、つぎのようなことが可能である。

解析モード: インタプリタ [SUCCESS] 解析木数:8 解析文文字数:46 解析時間:600msec
人間・が・ten こ・の 地球・の 上・で 生・き・続・け・て・い・く・ため・に・は・ten/生・き・続・け・て・い・く・ために・は・ どうしても・ten/ど・う・して・も・ten 自然・の 恵み・に

図 2.54: 解析木表示ウインドウ

1. mouse#1による選択
   形態素の詳細情報 (辞書内容) を知りたい場合は、形態素接続情報表示ウインドウの形態素をマウス左1回クリックで選択する。 選択された形態素の詳細情報 (辞書内容) を形態素詳細情報表示ウインドウに表示され、形態素接続情報表示ウインドウ上の形態素の頭に*印を付けて表示される。

2. mouse#11による選択
   特定の形態素とその左方あるいは右方にある形態素が、接続しているかどうか、どのように接続しているかを知りたい場合は、マウス左2回で形態素を選択する。

   (a) 選択された形態素と接続関係にある形態素の頭に番号 (1から始まる連番) が付けられて表示される (左方で形態素が選択された場合は、これと接続する右方の形態素、右方で形態素が選択された場合は、これと接続する左方の形態素)。

   (b) 選択された側では、その形態素の頭に、“*”印を付けられ、その詳細情報 (辞書内容) が形態素詳細情報表示ウインドウに表示される。さらに、実際に接続に使われた接続素性の前後に“★”印が付けられる (左方では右方接続素性、右方では左方接続素性)。

   (c) 反対側では、選択された形態素と接続関係にある形態素のうち番号1が付けられたものの詳細情報 (辞書内容) が、形態素詳細情報表示ウインドウに表示される。さらに、実際に接続に使われた接続素性の前後に“★”印が付けられる (左方では右方接続素性、右方では左方接続素性)。

   (d) 1以外の番号の付いた形態素と反対側の形態素との接続関係を知りたい場合は、マウス左1回で形態素を選択する。

   (e) 接続関係にある形態素がない場合は、マウス・マーカの位置に“接続形態素はありません”というテンポラリ・ウインドウが表示される。マウス・マーカをこのウインドウから外すことにより、このウインドウは消える。

3. mouse#mによる選択
   インスペクタから辞書エディタを呼び出して、辞書の内容を変更したい場合は、マウス中1回またはマウス中2回で、形態素を選択する。辞書エディタに変更したい形態素を知らせたい場合は、形態素をホルダ (2.3.1を参照) に登録する。マウス中1回で形態素を選択すると、その形態素はホルダに登録される。この時、テンポラリ・ウインドウに“ホルダに追加しました”というメッセージが表示される。マウス・マーカをこのウインドウから外すと、このウインドウは消える。

4. mouse#mmによる選択
   辞書エディタを呼び出したい場合は、マウス中2回で形態素を選択する。選択された形態素がホルダに登録された後、辞書エディタが呼び出され、インスペクタ・ウインドウは消える。辞書エディタから復帰すると、インスペクタ・ウインドウが再度表示される。

左方形態素	右方形態素
生 用言語基 ＊1 生 用言語基	き イ系活用助辞 き ウ系カ変活用助辞 き う系強変化カ活用助辞 ＊1 き 自動接辞
トークン：生 カテゴリ：用言語基 左方接続素性：end(文節) 右方接続素性：用言語基(★自動接辞き★, 他動接辞	トークン：き カテゴリ：自動接辞 左方接続素性：用言語基(★自動接辞き★) 右方接続素性：屈折(副など, 副なり, 係は, 終か, 係も,

図 2.55: 形態素接続情報表示ウインドウと形態素詳細情報表示ウインドウ

形態素詳細情報表示ウインドウ　形態素詳細情報表示ウインドウを図 2.55に示す。 形態素詳細情報表示ウインドウは、左方形態素詳細情報表示ウインドウと右方形態素詳細情報表示ウインドウがある。それぞれ、左方形態素接続情報表示ウインドウと右方形態素接続情報表示ウインドウと対応している。

1. 表示情報
   形態素接続情報表示ウインドウで“＊”印の付いた形態素の詳細情報(辞書内容)が表示される。左方形態素詳細情報表示ウインドウに表示された形態素の右方接続素性のうち“★”印が付いたものは、右方形態素詳細情報表示ウインドウに表示された形態素との接続に使われたものである。逆に、右方形態素詳細情報表示ウインドウに表示された形態素の左方接続素性のうち“★”印が付いたものは、左方形態素詳細情報表示ウインドウに表示された形態素との接続に使われたものである。

2. 画面のスクロール
   形態素詳細情報表示ウインドウの大きさに制限があるため、詳細情報を1画面に表示できない場合がある。このような時、マウス・マーカをウインドウの左側いっぱいに移動させ、マウス左1回クリックすると、ウインドウは上方向にスクロールし、マウス中1回クリックするとウインドウは下方向にスクロールする。

操作方法

文の入力と解析

1. 文の入力
   文を入力するには、インスペクタ・コマンド・ウインドウの項目“文入力”又は“文選択”をマウスで選択する。ただし、既に文入力ウインドウが表示されていれば、この操作は必要ない。

   (a) 文入力 (図 2.56を参照のこと)
   項目“文入力”を選択すると、文入力ウインドウが表示される。このウインドウに、テキストを“xxxxx<CR>”と入力する。文入力を中止したければ、単に<CR>を押下すればよい。

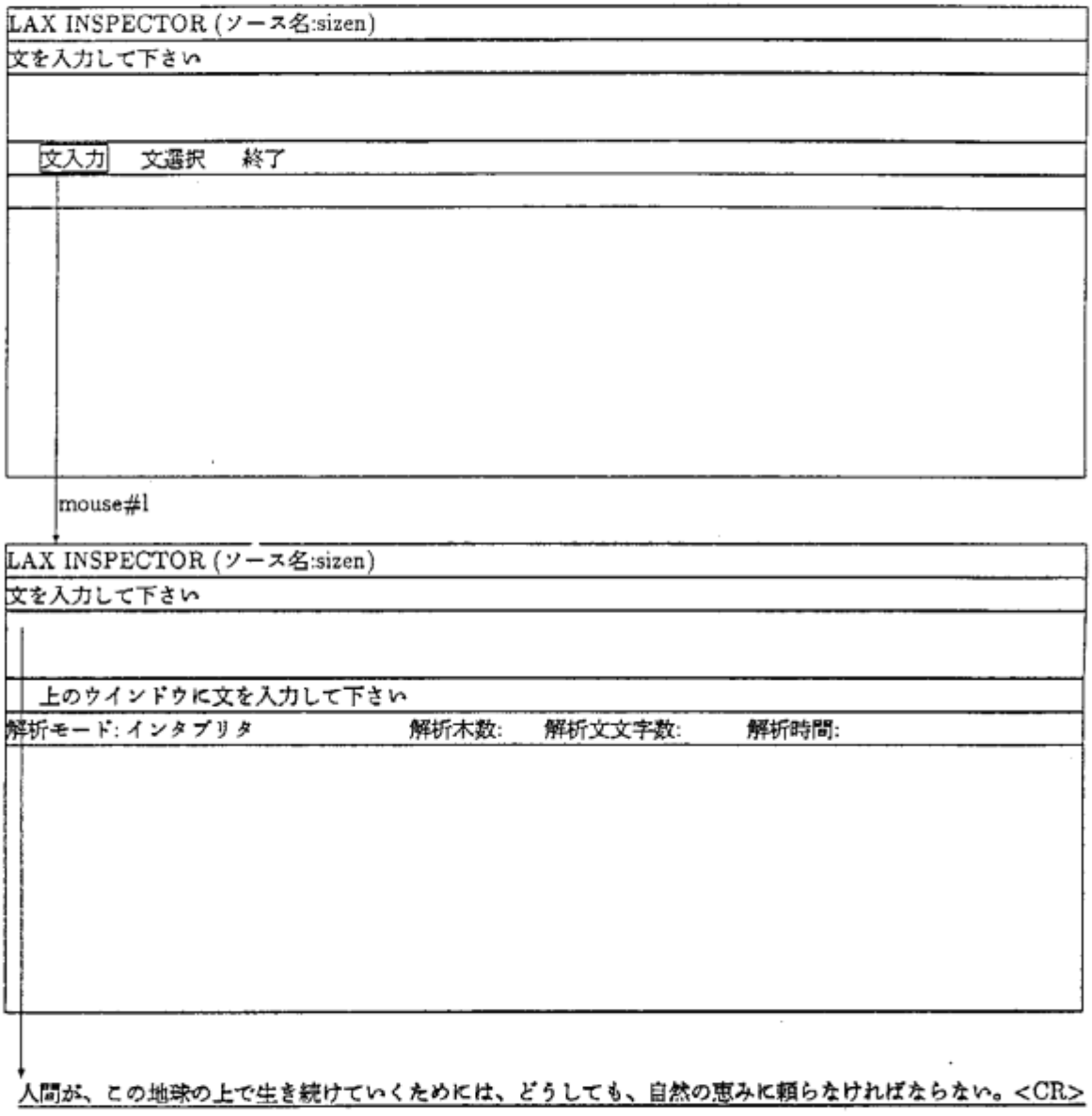


図 2.56: 項目“文入力”による文の入力

(b) 文選択 (図 2.57を参照のこと)
項目“文選択”を選択すると、文選択ウインドウが表示される。このウインドウに表示された1テキストをマウスで選択すると、その文が文入力ウインドウに表示される。このままの文を入力するのならば、単に<CR>を押下する。文を書き換えることもでき、この場合は文を書き換えた後、カーソルを文の最後に設定して<CR>を押下する。

2. 文の解析結果の表示
文を入力すると、解析結果が解析木表示ウインドウ(図 2.58を参照のこと) に表示される。

3. 解析木表示ウインドウの見かた

(a) 解析木表示ウインドウのラベル
解析木表示ウインドウのラベルには、次の情報が表示される。

- 解析モード(13頁を参照のこと)
  解析エンジンの動作モード (コンパイラ又はインタプリタ)。
- 解析木数
  入力文解析結果のオールタナティブの数。
- 解析文文字数
  入力文の文字数。
- 解析時間
  解析エンジンが入力文の解析に要した時間。単位は、ミリ秒。

(b) 解析木表示ウインドウ
解析木表示ウインドウには、入力文の解析結果が解析木の形で表示される。
“・”は形態素の区切り、改行は文節の区切り、“/”は解析結果にオールタナティブが存在することを表す。形態素は、“トークン (カテゴリ, マトリクス, 接続素性)”の形式で表示される。トークンが未登録語の場合は、トークンの後に“@”印が、左方形態素と接続しなかった場合は、トークンの後に“＊”印が付けられる。

4. 解析木表示ウインドウの操作
解析木表示ウインドウに対して次の操作ができる。

- 解析木表示形式の変更
- ログの取得
- 画面のスクロール

(a) 解析木表示形式の変更
解析木表示形式は、形態素のどの情報を解析木表示ウインドウに表示するかを表す項目である。 解析木表示形式には次の種類があり、インスペクタ・コマンド・ウインドウの項目“標準”、“詳細”、“より詳細”をマウスで選択するとそれぞれのモードに変わる。

標準
“トークン”のみが表示される。

詳細
“トークン”、“カテゴリ”が表示される。

より詳細
“トークン”、“カテゴリ”、“マトリクス”、“接続素性”が表示される。

(b) ログの取得
インスペクタ・コマンド・ウインドウの項目“ログ”を選択すると、入力文と解析木表示ウインドウに表示されている情報が、“INSLOG”という名のエディタ・テキスト・バッファに書き込まれる (アペンド)。ユーザは、この内容を SIMPOS のサブ・システム Pmacs を使って参照することができる。

(c) 画面のスクロール
解析木表示ウインドウの大きさに制限があるため、1解析木を1画面に表示できない場合がある。このような時、マウス・マーカをウインドウの左側いっぱいに移動させ、マウス左1回クリックすると、ウインドウは上方向にスクロールし、マウス中1回クリックするとウインドウは下方向にスクロールする。

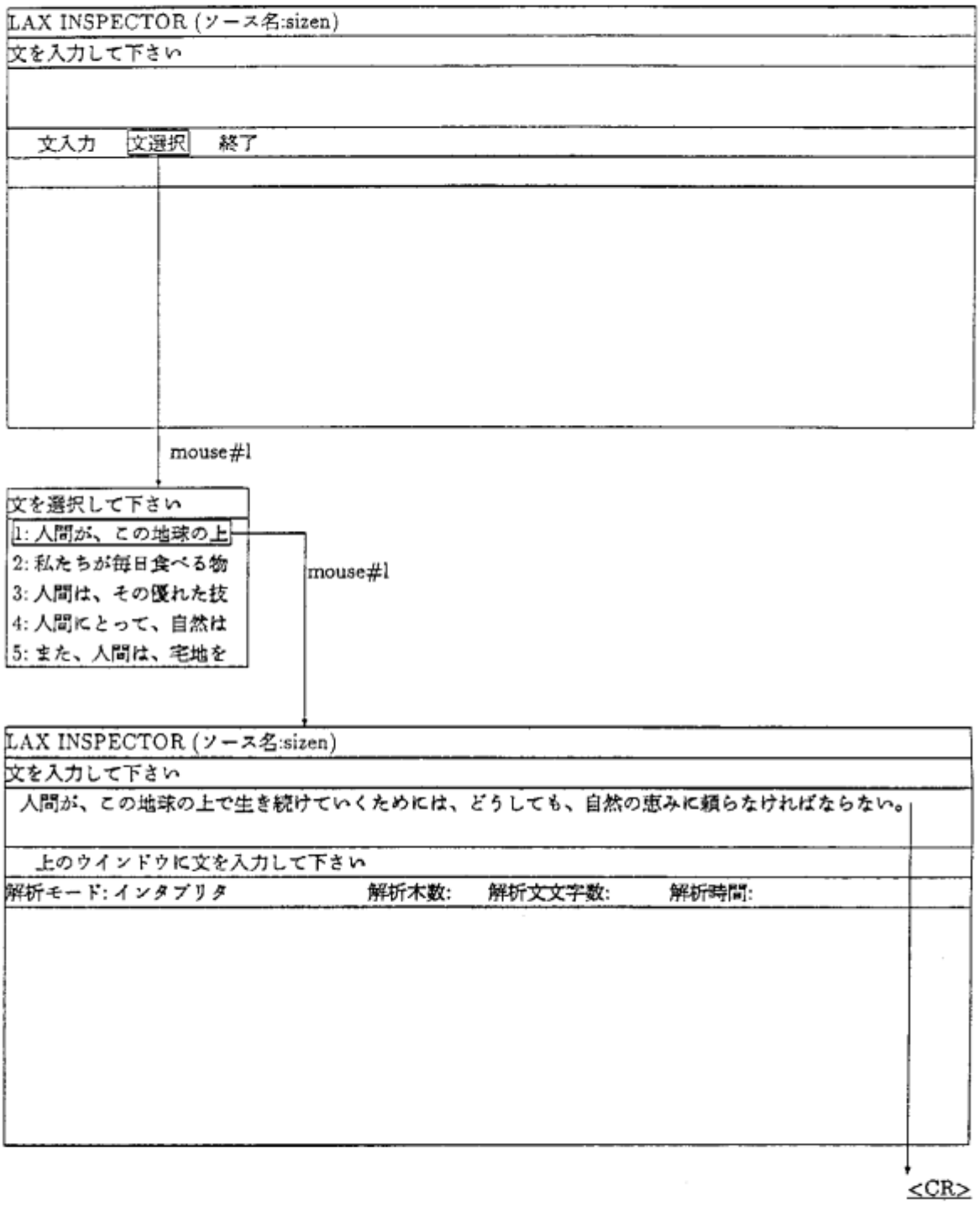


図 2.57: 項目“文選択”による文の入力

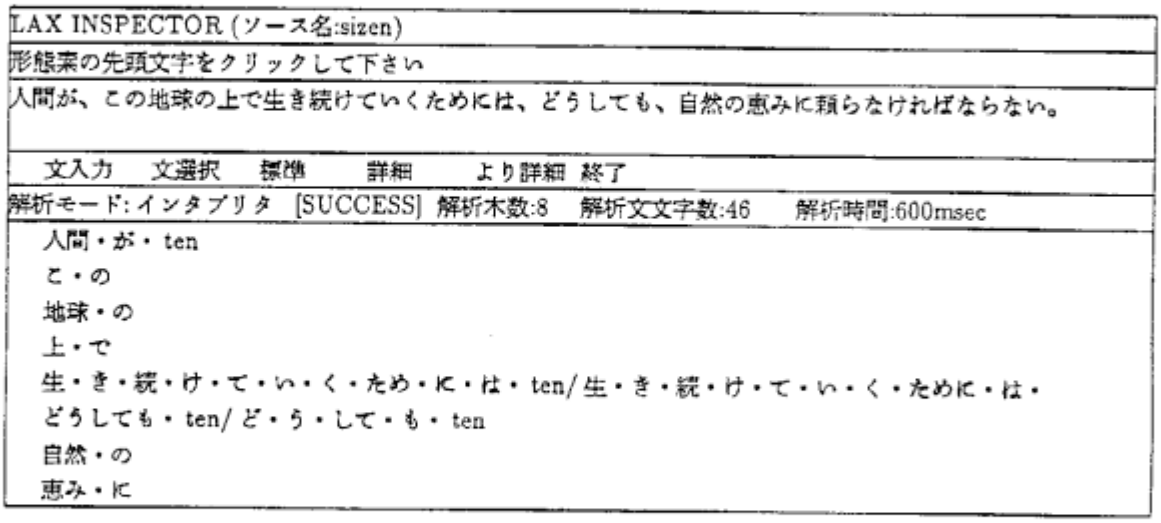


図 2.58: 解析木表示ウインドウ

形態素間の接続情報の表示　入力文中の形態素と形態素の接続関係を表示する方法について説明する (図 2.59を参照のこと)。

1. 右方形態素の指定
   2つの形態素間の接続関係を表示するには、形態素選択ウインドウに表示されている入力文の1文字をマウスでを指定することにより、右側の形態素の頭を指定する。

2. 接続関係の表示
   形態素間の接続情報は、左方形態素接続情報表示ウインドウと右方形態素接続情報表示ウインドウに表示される。右方形態素接続情報表示ウインドウには、マウスによって指定された文字を先頭に持つ形態素の表層とカテゴリが表示され、左方形態素接続情報表示ウインドウには、右方形態素の直ぐ左側にある形態素の表層とカテゴリが表示される。左方形態素と右方形態素がどのような接続関係にあるかを見るには、次のような操作を行う。

   (a) 左方形態素の1つと右方形態素のどれが、どのように接続しているかを見る場合
   左方形態素をマウス左2回クリックで選択する。もし、この形態素と接続している右方形態素があれば、

   - 選択された左方形態素の頭に“＊1”が表示される。
   - これと接続している右方形態素の頭に上から順に“1”,“2”,…が付き、更に“1”が付けられた形態素の頭には、“＊”印が付けられて表示される。
   - 左方形態素詳細情報表示ウインドウと右方形態素詳細情報表示ウインドウに、頭に“＊”印が付いた形態素の辞書内容が表示される。この辞書内容中の接続素性のうち、前後を“★”印ではさまれたもの (左方では右方接続素性、右方では左方接続素性) は、実際に接続に使われた接続素性を表す。

   右方形態素のうち、頭に“2”,“3”,…が付いた形態素と頭に“＊”が付いた左方形態素とが、どの接続素性で接続したかを見るには、右方形態素をマウス左1回クリックで選択する。

   左方形態素のどれとも接続していなければ、テンポラリ・ウインドウに“接続形態素はありません”というメッセージが表示される。

   (b) 右方形態素の1つと左方形態素のどれとがどのように接続しているかを見る場合
   上述した左方形態素の1つと右方形態素のどれとがどのように接続しているかを見る方法と同様である。

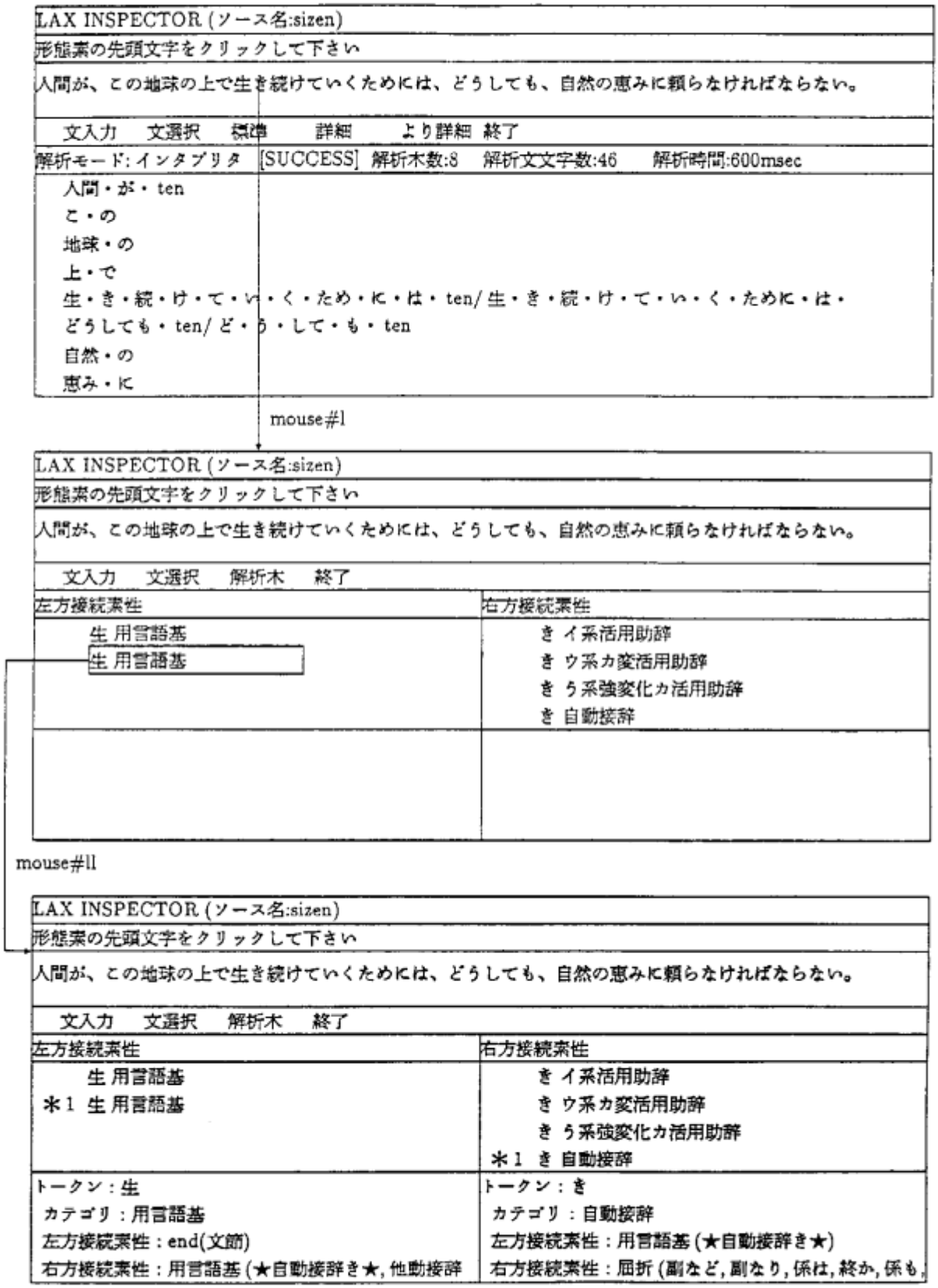


図 2.59: 形態素間の接続情報の表示

3. 形態素の辞書内容の表示
   形態素間の接続情報ではなく、単に形態素の辞書内容を見るには、左方又は右方形態素接続情報表示ウインドウに表示されている形態素をマウス左1回クリックで選択する。辞書内容は、左方又は右方形態素詳細情報表示ウインドウに表示される。

辞書エディタの呼び出し　インスペクタから辞書エディタを動的に呼び出すことができる。呼び出す時、形態素(例えば、辞書エディタで修正したい形態素)をホルダ(2.3.1を参照)に登録して辞書エディタに引き渡す(図 2.60を参照のこと)。

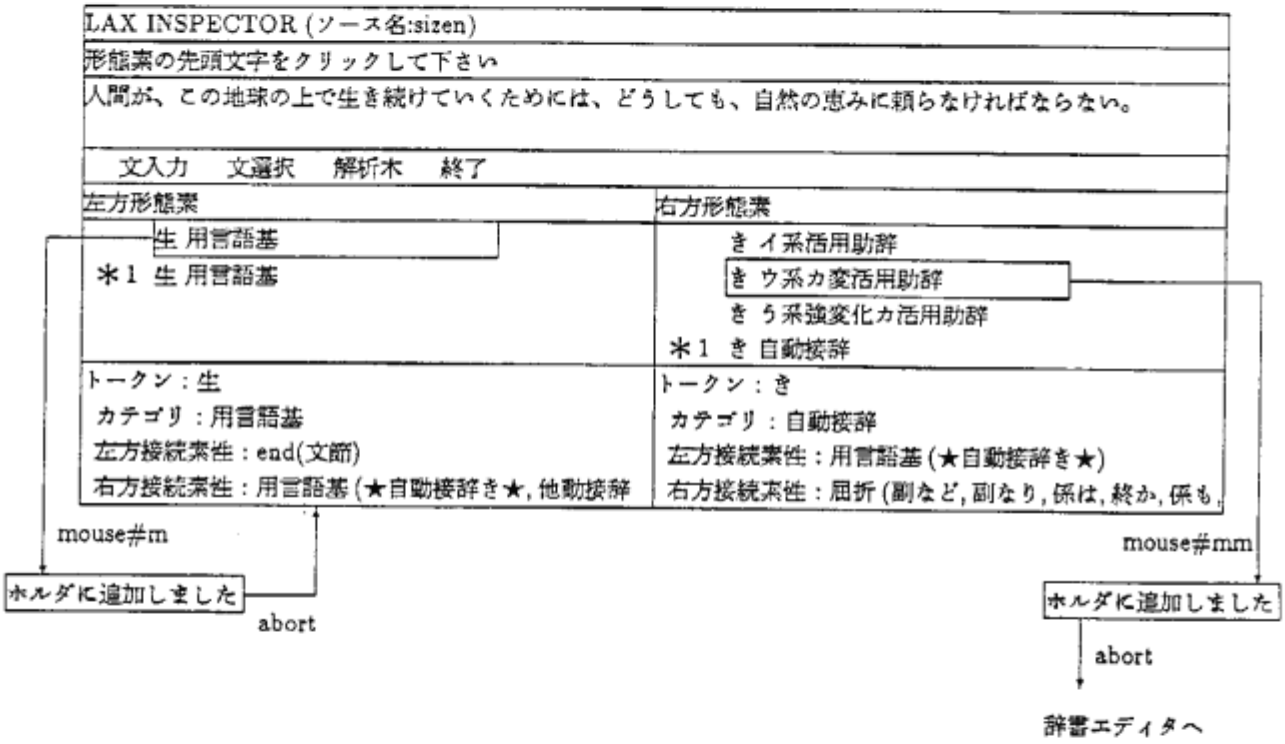


図 2.60: 辞書エディタの呼び出し

1. 形態素のホルダへの登録
   左方又は右方形態素接続情報表示ウインドウに表示されている形態素を、マウス中1回クリックで選択するとホルダに登録される。この時、テンポラリ・ウインドウに“ホルダに追加しました”というメッセージが表示される。この操作を繰り返すことにより、ホルダに複数の形態素を登録することができる。

2. 辞書エディタの呼び出し
   左方又は右方形態素接続情報表示ウインドウに表示されている形態素をマウス中2回クリックで選択する。選択された形態素は、ホルダに登録され、インスペクタのウインドウが消えて辞書エディタが起動される。辞書エディタの操作方法は、2.3.1を参照されたい。

3. 辞書エディタからの戻り
   辞書エディタを終了させると、インスペクタのウインドウが表示される。インスペクタのどのウインドウが表示されるかはインスペクタ再表示位置によって決まる。インスペクタ再表示位置は、LAX トップ・ウインドウの項目“環境設定”によってユーザが指定できる(19頁を参照のこと)。

(a) インスペクタ再表示位置 = 解析木
入力文が解析し直され、結果が解析木表示ウインドウに表示された状態に戻る。

(b) インスペクタ再表示位置 = 形態素接続素性
入力文が解析し直され、更に前回マウスで文字位置指定された形態素の接続情報を表示した形態素接続情報表示ウインドウが表示された状態に戻る。

(c) 辞書エディタ操作中にカレント辞書を変更した場合
インスペクタ再表示位置の値如何にかかわらず、インスペクタの初期ウインドウが表示される。

インスペクタの終了 インスペクタ・コマンド・ウインドウの項目“終了”をマウスで選択する。文入力ウインドウが表示されている時は、<CR>を押下後、項目“終了”をマウスで選択する。

2.3.3 プリティ・プリンタ

プリティ・プリンタの操作方法を説明する。

プリティ・プリンタの起動

LAX トップ・ウインドウのコマンド・ウインドウの項目“プリティ・プリンタ”をマウスで選択 (又は、インタプリタ・ウインドウから同等のコマンドprinter<CR>を入力) すると、

1. “ウインドウを開いて下さい”というテンポラリ・ウインドウが表示される[4]ので、ウインドウの位置と大きさを適当に決めウインドウを開く。もし、ユーザの指定したウインドウの位置、大きさが不適当であれば、プリティ・プリンタが自動的にそれらを調整してウインドウを開く。ユーザは、プリティ・プリンタ起動中いつでも、ウインドウの位置と大きさを変更することができる (但し、ウインドウをより大きくすることはできるが、より小さくはできない)。

2. プリティ・プリンタの初期ウインドウが表示される。

なお、カレント辞書が設定されていない場合、辞書名選択ウインドウ (図 2.5) が、表示されるので辞書を一つ選択する。

ウインドウの構成

図 2.61に示すように、プリティ・プリンタは 3 つのウインドウからなる。

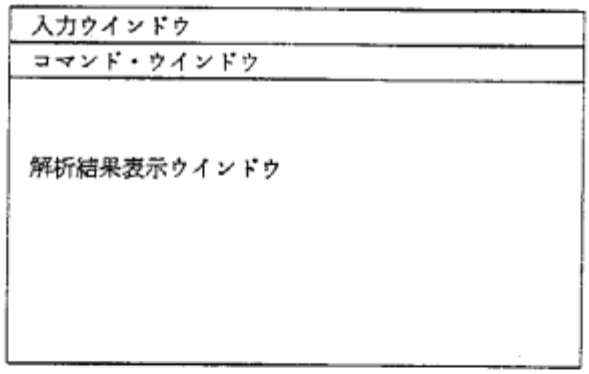


図 2.61: プリティ・プリンタのウインドウ

入力ウインドウ LAX の解析の対象となる文を入力するウインドウであり、Pmacs ウインドウの機能が使える。

コマンド・ウインドウ プリティ・プリンタの操作コマンドのメニュー (図 2.62を参照のこと) であり、マウスで項目の 1 つを選択する。

1. 文入力
   項目“文入力”を選択すると、入力ウインドウからの入力待ちになる。そこで、解析の対象とする文を入力する。文入力は<CR>の押下で終了と見なされ、その解析結果が解析結果表示ウインドウに表示される。

[4]LAX 起動後、最初にプリティ・プリンタを呼び出した時のみ。既に、プリティ・プリンタを起動・終了した後ならば、その時作られたウインドウが使われるので新たにウインドウを開く必要はない

文頭	文末	<<	<	>	>>	up	down	文入力	文選択	終了

図 2.62: コマンド・ウインドウ

2. 文選択
項目“文選択”を選択すると、文選択ウインドウ (2.3.4を 参照のこと) が表示される。表示された文の 1 つを選択すると、その文が入力ウインドウに表示される。<CR>を押下すると入力は終了する。

3. 終了
項目“終了”を選択すると、ウインドウが見えなくなりプリティ・プリンタは終了する。ただし、ウインドウは見えなくなるだけで残っているので、再起動時には新たにウインドウを開く必要はない。

入力文が長い場合や解析結果が複雑な場合、解析結果表示ウインドウに出力が収まり切らない場合がある。この場合、以下の項目を選択することにより、ウインドウをスクロールさせて全体を見ることができる (図 2.63)。

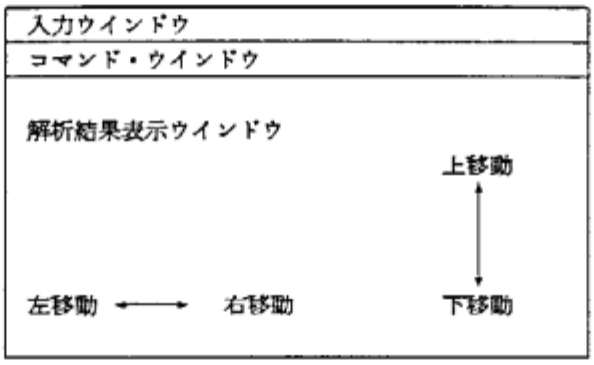


図 2.63: 表示画面の移動

4. 文頭
解析結果表示ウインドウの左端に文頭を移動する。

5. 文末
解析結果表示ウインドウの右端に文末を移動する。

6. < (左移動)
表示画面を左方向に 1/4 画面分スクロールする。ただし、文末が表示されている時はスクロールしない。

7. > (右移動)
表示画面を右方向に 1/4 画面分スクロールする。ただし、文頭が表示されている時はスクロールしない。

8. << (左移動)
表示画面を左方向に 1/2 画面分スクロールする。ただし、文末が表示されている時はスクロールしない。

9. >> (右移動)
表示画面を右方向に 1/2 画面分スクロールする。ただし、文頭が表示されている時はスクロールしない。

10. down (上移動)
1/2 画面スクロール・ダウンする。

11. up (下移動)
    1/2画面スクロール・アップする。

解析結果表示ウインドウ 入力文の解析結果を表示するウインドウである。プリティ・プリンタは、文章毎にブロック化して解析結果表示ウインドウに表示する。トークンとして同じものであっても、文法的に異なるものであれば別のものとして表示する。文節の区切りは−で、文節内の語の区切りは“・”で表す。

(表示例)

操作方法

文の入力と解析

1. 文の入力
   文を入力するには、コマンド・ウインドウの項目“文入力”または“文選択”を選択する。

   (a) 文入力
   項目“文選択”を選択すると、入力ウインドウから文を入力できるようになる。そこで解析したい文を入力する(<CR>で入力終了)。

   (b) 文選択
   項目“文選択”を選択すると、文選択ウインドウ(2.3.4を参照のこと)が表示される。表示された文の1つを選択すると、その文が入力ウインドウに表示される。そのまま解析したければ<CR>を押下すればよい。変えたければ修正後<CR>を押下すればよい。

2. 解析結果の表示
   入力された文は解析エンジンに渡され、その解析結果が解析結果表示ウインドウに表示される。

3. 画面のスクロール
   入力文が長い場合とか解析結果が複雑な場合には、解析結果表示ウインドウに解析結果を一度に全てを表示できないことがある。この時は、コマンド・ウインドウの項目“>”、“<”、“<<”、“>>”、“up”、“down”のうちから移動したい方向の操作項目を選んで画面をスクロールさせれば、別の部分を見ることができる。

4. プリティ・プリンタの終了
   コマンド・ウインドウの項目“終了”を選択する。

2.3.4 センテンス・セレクタ

センテンス・セレクタの操作方法を説明する。

起動及び操作方法

センテンス・セレクタは、LAXの他のツールと異なり、LAXトップ・ウインドウから呼び出すことはできず、インスペクタまたはプリティ・プリンタから呼び出すことにより使用する。インスペクタまたはプリティ・プリンタのコマンド・ウインドウの項目“文選択”をマウスでクリックすると、図 2.64に示す文選択ウインドウが表示される。なお、センテンス・セレクタは、入力テキスト・ファイルに書かれているテキストを読み込んで文選択ウインドウに表示する。入力テキスト・ファイルがない場合または入力テキスト・ファイルにテキストが書かれていない場合は、文選択属性変更ウインドウが表示されるので、項目“入力ファイル名”の値を変えた後、再度項目“文選択”を選択する。あるいは、入力ファイル名は変えないで、SIMPOSのサブ・システムPmacsによってこの入力ファイルを作成後、再度項目“文選択”を選択してもよい。

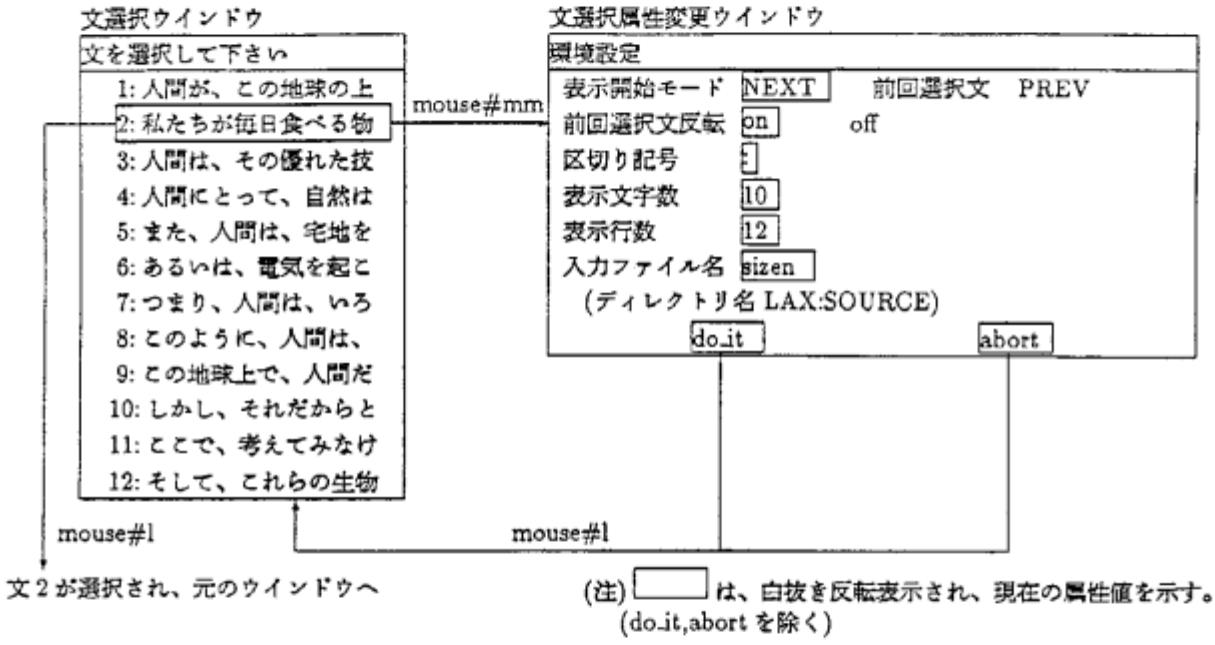


図 2.64: センテンス・セレクタのウインドウ

文の選択

文選択ウインドウには、テキストが選択可能なメニュー項目として表示される。テキストの個数が表示可能なサイズを越える場合は、本ウインドウをスクロールさせることにより表示させることができる。文選択ウインドウの1項目をマウス左1回クリックで選択すると、そのテキストが選択され、ウインドウは消える。文選択ウインドウからマウス・マーカを外すと、テキストは選択されることなくウインドウは消える。

文選択ウインドウの属性変更

文選択ウインドウの1項目をマウス中2回クリックすると、文選択属性変更ウインドウが表示される。文選択属性変更ウインドウは、文選択ウインドウの種々の属性を変更するためのウインドウである。変更したい属性の項目値をマウスで選択した後、項目“do_itを選択すると、属性が変更されウインドウが消える。 項目“abortを選択すると、属性は変更されない。変更可能な属性は、以下のようなものである。

- 表示開始モード
  文選択ウインドウが表示された時、マウス・マーカがどのテキスト上に位置するかを指定するものであり、次

の3つのうちの1つを選択できる。

- 前回選択テキストの次のテキスト
  項目値 "NEXT" を選択する。
- 前回選択テキスト
  項目値 "前回選択文" を選択する。
- 前回選択テキストの前のテキスト
  項目値 "PREV" を選択する。

- 前回選択文反転
  前回選択テキストを白抜き反転表示するかどうかを指定するものであり、次の2つのうちの1つを選択できる。
  - 反転する
    項目値 "on" を選択する。
  - 反転しない
    項目値 "off" を選択する。

- 区切り記号
  文選択ウインドウに表示されるテキストの頭に付けて表示する識別記号を5文字以下の文字列で指定する。

- 表示文字数
  文選択ウインドウに表示されるテキストはテキスト全文ではなく、各テキストの頭の部分である。本項目では、頭の部分の何文字を表示するかを数字で指定する。

- 表示行数
  文選択ウインドウにテキスト何文を表示するかを数字で指定する。未表示テキストは、ウインドウをスクロールすることによって表示させることができる。

- 入力ファイル名
  文選択ウインドウに表示されるテキストは、入力テキスト・ファイルに書かれている。本項目では、入力テキスト・ファイル名を20文字以下の文字列で指定する。

入力テキスト・ファイル

入力テキスト・ファイルのパス名は、

論理パス名 > ディレクトリ名 > 入力テキスト・ファイル名 > 拡張子名

という形式をしている。これらのうち、ユーザが指定できるのは入力テキスト・ファイル名のみである。
論理パス名は、LAXのインストール時に作成され LAX と決まっている。ディレクトリ名は SOURCE 、拡張子名は txt である。

2.4 出力メッセージ

2.4.1 トップ・ウインドウの出力メッセージ

1. end(xxx) が見つかりません。
   xxx は、カテゴリ名。

   [ツール] トップ・ウインドウ (トランスレータ)

   [メッセージの意味] begin(カテゴリ名) に対応する end(カテゴリ名) がない。

   [ユーザの対応] ソース形式辞書ファイルを修正して、再度トランスレータを実行する。

   [関連参照ページ]

2. インクルード・ファイル名が不適当です。

   [ツール] トップ・ウインドウ (トランスレータ)

   [メッセージの意味] 指定されたインクルード・ファイルが存在しない。

   [ユーザの対応] ソース形式辞書ファイルを修正後、再度トランスレータを起動する。

   [関連参照ページ]

3. インタプリタ・モードに変わりました。

   [ツール] トップ・ウインドウ (モード変更)

   [メッセージの意味] 解析モード (解析エンジンの実行モード) が、コンパイラ・モードからインタプリタ・モードに変わった。

   [ユーザの対応] なし。

   [関連参照ページ]

4. オブジェクトがありません。

   [ツール] トップ・ウインドウ (コンパイラ・モードでの、インスペクタまたはプリティ・プリンタの起動)

   [メッセージの意味] 指定された辞書のオブジェクト・プログラム (解析エンジン) がロードされていない。

   [ユーザの対応] 以下のいずれかを行った後、再度インスペクタまたはプリティ・プリンタを起動する。

   - 既に、オブジェクト形式辞書ファイルが存在していれば、LAX トップ・ウインドウの項目“ファイル・カタログ”を選択して、オブジェクト・プログラムをコンパイル、カタログする。
   - オブジェクト形式辞書ファイルが存在していなければ、LAX トップ・ウインドウの項目“バッファ・カタログ”を選択してオブジェクト・プログラムの作成とコンパイル、カタログを行う。

   [関連参照ページ]

5. ーオブジェクト・ファイルを読み込んでいますー

   [ツール] トップ・ウインドウ (モード変更)

   [メッセージの意味] オブジェクト・プログラム (解析エンジン) を主記憶上にロードする。

   [ユーザの対応] なし。

   [関連参照ページ]

6. オブジェクトをコンパイルします。

   [ツール] トップ・ウインドウ (オブジェクト作成)

   [メッセージの意味] カタログ・コンパイルを開始する。

   [ユーザの対応] なし。

   [関連参照ページ]

7. オブジェクトをセーブします。

   [ツール] トップ・ウインドウ(オブジェクト作成、終了)

   [メッセージの意味] カタログ・コンパイル済みのクラスをセーブする。

   [ユーザの対応] なし。

   [関連参照ページ]

8. オブジェクトをセーブしますか?[y/n]

   [ツール] トップ・ウインドウ(終了)

   [メッセージの意味] オブジェクト・プログラム(解析エンジン)のカタログ・コンパイルがなされたが、セーブされていないクラスがある。このあるいはこれらのクラスをセーブするかどうか。

   [ユーザの対応] セーブする場合はyを、セーブしない場合はnを入力する。セーブしない場合、次回PSIを起動した時、カタログ・コンパイルの効果は反映されない。

   [関連参照ページ]

9. オブジェクトをもったソースを指定して下さい。

   [ツール] トップ・ウインドウ(オブジェクト作成)

   [メッセージの意味] 指定されたオブジェクト形式辞書ファイルが存在しない。

   [ユーザの対応] 以下のいずれかを行う。

   - 再度項目“ファイル・カタログ”を選択して、オブジェクト形式辞書ファイルの正しいパス名を入力する。
   - 項目“ファイル作成”を選択してオブジェクト形式辞書ファイルを作成後、再度項目“ファイル・カタログ”を選択する。

   [関連参照ページ]

10. オブジェクトをバッファに作成します

    [ツール] トップ・ウインドウ(オブジェクト作成)

    [メッセージの意味] 中間形式辞書ファイルからオブジェクト・プログラム(解析エンジン)をスタンダード入出力バッファ上に作成する(オブジェクト形式辞書ファイルは作成されない)。

    [ユーザの対応] なし。

    [関連参照ページ]

11. カテゴリ名が一致しません。 :xxx
    xxxは、カテゴリ名

    [ツール] トップ・ウインドウ(トランスレータ)

    [メッセージの意味] begin(カテゴリ名)に対応するend(カテゴリ名)がない(一致しない)。

    [ユーザの対応] ソース形式辞書ファイルを修正して、再度トランスレータを実行する。

    [関連参照ページ]

12. カテゴリxxxに属する形態素はありません。

    [ツール] トップ・ウインドウ(逆トランスレータ)または辞書エディタ

    [メッセージの意味] カテゴリxxxを持つ形態素は1件もない。辞書エディタで、カテゴリxxxを持つ形態素が全て削除された。

    [ユーザの対応] なし。

    [関連参照ページ]

13. カテゴリ xxx を編集します (n件)
xxx はカテゴリ、nはカテゴリ xxx を持つ形態素の個数。

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] カテゴリ xxx を持つn個の形態素が、ソース形式辞書ファイルに書き込まれる。

[ユーザの対応] なし。

[関連参照ページ]

14. クローズしました。

[ツール] トップ・ウインドウまたは辞書エディタ

[メッセージの意味] 中間形式辞書ファイルのクローズを終了した。

[ユーザの対応] なし。

[関連参照ページ]

15. コマンドがありません

[ツール] トップ・ウインドウ

[メッセージの意味] LAX トップ・ウインドウのインタプリタ・ウインドウからコマンドとして許されていない文字列が入力された。

[ユーザの対応] 正しいコマンドを入力する。

[関連参照ページ]

16. コンパイラ・モードに変わりました。

[ツール] トップ・ウインドウ (モード変更)

[メッセージの意味] 解析モード (解析エンジンの実行モード)が、インタプリタ・モードからコンパイラ・モードに変わった。

[ユーザの対応] なし。

[関連参照ページ]

17. コンパイル終了しました。

[ツール] トップ・ウインドウ (オブジェクト作成)

[メッセージの意味] カタログ・コンパイルが終了した。

[ユーザの対応] なし。

[関連参照ページ]

18. 辞書逆変換を開始しました

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 逆トランスレータが起動された。

[ユーザの対応] なし。

[関連参照ページ]

19. 辞書逆変換に失敗しました

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 逆トランスレータが正常に終了しなかった。

[ユーザの対応] 中間形式辞書ファイル、ソース形式辞書ファイルが共に壊れている可能性があるので、バックアップ・ファイルがあればコピーして復旧する。

[関連参照ページ]

20. 辞書逆変換を終了しました

[ツール] トップ・ウインドウ(逆トランスレータ)または辞書エディタ

[メッセージの意味] 逆トランスレータが正常に終了し、中間形式辞書ファイルからソース形式辞書ファイルが作成された。

[ユーザの対応] なし。

[関連参照ページ]

21. 出力パス名:xxx.lax　xxxは、ソース形式辞書ファイルのパス名(ファイルの拡張子を除く)

[ツール] トップ・ウインドウ(逆トランスレータ)または辞書エディタ

[メッセージの意味] 作成されるソース形式辞書ファイルのパス名を問い合わせている。

[ユーザの対応] ソース形式辞書ファイルのパス名がxxx.laxでよければ<CR>を押下する。変えたければ変更後<CR>を押下する。

[関連参照ページ]

22. シンタックス・エラー : xxx

[ツール] トップ・ウインドウ(トランスレータ)または辞書エディタ

[メッセージの意味] ソース形式辞書ファイルの記述に誤りがある。xxxは、誤りのある行の内容。

[ユーザの対応] ソース形式辞書ファイルを修正して、再度トランスレータを実行する。

[関連参照ページ]

23. 接続素性名が多すぎます。(マトリクス : xxx)

[ツール] トップ・ウインドウ(トランスレータ)または辞書エディタ

[メッセージの意味] マトリクスxxxに属する接続素性名の個数が、ソース形式辞書ファイルの宣言部の“素性数/1マトリクス”で定義された値を越えた。

[ユーザの対応] ソース形式辞書ファイルの宣言部の“素性数/1マトリクス”の値を大きくして、再度トランスレータを実行する。

[関連参照ページ]

24. 設定に失敗しました。もとに戻します。

[ツール] トップ・ウインドウまたは辞書エディタ(ソース変更)

[メッセージの意味] 辞書の変更ができなかったので、カレント辞書を変更前の辞書に戻した。

[ユーザの対応] 辞書の変更に失敗したのは、何らかの理由で変更後の中間形式辞書ファイルが壊れているためなので、バックアップ・ファイルがあればコピーすることにより復旧する。

[関連参照ページ]

25. 宣言部の記述が不適当です。

[ツール] トップ・ウインドウ(トランスレータ)

[メッセージの意味] ソース形式辞書ファイルの宣言部の記述に誤りがある。トランスレータは処理を打ち切った。

[ユーザの対応] ソース形式辞書ファイルの宣言部の記述を修正後、再度トランスレータを起動する。

[関連参照ページ]

26. ソースxxxを設定しました。

[ツール] トップ・ウインドウ(ソース変更)または辞書エディタ(ソース変更)

[メッセージの意味] カレント辞書を辞書xxxにした。

[ユーザの対応] なし。

[関連参照ページ]

27. ソース・ディレクトリ: ××× 入力ファイル名:xxx
××× はソース形式辞書ファイルのディレクトリ、xxx はソース形式辞書ファイルのファイル名(ファイル拡張子を除く)。

[ツール] トップ・ウインドウ(トランスレータ)

[メッセージの意味] トランスレートの対象となるソース形式辞書ファイルのファイル名の問い合わせである。

[ユーザの対応] ファイル名が表示されたものでよければ、<CR>を押下する。変えたければ、変更後<CR>を押下する。なお、ディレクトリ名は固定されており変更できない。

[関連参照ページ]

28. ソースがありません

[ツール] トップ・ウインドウ(インタプリタ・モードでの、インスペクタまたはプリティ・プリンタの起動)

[メッセージの意味] 指定された辞書の中間形式辞書ファイルが存在しない。

[ユーザの対応] LAX トップ・ウインドウの項目“トランスレート”または“トランスレート&カタログ”を選択して中間形式辞書ファイルを作成した後、再度インスペクタまたはプリティ・プリンタを起動する。

[関連参照ページ]

29. !!注意!! 形態素を二重定義しています : 形態素 > ××× カテゴリ > xxx 続行しますか / ?[y/n]:

[ツール] トップ・ウインドウ(トランスレータ)

[メッセージの意味] 全く同じ形態素が2つあるいは2つ以上ある。これらの形態素のうち、2番目以降は書き出されない。

[ユーザの対応] トランスレートを続ける場合はyを、中止する場合はnを入力する。

[関連参照ページ]

30. !!注意!! 辞書のクラス数を以下のように見なします。

[ツール] トップ・ウインドウ(トランスレータ)または辞書エディタ

[メッセージの意味] ソース形式辞書ファイルの宣言部で定義されている辞書のクラス数を、本メッセージの次に表示される数に置き換えた。ユーザが定義したクラス数をnとすると、nより大きくかつ最も近い2のべき乗に置き換えられる。

[ユーザの対応] なし。

[関連参照ページ]

31. 中間ファイルをクローズします。

[ツール] トップ・ウインドウ(ソース変更)または辞書エディタ

[メッセージの意味] 辞書の変更にともない現在まで使われていた辞書をクローズする。

[ユーザの対応] なし。

[関連参照ページ]

32. 中間辞書を読み込んでいます

[ツール] トップ・ウインドウ(逆トランスレータ)または辞書エディタ

[メッセージの意味] 中間形式辞書ファイルの内容を、主記憶上に読み込んでいる。

[ユーザの対応] しばらく待つと、本メッセージが表示されているテンポラリ・ウインドウが消える。

[関連参照ページ]

33. 中間ファイルのバックアップをとります

[ツール] トップ・ウインドウ (終了)

[メッセージの意味] 中間形式辞書ファイルのコピー・ファイルを作成する。

[ユーザの対応] なし。

[関連参照ページ]

34. 中間ファイルをもったソースを指定して下さい

[ツール] トップ・ウインドウ (逆トランスレートまたはオブジェクト作成)

[メッセージの意味] 指定された中間形式辞書ファイルが存在しない。

[ユーザの対応]

- 逆トランスレートは実行できない。
- オブジェクト作成の場合、LAX トップ・ウインドウの項目“トランスレート”を選択して、中間形式辞書ファイルを作成後、オブジェクト作成を行うか、LAX トップ・ウインドウの項目“トランス&カタログ”を選択して、中間形式辞書ファイルの作成とオブジェクトの作成を行う。

[関連参照ページ]

35. パス名が見つかりません。

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] ソース形式辞書ファイルのパス名のうち、ディレクトリが正しくない (存在しない)。

[ユーザの対応] 再度逆トランスレータを起動して、既存のディレクトリを指定する。または、SIMPOS のサブ・システム、ファイル・マニピュレータでディレクトリを作成してもよい。

[関連参照ページ]

36. バックアップ・ファイルが存在しません。

[ツール] トップ・ウインドウ (リカバリ)

[メッセージの意味] 中間形式辞書ファイルのコピー・ファイルが存在しないため、中間形式辞書ファイルをリカバリできない。

[ユーザの対応] この辞書の中間形式辞書ファイルが存在すれば、トランスレータを起動して中間形式辞書ファイルを作成する。

[関連参照ページ]

37. 表層 xxx の次で中止しました

[ツール] トップ・ウインドウ (トランスレータ)

[メッセージの意味] 表層の二重登録時に表示される問い合わせメッセージに対して、n を入力した時表示され、トランスレータの実行が中断した。

[ユーザの対応] ソース形式辞書ファイルから二重登録されている形態素を削除後、再度トランスレータを実行する。

[関連参照ページ]

38. *** ファイルが存在しません ***

[ツール] トップ・ウインドウ (トランスレータ)

[メッセージの意味] 指定されたファイル名を持つソース形式辞書ファイルが存在しない。

[ユーザの対応] 再度トランスレータを起動し、正しいソース形式辞書ファイル名を指定する。

[関連参照ページ]

39. ファイルをクローズしています。

[ツール] トップ・ウインドウ (トランスレータ)

[メッセージの意味] トランスレータ実行のため、中間形式辞書ファイルをクローズする。

[ユーザの対応] なし。

[関連参照ページ]

40. ファイルをクローズします。

[ツール] トップ・ウインドウまたは辞書エディタ

[メッセージの意味] 中間形式辞書ファイルをクローズする。

[ユーザの対応] なし。

[関連参照ページ]

41. 変更がありません。

[ツール] トップ・ウインドウ (ソース変更) または辞書エディタ

[メッセージの意味] 辞書の変更において、現在まで使われていた辞書と全く同じ辞書が変更後の辞書として選択されたので、辞書の変更は行わない。

[ユーザの対応] なし。

[関連参照ページ]

42. 未定義のマトリクスを使用しています : xxx
xxx は、マトリクス。

[ツール] トップ・ウインドウ (トランスレータ) または辞書エディタ

[メッセージの意味] ソース形式辞書ファイルの宣言部で定義されたマトリクス以外のものがマトリクスとして使用されている。

[ユーザの対応] ソース形式辞書ファイルを修正後、再度トランレータを起動する。

[関連参照ページ]

43. メニュー・ウインドウを開いて下さい

[ツール] トップ・ウインドウ

[メッセージの意味] LAX トップ・ウインドウを作成して下さい。

[ユーザの対応] マウス・マーカを本ウインドウから外し、LAX トップ・ウインドウを作成する。

[関連参照ページ]

44. モード変更を中止します。

[ツール] トップ・ウインドウ (モード変更)

[メッセージの意味] 解析モードをインタプリタ・モードからコンパイラ・モードに変更しようとしたが、指定された辞書のオブジェクト・プログラム (解析エンジン) がカタログ、コンパイルされていない。

[ユーザの対応] オブジェクト・プログラム (なければ、中間形式辞書ファイルから作成する) をカタログ、コンパイル後モード変更を行う。

[関連参照ページ]

2.4.2 辞書エディタの出力メッセージ

1. xxx はありません。
   xxx は、形態素。

   [ツール] 辞書エディタ

   [メッセージの意味] 指定された形態素がない。

   [ユーザの対応] なし。

   [関連参照ページ]

2. xxx は二重登録です。
   xxx は、表層。

   [ツール] 辞書エディタ

   [メッセージの意味] 追加しようとした形態素は、既に中間形式辞書ファイルに登録済みである。または、カテゴリ、接続素性の変更の結果、既に中間形式辞書ファイルに登録済みの形態素と一致した。

   [ユーザの対応] カテゴリ、表層、左方または右方形態素のどれかを変える。

   [関連参照ページ]

3. xxx を書き換えました。
   xxx は、表層。

   [ツール] 辞書エディタ

   [メッセージの意味] 形態素 xxx のカテゴリまたは左方ないし右方接続素性の書き換えがなされた。

   [ユーザの対応] なし。

   [関連参照ページ]

4. xxx を削除しました。
   xxx は、表層。

   [ツール] 辞書エディタ

   [メッセージの意味] 形態素 xxx が、中間形式辞書ファイルから削除された。

   [ユーザの対応] なし。

   [関連参照ページ]

5. xxx を追加しました。
   xxx は、表層。

   [ツール] 辞書エディタ

   [メッセージの意味] 形態素 xxx が追加された。

   [ユーザの対応] なし。

   [関連参照ページ]

6. エディタ・ウインドウを開いて下さい。

   [ツール] 辞書エディタ

   [メッセージの意味] 辞書エディタのウインドウを作成して下さい。

   [ユーザの対応] 辞書エディタのウインドウを作成する。

   [関連参照ページ]

7. エディタ準備中です。しばらくお待ち下さい。

   [ツール] 辞書エディタ

[メッセージの意味] 辞書エディタの初期設定中。

[ユーザの対応] しばらくすると、本メッセージが表示されているウインドウが消え、辞書エディタのウインドウが表示される。

[関連参照ページ]

8. エディタを中止します。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の初期設定 (LAX トップ・ウインドウの項目 "新規" 選択時) において、項目 "ABORT" を選択したので、辞書エディタが終了した。

[ユーザの対応] なし。

[関連参照ページ]

9. 同じソース名があります。別のソース名を指定して下さい。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の初期設定において、指定されたファイル名を持つ中間形式辞書ファイルが既に存在する。

[ユーザの対応] ソース名を変えて、エディタ・コマンド・ウインドウの項目 "DO IT" を選択する。

[関連参照ページ]

10. 書き換え対象エントリを検索により指定して下さい。

[ツール] 辞書エディタ

[メッセージの意味] エディタ・コマンド・ウインドウの項目 "キー検索" が選択された時表示される。意味は、題意の通り。

[ユーザの対応] 検索を行って、カテゴリまたは接続素性の書き換えを行う形態素をホルダに登録する。

[関連参照ページ]

11. カテゴリ xxx に属する形態素はありません。

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] カテゴリ xxx を持つ形態素は 1 件もない。辞書エディタで、カテゴリ xxx を持つ形態素が全て削除された。

[ユーザの対応] なし。

[関連参照ページ]

12. カテゴリ xxx を編集します (n件)
xxx はカテゴリ、nはカテゴリ xxx を持つ形態素の個数。

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] カテゴリ xxx を持つn個の形態素が、ソース形式辞書ファイルに書き込まれる。

[ユーザの対応] なし。

[関連参照ページ]

13. カテゴリ、表層、左方 / 右方形態素のどれかに誤りがあります。

[ツール] 辞書エディタ

[メッセージの意味] 題意のとおり。

[ユーザの対応] 修正する。

[関連参照ページ]

14. カテゴリ、表層、左方 / 右方素性のどれかに指定がありません。

[ツール] 辞書エディタ

[メッセージの意味] 形態素の追加に置いて、カテゴリ、表層、左方または右方形態素のどれかが指定されていない。

[ユーザの対応] 形態素の追加に置いては、カテゴリ、表層、左方または右方形態素の全てを指定しなければならない。

[関連参照ページ]

15. カテゴリ置換の方法を選択して下さい。　一括 (a), バッファ表示 (b), メニュー選択 (m), キャンセル (c) ?

[ツール] 辞書エディタ

[メッセージの意味] カテゴリ書き換え方法の問い合わせである。

[ユーザの対応] ホルダに登録されている形態素のカテゴリを全て置換する場合は a を、エディタ・インタプリタ・ウインドウに逐次表示して指示する場合は b を、表層の一覧を表示して置換対象形態素を絞り込む場合は m を、カテゴリ置換を取り止める場合は c を入力する。

[関連参照ページ]

16. カテゴリの長さが長すぎます。(xxx)

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウで指定されたカテゴリの長さが辞書環境項目の“カテゴリの最大長”に定義された値を越える。

[ユーザの対応] 以下のどちらかを行う。

- エディタ・インタプリタ・ウインドウで指定したカテゴリを修正する (短くする)。
- 辞書環境項目の“カテゴリの最大長”に定義された値を大きくする。

[関連参照ページ]

17. カテゴリを入力して下さい。 :

[ツール] 辞書エディタ

[メッセージの意味] カテゴリ置換において、置換後のカテゴリの問い合わせである。

[ユーザの対応] カテゴリを入力する。

[関連参照ページ]

18. –既存ファイルをクローズしています–

[ツール] 辞書エディタ

[メッセージの意味] オープン中の中間形式辞書ファイルをクローズ中。

[ユーザの対応] なし。

[関連参照ページ]

19. ホルダは空です。

[ツール] 辞書エディタ

[メッセージの意味] ホルダが空にも関わらず、ホルダ選択を行おうとした。

[ユーザの対応] 検索によって、ホルダに形態素を登録する。

[関連参照ページ]

20. ホルダをスタックに追加しました。

[ツール] 辞書エディタ

[メッセージの意味] 題意のとおり。

[ユーザの対応] なし。

[関連参照ページ]

21. ホルダを取り出しました。

[ツール] 辞書エディタ

[メッセージの意味] ホルダ・スタックからホルダを取り出した。

[ユーザの対応] なし。

[関連参照ページ]

22. クローズしました。

[ツール] トップ・ウインドウまたは辞書エディタ

[メッセージの意味] 中間形式辞書ファイルのクローズを終了した。

[ユーザの対応] なし。

[関連参照ページ]

23. 削除方法を選択して下さい。　一括 (a), バッファ表示 (b), メニュー表示 (m), キャンセル (c) ?

[ツール] 辞書エディタ

[メッセージの意味] 削除方法の問い合わせである。

[ユーザの対応] ホルダに登録されている形態素を全て削除する場合はaを、エディタ・インタプリタ・ウインドウに逐次表示して取捨選択する場合はbを、表層の一覧を表示して削除対象形態素を絞り込む場合はmを、削除を取り止める場合はcを入力する。

[関連参照ページ]

24. 削除を終了しました。

[ツール] 辞書エディタ

[メッセージの意味] 形態素の削除処理が終了した。

[ユーザの対応] なし。

[関連参照ページ]

25. 辞書逆変換を開始しました

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 逆トランスレータが起動された。

[ユーザの対応] なし。

[関連参照ページ]

26. 辞書逆変換に失敗しました

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 逆トランスレータが正常に終了しなかった。

[ユーザの対応] 中間形式辞書ファイル、ソース形式辞書ファイルが共に壊れている可能性があるので、バックアップ・ファイルがあればコピーして復旧する。

[関連参照ページ]

27. 辞書逆変換を終了しました

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 逆トランスレータが正常に終了し、中間形式辞書ファイルからソース形式辞書ファイルが作成された。

[ユーザの対応] なし。

[関連参照ページ]

28. 指定されたエントリはありません。

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウで指定された形態素は存在しない (削除、置換時)。

[ユーザの対応] なし。

[関連参照ページ]

29. 出力パス名:xxx.lax　　xxx は、ソース形式辞書ファイルのパス名 (ファイルの拡張子を除く)

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 作成されるソース形式辞書ファイルのパス名を問い合わせている。

[ユーザの対応] ソース形式辞書ファイルのパス名が xxx.lax でよければ<CR>を押下する。変えたければ変更後<CR>を押下する。

[関連参照ページ]

30. スタックが空です。

[ツール] 辞書エディタ

[メッセージの意味] ホルダ・スタックが空にも関わらず、ホルダを取り出そうとした。

[ユーザの対応] なし。

[関連参照ページ]

31. 接続素性が多すぎます。(マトリクス : xxx)

[ツール] 辞書エディタ

[メッセージの意味] マトリクス xxx に属する接続素性の個数が、辞書環境項目の“素性数 /1 マトリクス”で定義された値を越えた。

[ユーザの対応] 以下のどちらかを行う。

- エディタ・インタプリタ・ウインドウで指定した左方または右方接続素性を既存のものに変える。
- 辞書環境項目の“素性数 /1 マトリクス”に定義された値を大きくする。

[関連参照ページ]

32. 接続素性の長さが長すぎます。(素性 : xxx)

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウで指定された左方または右方接続素性の長さが辞書環境項目の“接続素性の最大長”に定義された値を越える。

[ユーザの対応] 以下のどちらかを行う。

- エディタ・インタプリタ・ウインドウで指定した左方または右方接続素性を修正する (短くする)。
- 辞書環境項目の“接続素性の最大長”に定義された値を大きくする。

[関連参照ページ]

33. 設定に誤りがあります。　　もう一度設定しなおして下さい。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の初期設定または変更において、辞書環境項目の設定値に誤りがある。

[ユーザの対応] 修正して、エディタ・コマンド・ウインドウの項目 "DO IT" を選択する。

[関連参照ページ]

34. 設定に失敗しました。もとに戻します。

[ツール] トップ・ウインドウまたは辞書エディタ (ソース変更)

[メッセージの意味] 辞書の変更ができなかったので、カレント辞書を変更前の辞書に戻した。

[ユーザの対応] 辞書の変更に失敗したのは、何らかの理由で変更後の中間形式辞書ファイルが壊れているためなので、バックアップ・ファイルがあればコピーすることにより復旧する。

[関連参照ページ]

35. 設定を終了しました。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の初期設定または変更が終了した。

[ユーザの対応] なし。

[関連参照ページ]

36. ソース xxx を設定しました。

[ツール] トップ・ウインドウ (ソース変更) または辞書エディタ (ソース変更)

[メッセージの意味] カレント辞書を辞書 xxx にした。

[ユーザの対応] なし。

[関連参照ページ]

37. ソース・ファイルを作成しますか?[y/n]

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の項目値の変更に伴って、ソース形式辞書ファイルを作り直すかどうかの問い合わせである。

[ユーザの対応] ソース形式辞書ファイルを作り直す場合は y を、作り直さない場合は n を入力する。辞書環境の変更をやり直す場合は M−g を入力する。

[関連参照ページ]

38. ソース名を指定して下さい。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の初期設定において、ソース名の指定がない。

[ユーザの対応] ソース名を設定して、エディタ・コマンド・ウインドウの項目 "DO IT" を選択する。

[関連参照ページ]

39. 対象エントリは xx 件あります。

[ツール] 辞書エディタ

[メッセージの意味] 検索条件に合う形態素は xx 個あり、ホルダに登録された。

[ユーザの対応] なし。

[関連参照ページ]

40. 対象エントリはありません。

[ツール] 辞書エディタ

[メッセージの意味] 検索条件に合致する形態素はない。

[ユーザの対応] なし。

[関連参照ページ]

41. !!注意!! 辞書のクラス数を以下のように見なします。

[ツール] トップ・ウインドウ (トランスレータ) または辞書エディタ

[メッセージの意味] ソース形式辞書ファイルの宣言部で定義されている辞書のクラス数を、本メッセージの次に表示される数に置き換えた。ユーザが定義したクラス数をnとすると、nより大きくかつ最も近い2のべき乗に置き換えられる。

[ユーザの対応] なし。

[関連参照ページ]

42. 中間ファイルをクローズします。

[ツール] トップ・ウインドウ (ソース変更) または辞書エディタ

[メッセージの意味] 辞書の変更にともない現在まで使われていた辞書をクローズする。

[ユーザの対応] なし。

[関連参照ページ]

43. 中間辞書を読み込んでいます

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] 中間形式辞書ファイルの内容を、主記憶上に読み込んでいる。

[ユーザの対応] しばらく待つと、本メッセージが表示されているテンポラリ・ウインドウが消える。

[関連参照ページ]

44. 中間ファイルをもったソースを指定して下さい

[ツール] トップ・ウインドウ (逆トランスレートまたはオブジェクト作成)

[メッセージの意味] 指定された中間形式辞書ファイルが存在しない。

[ユーザの対応]

- 逆トランスレートは実行できない。
- オブジェクト作成の場合、LAXトップ・ウインドウの項目“トランスレート”を選択して、中間形式辞書ファイルを作成後、オブジェクト作成を行うか、LAXトップ・ウインドウの項目“トランス&カタログ”を選択して、中間形式辞書ファイルの作成とオブジェクトの作成を行う。

[関連参照ページ]

45. 追加するホルダがありません。

[ツール] 辞書エディタ

[メッセージの意味] ホルダが空にも関わらず、ホルダ・スタックに登録しようとした。

[ユーザの対応] なし。

[関連参照ページ]

46. トークンの長さが長すぎます。(xxx)

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウで指定されたトークンの長さが辞書環境項目の“表層の最大長”に定義された値を越える。

[ユーザの対応] 以下のどちらかを行う。

- エディタ・インタプリタ・ウインドウで指定したトークンを修正する (短くする)。

- 辞書環境項目の“表層の最大長”に定義された値を大きくする。

[関連参照ページ]

47. パス名が見つかりません。

[ツール] トップ・ウインドウ (逆トランスレータ) または辞書エディタ

[メッセージの意味] ソース形式辞書ファイルのパス名のうち、ディレクトリが正しくない (存在しない)。

[ユーザの対応] 再度逆トランスレータを起動して、既存のディレクトリを指定する。または、SIMPOS のサブ・システム、ファイル・マニピュレータでディレクトリを作成してもよい。

[関連参照ページ]

48. バッファに表示された形態素についての指示を選択して下さい。　削除する (y), 削除しない (n), 中止する (c) ?

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウに表示されている形態素を削除するかどうかの問い合わせである。

[ユーザの対応] 削除する場合は y を、削除しない場合は n を、削除を終了する場合は c を入力する。

[関連参照ページ]

49. バッファに表示された形態素についての指示を選択して下さい。　選択する (y), しない (n), 終了 (e) ?

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウに表示されている形態素をホルダに登録するかどうかの問い合わせである。

[ユーザの対応] ホルダに登録する場合は y を、登録しない場合は n を、ホルダ選択を終了する場合は e を入力する。

[関連参照ページ]

50. バッファに表示された形態素についての指示を選択して下さい。　置換する (y), 置換しない (n), 中止する (c) ?

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウに表示されている形態素のカテゴリを書き換えるかどうかの問い合わせである。

[ユーザの対応] 書き換える場合は y を、書き換えない場合は n を、カテゴリ置換を終了する場合は c を入力する。

[関連参照ページ]

51. バッファに表示された形態素についての指示をマウスで選択して下さい。

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウに表示されている形態素の素性を書き換えるかどうかの問い合わせである。

[ユーザの対応]

書き換える場合は、エディタ・インタプリタ・ウインドウに表示されている形態素を修正後、エディタ・コマンド・ウインドウの項目“DO IT”を、書き換えないでホルダに登録されている次の形態素を表示する場合は“SKIP”を、素性の書き換えを終了する場合は“ABORT”をマウスで選択する。

[関連参照ページ]

52. 表層の長さが長すぎます。(xxx)

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウで指定された表層の長さが辞書環境項目の“表層の最大長”に定義された値を越える。

[ユーザの対応] 以下のどちらかを行う。

- エディタ・インタプリタ・ウインドウで指定した表層を修正する(短くする)。
- 辞書環境項目の“表層の最大長”に定義された値を大きくする。

[関連参照ページ]

53. ファイルが壊れています。ファイル名 : xxx

[ツール] 辞書エディタ

[メッセージの意味] 中間形式辞書ファイルが壊れている。

[ユーザの対応] 中間形式辞書ファイルのバックアップ・ファイルがあれば、コピーして復旧する。

[関連参照ページ]

54. ファイルをクローズしています。

[ツール] トップ・ウインドウ(トランスレータ)

[メッセージの意味] トランスレータ実行のため、中間形式辞書ファイルをクローズする。

[ユーザの対応] なし。

[関連参照ページ]

55. ファイルをクローズします。

[ツール] トップ・ウインドウまたは辞書エディタ

[メッセージの意味] 中間形式辞書ファイルをクローズする。

[ユーザの対応] なし。

[関連参照ページ]

56. ファイルを再作成します。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の項目値が変更されたので、中間形式辞書ファイルを作り直す。

[ユーザの対応] なし。

[関連参照ページ]

57. 文法の誤りです

[ツール] 辞書エディタ

[メッセージの意味] エディタ・インタプリタ・ウインドウで指定されたカテゴリ、表層、左方または右方接続素性に誤りがある。

[ユーザの対応] 修正する。

[関連参照ページ]

58. 変更ありません。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の変更において、辞書環境項目の値を変えないでエディタ・コマンド・ウインドウの項目“DO IT”が選択されたので、変更は行わない。

[ユーザの対応] なし。

[関連参照ページ]

59. 変更がありません。

[ツール] トップ・ウインドウ (ソース変更) または辞書エディタ

[メッセージの意味] 辞書の変更において、現在まで使われていた辞書と全く同じ辞書が変更後の辞書として選択されたので、辞書の変更は行わない。

[ユーザの対応] なし。

[関連参照ページ]

60. 変更しました。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の変更が終了した。

[ユーザの対応] なし。

[関連参照ページ]

61. 変更できません。　　もう一度値を設定して下さい。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境項目の変更をし直して下さい。

[ユーザの対応] 値を修正してエディタ・コマンド・ウインドウの項目 "DO IT" を選択する。

[関連参照ページ]

62. 変更を終了しました。

[ツール] 辞書エディタ

[メッセージの意味] 形態素のカテゴリまたは接続素性の書き換え処理を終了した。

[ユーザの対応] なし。

[関連参照ページ]

63. マトリクスを一つ以上設定して下さい。

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境の初期設定において、マトリクスの指定がない。

[ユーザの対応] マトリクスを 1 つ以上指定して、エディタ・コマンド・ウインドウの項目 "DO IT" を選択する。

[関連参照ページ]

64. 未定義のマトリクスを使用しています。(マトリクス : xxx)

[ツール] 辞書エディタ

[メッセージの意味] 辞書環境項目で定義されていないマトリクスをエディタ・インタプリタ・ウインドウで指定した。

[ユーザの対応] エディタ・インタプリタ・ウインドウのマトリクスを修正する。

[関連参照ページ]

2.4.3 インスペクタの出力メッセージ

1. ILLEGAL INPUT

   [ツール] インスペクタ

   [メッセージの意味] 不正なマウス・クリックがなされた。マウスの位置が正しくないか、マウス・ボタンが正しくないか、クリック方法が正しくない。

   [ユーザの対応] マウス・クリックし直す。

   [関連参照ページ]

2. 解析できませんでした

   [ツール] インスペクタ

   [メッセージの意味] 解析エンジンが入力文の解析に失敗した。

   [ユーザの対応] 辞書中の関連形態素を修正する。

   [関連参照ページ]

3. ホルダに追加しました

   [ツール] インスペクタ

   [メッセージの意味] 形態素が辞書エディタに渡されるホルダに登録された。

   [ユーザの対応] なし。

   [関連参照ページ]

4. 接続形態素はありません

   [ツール] インスペクタ

   [メッセージの意味] 左方または右方形態素接続情報表示ウインドウで、マウスで選択された形態素はその右方または左方の形態素と接続していない。

   [ユーザの対応] なし。

   [関連参照ページ]

5. ログを取得しました

   [ツール] インスペクタ

   [メッセージの意味] 解析木表示ウインドウに表示されている情報をエディタ・テキスト・バッファに書き込んだ。

   [ユーザの対応] SIMPOS のサブ・システム Pmacs を使って、この情報を見ることができる。

   [関連参照ページ]

2.4.4 プリティ・プリンタの出力メッセージ

1. ウインドウを開いて下さい。

   [ツール] トップ・ウインドウ (プリティ・プリンタの起動)

   [メッセージの意味] プリティ・プリンタのウインドウを作成して下さい。

   [ユーザの対応] プリティ・プリンタのウインドウを作成する。

   [関連参照ページ]

2.4.5 センテンス・セレクタの出力メッセージ

1. テキスト・ファイルが見つかりません

   [ツール] センテンス・セレクタ

   [メッセージの意味] 指定された入力テキスト・ファイルが存在しない。

   [ユーザの対応] 以下に示すいずれかを行う。

   - 文選択属性変更ウインドウが表示されるので、入力ファイル名を変更する。
   - 文選択変更ウインドウの項目 "abort" を選択した後、入力テキスト・ファイルにテキストを登録 (Pmacs を使う) して、項目 " 文選択 " を再度選択する。
   - 文選択変更ウインドウの項目 "abort" を選択した後、 項目 " 文入力 " を選択する。

   [関連参照ページ]

2. 登録文がありません

   [ツール] センテンス・セレクタ

   [メッセージの意味] 指定された入力テキスト・ファイルにテキストが登録されていない。

   [ユーザの対応] 以下に示すいずれかを行う。

   - 文選択属性変更ウインドウが表示されるので、入力ファイル名を変更する。
   - 文選択変更ウインドウの項目 "abort" を選択した後、入力テキスト・ファイルにテキストを登録 (Pmacs を使う) して、項目 " 文選択 " を再度選択する。
   - 文選択変更ウインドウの項目 "abort" を選択した後、 項目 " 文入力 " を選択する。

   [関連参照ページ]

3. - 文章を読み込んでいます -

   [ツール] センテンス・セレクタ

   [メッセージの意味] 入力テキスト・ファイルからテキストを読み込み中。

   [ユーザの対応] しばらくすると、本メッセージが表示されたウインドウが消え、文選択ウインドウが表示される。

   [関連参照ページ]

第 3 章

LAX の文法

本章では、

- LAX の辞書作成手順
- 形態素解析の基本的な文法

について述べる。

3.1 LAX での辞書作成手順

3.1.1 LAX の辞書

辞書の形式

ユーザは LAX に複数の辞書を登録できる。各々の辞書は実際には 3 つの形式のファイルからなる。

- ソース形式辞書ファイル
  3.1.3に示す記述形式をもつファイルである。ユーザは SIMPOS のサブシステム Pmacs を使って本ファイルを作成する。本ファイルをトランスレートすると中間形式辞書ファイルが作成される。また、中間形式辞書ファイルを逆トランスレートすると本ファイルが得られる。

- 中間形式辞書ファイル
  辞書エディタを使って辞書を更新する時、その処理対象となるファイルである。また、インスペクタがインタプリタ・モードで動作する時、本ファイルを参照する。ユーザは、本ファイルの記述形式を意識する必要はない。

- オブジェクト形式辞書ファイル
  中間形式辞書ファイルをジェネレートすると作成されるファイルであり、実態は ESP ソース・プログラムである。このとき、一つの辞書はユーザが指定した数 (ソース形式辞書ファイルの宣言部の `classes` 記述時 (3.1.3参照) または、辞書環境の初期設定 / 変更時 (2.3.1参照) に指定する) のクラス群に分割される。本ファイルに含まれているクラス群をコンパイルしたものが、解析エンジンの一部として動作する (コンパイラ・モード)。ユーザは、本ファイルの記述形式を意識する必要はない。

辞書のファイル名

辞書はすべて SIMPOS が管理するファイルとして実現される。辞書のパス名は、

論理パス名 > ディレクトリ名 > 辞書名 > 拡張子名

という形式をしている。これらのうち、ユーザが指定できるのは辞書名のみである[1]。
論理パス名は、LAX のインストール時に作成され `LAX` と決まっている。ディレクトリ名、拡張子名については以下に示す。

	ディレクトリ名	拡張子名
ソース形式辞書ファイル	SOURCE	lax
中間形式辞書ファイル	MDIF	imf 等
オブジェクト形式辞書ファイル	OBJECT	obj

3.1.2 辞書作成手順

LAX での辞書の作成目的は、最終的に形態素解析プログラムおよび意味構成プログラムを得ることにある。現在の LAX では、いずれも中間形式辞書と呼ばれる辞書からジェネレータを用いて生成している。従って、ここでの問題はいかに中間形式辞書を作成するかに言い換えることができる。

中間形式辞書の作成手順は、2 つある。一つは LAX 辞書からトランスレータを用いて一度に作る方法、二つ目は辞書エディタを用いて一つ一つエントリを入力していく方法である。これらの方法の得失を以下に掲げる。

	方法 1 (トランスレータ)	方法 2 (辞書エディタ)
利点	・一度に多量の辞書データ	・個々のデータの修正が

[1] 逆トランスレートでは、出力ファイルであるソース形式辞書ファイルのパス名を指定できる

	を作成できる ・可読性がある	容易 ・可読性がない
欠点	・個々のデータの変更に時間がかかる	・多量のデータの作成には不向き

このように、これらの方法は相互補完的になっており、辞書開発の規模やフェーズに合せて方法を選択する必要がある。数十語の辞書を実験的に作るのであれば、初めから辞書エディタのみを使用して作成するので十分である。辞書エディタにより新規の辞書を作成する方法については、 2.3.1を参照のこと。

一方、数千語の辞書を作成するのであれば、最初にLAX 辞書を作成しそれをトランスレートするほうが良い。なぜなら、多量のエントリの辞書化には、形態素のタイプの分析と、対象形態素がどのタイプに属するかの分類作業が必要であり、LAX 辞書はその記述に適した単純マクロを用意しているからである。具体的には、辞書記述形式 (3.1.3) を参照のこと。

ローカルな記述の変更や新たなエントリの追加等を辞書エディタを用いて行なう。そうすると初めに作成したLAX 辞書と内容がだんだんずれていくので、切りの良いところで中間形式辞書から逆トランスレートによって最新のLAX 辞書を得ておくことが望ましい。この逆変換作業は中間形式辞書のバックアップをとるという側面と、記述内容の一覧を得るという側面を合わせもつ。この方法の唯一の欠点は、元々のLAX 辞書に書かれていたコメントが逆変換によって失われてしまう点である。

3.1.3 形態素意味辞書の記述形式

LAX 辞書記述形式について説明する。LAX で記述できる文法は3型の文法であり、特に制限はないが、シンタクスが若干特殊である。これは特に森岡文法のインプリメントが容易になるように工夫されているためであるが、他の文法での記述の妨げになることはない。はじめに文法記述形式の BNF 仕様をあたえる。

```
LAX 形態素文法 ::=  宣言部
                        begin(spec).
                        定義部
                        end(spec).
宣言部 ::= conn_matrix( マトリクス・リスト ).
           classes(べき乗数).
           word_length( 整数 ).
           category_length( 整数 ).
           feature_length( 整数 ).
           feature_number( 32 倍数 ).
         { include( ファイル名 ). }
マトリクス・リスト ::= [ マトリクス名 {, マトリクス名} ]
べき乗数 ::= 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256
32 倍数 ::= 32 | 64 | 96 | 128 | 160 | 192 | 224
定義部 ::= カテゴリ記述 { カテゴリ記述 }
カテゴリ記述 ::= begin(カテゴリ名).
                   形態素記述 { 形態素記述 }
                   end(カテゴリ名).
形態素記述 ::= 形態素 { (トークン) } ::    接続情報部
         {  $$   意味情報部 }.
接続情報部 ::=   接続情報
               | [ 接続情報 {, 接続情報 } ]
       | same
意味構成部 ::=   [ 変数宣言部 { , クローズ } ]
```

```
        | same
変数宣言部 ::= [ 予約語 { , 予約語 } ]
予約語 ::= []  | in( 変数 ) | out( 変数 ) | morph( 変数, 変数 )
              | lfea( 変数, 変数 ) | rfea( 変数, 変数 )
クローズ ::= ESP クローズ
変数 ::= ESP 変数
接続情報 ::= マトリクス名 ( 接続素性リスト )
接続素性リスト ::= [ 接続素性名 {, 接続素性名 } ]
マトリクス名 ::= ESP アトム
ファイル名 ::= ESP ストリング
接続素性名 ::= ESP アトム
形態素 ::= ESP アトム
トークン ::= ESP アトム
```

ここで、ESP アトム、ストリングとはそれぞれ ESP の言語仕様におけるアトム、16 ビット・ストリングである。また、include で取り込まれるファイルの形式は以下の通りである。

```
インクルード・ファイル形式 ::= begin(include).
                                定義部
                                end(include).
```

インクルード宣言は、宣言部にただ一つ許されるだけである。これは、このインクルード宣言の用途を付属語辞書等の汎用の辞書のインクルードにしか適用しないことを前提としているためである。

記述は大きく宣言部と形態素定義部とに分かれる。宣言部ではトランスレータが中間形式辞書を生成する際に必要な諸情報の宣言を行なう。宣言の内容を以下に説明する。

- conn_matrix:
  辞書で使用する状態遷移表名を宣言する。宣言の順序は意味を持たない。
- classes:
  辞書が大きくなると、解析プログラムの辞書モジュールが 1 クラスに収まり切れな い。そのような場合に幾つのクラスに分割するかを指定する。指定値は 256 まで の 2 のべき乗数である。ちなみに 1 クラスにはいるエントリ数は 800 〜 1000 語が目安である。
- word_length:
  形態素を構成する文字数の上限を設定する。
- category_length:
  カテゴリ名を構成する文字数の上限を設定する。
- feature_length:
  接続素性を構成する文字数の上限を設定する。
- feature_number:
  接続素性の個数の上限を設定する。224 までの 32 倍数を設定できる。

end マトリクスについて

ここで LAX で形態素解析を行なう上で非常に重要なマトリクスである end マトリクスについて説明する。end マトリクスはシステムがデフォルトで持っているマトリクスで、語または文の切れ目を認識するためにある。従って、左方に end を持つ形態素は語 (または文) の先頭に来得ることを表し、右方に end をもつ形態素は語 (または文) の切れ目になり得る。また、文頭と文末は仮想的に以下の定義がなされているものと考えてよい。

```
<文頭> ::  don't care
       && [ end([文節, 文頭]) ]
       $$ [ [out(_)] ].

<文末> ::  end([文節, 文末])
       &&  don't care
       $$ [ [in(SAX),out(SAX)] ].
```

LAX では、語へのまとめの作業をすべてこの end マトリクスでの状態遷移時に行なう。すなわち、形態素解析時に end マトリクス同士で繋げられた2つの形態素は、意味的にはそこで語が切れているということである。

形態素意味記述

辞書記述の中心は「形態素記述」の部分である。ここは、形態的な接続情報を記述する接続情報部と、意味構成プログラムの呼出し形式を決める意味構成部とに分けられる。いわゆる形態素解析(語の切りだし)のみを行なう場合は、前者を記述するだけでよい。また、記述が直前の形態素の定義と同じ場合は same と書くだけでよい。このシンタックス・シュガーは3通りの場合に分類でき、そのおのおのの意味は以下の通りである。

1. 自然 :: same.
   直前の記述の接続情報と同じ情報を有する。直前の記述が意味構成部を持てば、その 内容も同じである。

2. 自然 :: ...
   && ... $$ same. 意味構成部が直前の記述と同じ内容を有する。

3. 自然 :: same
   $$ [[...],...]. 接続情報が直前の記述と同じ内容を有する。

これらの形態素記述は、カテゴリ毎に begin と end で一纏めにしておく。カテゴリ名は形態素解析時に必要なものではなく、辞書作成者の形態素分類の目安に過ぎない。しかし辞書エディタでの検索時には威力を発揮する。形態素解析で使用するカテゴリ的なものは左方接続素性が担っている。

意味構成部

意味構成部の記述は、変数宣言部と意味構造の操作を行なうクローズ呼び出し部とに分かれる。クローズ呼び出し部には、決定的に動作するクローズを並べておく。バックトラックはゆるさない。フェイルした時点でその語の意味構成は放棄される。呼びだされたクローズの処理内でバックトラックするのはかまわない。

接続素性部

ここには、形態素の読みに相当する表層と解析結果として持ち上げるトークン、そしてその形態素の左方接続素性と右方接続素性を記述する。従来の接続情報では、左方接続素性集合と右方接続素性集合が別々にあり、接続判定述語はそれぞれの要素を引数に持つようなものであった。LAX では直観的な素性の記述が行ないやすいようにするためと、接続判定述語の簡略化の目的で、左方右方の接続素性集合を一本化した。これにより接続判定述語は、左方接続素性の右方接続素性に対するメンバーシップ述語になる。

辞書記述の指針としては次のようにするとよい。左方接続素性にはカテゴリを再分類するような、すなわち自分自身の ID となるような素性を書く。そして右方接続素性にはその形態素に後接しうる形態素の ID を並べる。従って、ある形態素の接続情報を見るだけで、その形態素が何者でどんな形態素を後ろにとるかが一目で分かり、その点で従来の記述に比べて自然であると考えている。

また、LAX では複数の状態遷移表(マトリクス)を使って接続素性の記述を行なうことができる。これはオートマトンとしてみた場合には、状態変化を単なる素性の遷移としてではなく、状態遷移表間の移動を含めた素性の遷移としてとらえていることになる。遷移表名に形態素の連接に伴う文法的な現象名(派生, 屈折など, 詳しくは森岡の

『語彙の形成』参照) を与えると、自然な形態素の連接規則が記述できる。

例えば、ある形態素間で「屈折」マトリクス内の素性で連接した場合には、その形態素はそこで屈折を起こしたと考える。つまりある形態素の右方素性に記述されたマトリクスを見れば、その形態素が今後どのような文法的ふるまいを起こす可能性があるかが、そのマトリクス内の素性を見れば、どのような形態素に繋がるかが分かる。逆に左方素性に書かれたマトリクスを見れば、その形態素が先行形態素と結び付いてどのような文法的ふるまいを起こす可能性があるかが分かる。左方接続マトリクス内に記述する素性名は、自らを表すものであってもよいし、同一カテゴリ内での分類名であってもよい。

形態素記述の一般形は次のようになる。

```
表層(トークン):: マトリクス名_F1( [ 素性, 素性, .. ] )
&&   [ マトリクス名_B1( [ 素性, 素性, .. ] ),
:
マトリクス名_Bn( [ 素性, 素性, .. ] ) ],
$$   [ [ 変数宣言部 ],
述語呼び出し部 ].
```

「$$」以降が意味構成部に関する情報であり、単に形態素切りを行なうだけならば記述は不必要である。トークンは、表層と同じであれば省略できる。

接続可能条件

直観的には、先行形態素の右方接続素性中に後接形態素の左方接続素性のマトリクスが存在し、かつその中に一致する素性がただ一つ存在する場合に接続が可能であるとする。

左方接続素性の優先順序

左方接続素性については、原則として単一マトリクスのみ記述するのであるが、例外的に複数記述することができる。これは、左方接続素性にプリファレンスをつけることにより不要な解釈を押えることができるからである。具体的な例を挙げて説明する。例えば、数字の接続素性を考えてみる。単一の英数字は文節頭にも文節末にも来うるし、後方にまた数字をとることもできる。これを通常の LAX の辞書記述にすると下記のようになる。

```
1 ::   end( [文節] )              %  文節頭に来る場合
     [ 合成( [数字] ),
       end( [文節] ) ].

1 ::   合成( [数字] )             %  数字に連なる場合
     [ 合成( [数字] ),
       end( [文節] ) ].
           :
```

ところが、これでは数字の連接、例えば「123」を解析すると、正しい解釈である 1・2・3 (『・』は連接を表す) の他に、1/2・3、1・2/3、1/2/3(『/』は文節の切れ目を表す) のような明らかに無意味な解釈も必ず出力されてしまう。この現象の質の悪い所は、連接文字 N 文字に対して曖昧な解釈を 2 の N − 1 乗個だけ発生してしまう点である。したがって、一文中に 5 ケタの数字が 3ヶ所に現われたとするとそれだけでもう 16 ＊ 16 ＊ 16 = 4096 個の曖昧な解釈を抱える事になる。この問題は、「後接するものが数字であれば文節で切れる解釈は発生させない」というヒューリスティクスを導入することで自然に解消できる。つまり、先の辞書記述を次のように改める。

```
1 :: [ 合成( [数字] ), end( [文節] ) ]
     [ 合成( [数字] ),
       end( [文節] ) ].
```

この時の先行形態素との接続可能性のテストは、左方接続素性定義中の初めの方のマトリクス定義のものから行ない、成功した時点でそれ以後の候補を捨ててしまうようにすればよい。つまり連接可能性の高いものを左から順に書いておくことにより、接続試験にプライオリティをつけられるようになっている。

意味構成部に関する補足事項

意味記述部の予約語について　現在、下記の5種の予約語が用意されている。

in(In)	: 左方形態素から流れてきたデータを受け取る。
out(Out)	: 右方へデータを流す．語の終わりには語の意味を返す。
morph(Tok,Cat)	: 定義中の形態素のトークンとカテゴリを返す。
lfea(Mat,Fea)	: 左方形態素と接続したマトリクスと素性を返す。
rfea(Mat,Fea)	: 右方形態素と接続したマトリクスと素性を返す。

SAX 入力データの出力　out(Out) によって出力変数宣言される変数には、語の構成途中であれば後方へ流すデータ構造を、語の終了時であれば SAX 入力データの形式にした語の意味構造をバインドしなければならない。SAX 入力データは、下記のようなリストの形態をとる。複数有るというのは同一文節において複数の意味解釈が生じたことを意味している。

```
       Out=[ lex_word([Data]) ]
または  Out=[ lex_word([Data]), lex_word([Data]), .. ]
```

3.2 文節の構成 (形態素解析)

3.2.1 はじめに

形態素解析は形態素を処理単位として扱う。構文の解析に適する語を基本単位とはしない。従って動詞の活用形処理や名詞の辞書検索といわれる処理はない。解析によって得られる結果は語であり、一般には文節と呼ばれる。以下では、形態素解析の基本的な文法について説明する。

3.2.2 文節構成の基本的枠組み

日本語の用言 (動詞, 形容詞, 形容動詞) の活用及び、体言・用言に後接する助辞 (助詞, 助動詞) 結合の合文法性を検査するアクセプタは、3型PSG(有限状態文法) で記述できると言われている。

[3型PSG]	$\alpha \rightarrow \beta$	
	$\alpha \rightarrow \beta\gamma$	(α, γ は非末端語, β は末端語)

図 3.1: 3型PSG

構文解析ではない、文節構成のための形態素解析においては、直接の解析入力の対象は文であっても、解析出力の対象は各文節であり、形態素解析の目的は、各文節の構成要素としての形態素の認定と、各形態素の接続関係の認定である。また、各文節間の関係は問題とされない。従って、記述すべき形態素解析の文法は、基本的には用言の活用、及び助辞結合であり、3型PSG で記述できることになる。
ただし、これとは別に形態素解析を漏れ無く、かつ、合文法的 (文節) の判定をすることができるかという問題がある (形態素解析に限らず、言語の算法化自体に関わる問題ではあるが)。形態素解析 (文節構成の) に限って考えてみても完全に解析できるものはまだないのが現状である。
形態素解析の記述には、解析用と生成用の2通りが考えられる。解析用、特に与えられた合文法的な文のみを解析するだけであれば、記述は簡単で、接続可能な組み合わせのマトリクスを1つ用意すればよい。この場合、接続可能な形態素の組だけに注目すれば良いので、各語基ごとに多数の接辞を調査することが容易である。これに対して非文法的な文節をチェックするには最低、形態素の可能な連接の順序を考慮しなければならない。この段階では、接辞はつねに他の接辞との順序関係を持つだけではなく、ある語基から他の語基相当へ派生 (転成 / 準用) した場合に、該当する語基の接辞連接のどの位置に相当するかを決定する必要がある。さらに生成に用いるためには、不自然な表現 (意味的な整合性のチェックを一部含むと考えられる) を避ける必要があるだろう。
解析・生成両用の高い能力を持つ形態素解析を記述するのが理想的ではあるが、ここでは、まず形態素の派生屈折の現象を幅広く記述することに主眼を置くために、基本的に解析の立場に立ち、記述のしやすさのために、CFG 的な記述をする。ただし、これに後から、可能な限り形態素の接続順序を取り入れるという方針で、3型PSG の形に置き換えられるように配慮することとする。

文節の定義 (語基, 接辞, 助辞)

文節構成の概念的な概略は以下の形式になる。

```
1. 文節   → X語基 X助辞
   X語基→ X語基 X接辞 | Y語基 Y接辞
```

図 3.2: 文節構成の概略

ここで、X語基は、体言・用言などの各語基を表し、X助辞は、X語基用の助辞を表すこととする。文節は、

X語基がX語基用のX助辞を取ったものと定義できる。さらに、X語基はX語基用の接辞(X語基への接辞)をとり、再び、X語基相当となる。また、別のY語基がY語基用のY接辞(のうちの、X語基への接辞)をとり、X語基相当となることもある。
助辞により、文節を構成することを屈折、接辞により語基相当を構成することを派生と呼ぶが、各語基によって、派生・屈折の仕方が異なるので、各語基毎に派生・屈折の記述が必要となる。
さて、間投助辞は一般に、各文節の終わりに付くことが出来る。間投助辞の構文的な振舞は、形態的観点から見れば、あらゆる文節末に出現すると簡潔に記述できる。また、文節構成の形態素解析では、文の成立論に触れることは出来ず(あるいは、その必要が無いため)、陳述に関わると言われる終助辞の類は特別な扱いをしない。この間投助辞を含めて、文節を定義すると下図になる。

```
2. 文節   → 文節1   |   文節1 間投助辞
   文節1 → X語基 X助辞
   X語基→ X語基 X接辞   |   Y語基 Y接辞
```

図 3.3: 文節の定義

間投助辞は、各語基とは直接の関係を持たず、出来上がった文節に付く任意の要素として扱っている。
ここまでの定義では、ある語基が文節1を構成するためには、助辞を取らなければならないが、自立する語基は、助辞を取らずに文節を構成することが出来る。これを零屈折と考えることも出来るが、処理のしやすさを考えて、各語基の自立形として扱うことにする。また、別の語基がある形を取り、X語基に派生(=転成)することもある。自立形・派生形は各語基毎に記述される。従って、文節の定義は以下のようになる。

```
3. 文節   → 文節1   |   文節1 間投助辞
   文節1 → X語基自立形   |   X語基 X助辞
   X語基→ X語基 X接辞   |   Y語基 Y接辞   |   Y語基X派生形
```

図 3.4: 文節の定義

文節を構成する語基の設定

さて、Xに相当する各語基であるが、これは森岡の語基に基本的に従う。森岡の語基(和語系)は図3.5に示す13に分類されている。

図3.5の分類では各形態素が自立するか(free form)、しないか(bound form)を第一の分類原理としているが、これは次の点で問題がある。

1. 用言の活用を代表形態素に対する異形態(allomorph)として、後接要素によって起る音韻変化と考えているのに、一方で、純粋に形態的側面からも活用を考えているため、9の用言結合語基が必要になり、活用を含めた諸形態が2の用言と9の用言結合語基に分断されることになる。

   ex.{行き}→用言, {行か-(ない)}→用言語基

2. 10の形容語基は,接辞{-い}を伴って用言としての立場を得るが、この活用は本来の用言(動詞)の場合と扱いが異なる。これは、情態言+{-だ}、体言+{-だ}についても同じである。

   ex.{寒}　→形容語基, [寒-い]　→用言に派生
   {鮮やか}→情態言, [鮮やか-だ]→用言に派生
   {静-(か)}→情態語基, [静-か]　→情態言に派生,

自立語基		1．体言 2．用言 3．情態言 4．副用語 5．感動語
結合語基	派生用	6．体言結合語基 7．数詞語基 8．指示語基 9．用言結合語基 10．形容語基 11．情態語基 12．象徴語基
	屈折用	13．副用語基

図 3.5: 森岡の語基分類

［静－か－だ］ →用言に派生
{彼} →体言, ［彼－だ］ →用言に派生

1、2の問題は活用の扱い及び、用言の語幹認定に関わる問題であり、詳しくは、114頁の「活用語の処理」で論ずることにするが、本来、語基は辞書項目 (lexicon) として扱うべきものであり、これに対して用言の活用は文法として記述すべきものである。文法として記述すべき用言の活用形を9用言語基として建てるところに問題があるといえる。

3. 自立するといわれる体言でも、いわゆる形式名詞のように自立しないものもあり、用言でも自立するのは、ある形態のみである。従って、自立するかしないかは、各語基の問題であるとともに、各形態素の問題でもある。

4. 結合語基のうち、体言結合語基、副用語基などの一部は、派生・屈折の形式が不規則であり、文法として記述する利点が少ない。

そこで、辞書項目 (lexicon) と文法を明確に分離する立場にたち、以下の方針で森岡の語基を再構成することにしたい (図 3.6参照)。

一応、上記の7つに分類し、語基毎の派生・屈折を調査することにするが、形態素の本籍を分類することが目的ではなく、各素性の判定基準を番号順に優先的に分類しただけである。判定基準は、格助辞の付加と、各活用形がそろっているかである。

従って、体言語基イ系とした「赤」は、体言としての資格を持つと共に、形容語基としての資格も持つため、「－い,－さ,－まる,－める」などの接辞を取る。しかし、同じ体言語基イ系でも「四角」は、「－い」のみしかとらない。これらは、結局、形態素ごとに素性を付加するしかない。

活用語の処理について

学校文法の活用表 用言の活用の捉え方により、助辞・接辞の範囲も自動的に決まり、文節構成の規則にも大きな影響を与える。まず、問題点が多く指摘されている学校文法の活用表を図 3.7に示す。

基本的に、助動詞に接続する直前の形態を活用形と呼び、50音図に従って整理したものが学校文法である。変化する形態（語尾）のみに着目すること、古語の活用を基としていることから、助詞・助動詞を多く認める結果になっ

語基の種類	判定基準	例
1．体言語基	格助辞の付加	花，山，文法，ペン (サ変) 心，賞，利，愛，旅行，ミス (ナ型) 味，内気，幸運，健康，フリー (イ系)*赤，*四角，*黄色
2．用言語基	ウ系活用	(ウ系) 行-，食べ-，来-，す-，ず- (サ変) 接-，関-，量-，排-
3．情態語基	ダ系活用	新た-，鮮やか-，静か-，簡単，*同じ (ノ型) 新規，*僅か， (イ系) 暖か，柔らか
4．形容語基	イ系活用	(イ系) 清-，低-，寒-，暑-，悲し- (ナ型) 細か-，可笑し-，大き-
5．副用語基		すぐ，すこし，一応，突然，きらきら
6．数詞語基		ひと-，ふた-，一，二，百，1，2，3
7．指示語基		こ，そ，あ，ど
8．感動詞 接続詞 連体詞		ああ，おや，まあ しかし，だが，ところで，そして いわゆる，ある，さる

（8．感動詞・接続詞・連体詞は，それ自身で，それぞれ文節，文節1，文節1を構成すると考えられるので以下では省略する．*は，かなり変則的なもの）

図 3.6: 再構成した森岡の語基

		語幹	未然形		連用形		終止形	連体形	仮定形	命令形	活用形
			ない	う	ます	て	。	体言	ば	!	接続語
動詞	五段	書	か	こ	き	い	く	く	け	け	
	一段	食	べ		べ		べる	べる	べれ	べろ	
	カ変	φ	こ		き		くる	くる	くれ	こい	
	サ変	φ	し		し		する	する	すれ	しろ	
形容詞		寒	かろ		かっ	く	う	い	い	けれ	φ
形容動詞		静か	だろ		だっ	で	に	だ	な	なら	φ
		語幹	う		た	ない		。	体言	ば	!	接続語
			未然		連用形			終止	連体	仮定形	命令形	活用形

図 3.7: 学校文法の活用表

ている。
問題点としては、語幹・語尾(活用形)の認定が曖昧であり、一段動詞の変化しない「べ」を活用語尾としたり、カ変、サ変動詞では、語幹が無くなること、また、活用形の認定と命名が恣意的であり、各活用形の機能からの命名や、後接する助動詞の意味からの命名などが混ぜんとしていることなどが挙げられる(この他、上の図には示していないが、上一段、下一段活用の区別など現代語として無意味なものが残されているなどの問題点も有る)。
学校文法の問題点は数多く指摘され、多くの文法研究者が独自の活用表を提案しているが、どの活用表も一長一短であり、また、活用をその文法論と切り放しては扱えないために、文法論自体の体系や質の問題もあり、学校文法に代わる活用表が現われていないのが現状である。

現代の活用表　これらの問題点を検討・整理し、現代的な観点から活用の形態的体系をまとめた代表的なものに寺村の活用表(図3.8)がある。寺村は次の方針で活用表を整理している。

1. 活用語尾は単一の形態素であること。

2. 活用語尾たる形態素には、それでその発話が言い切りになるか否かは別 として、それに固有の描叙類型的意味が認められること。

3. 形態的に同一である活用語尾は、構文的機能は違っても、同一の活用語 尾とする。逆に、構文的機能は同一、または似通っていても、形態的に異なる ものは別の活用語尾とし、呼び名の上でも区別する。

この方針でまとめられたものを基に作成した活用表を次のページに示す(動詞は1類から3類に分類されていたが、分かり易くするためと、後の活用表との対応のために強、弱、変格と変えた)。
寺村の活用表は、基本的に各活用形の構文的機能を重視しながらも、形態的にもすっきりしたものになっている。学校文法と比較して、若干の説明を加えると、まず、語幹・語尾をローマ字書きにしたことにより、母音変化として扱われていた活用現象を形態素に分割することが可能となっている。例えば、「行-か, き, く, け, こ」という活用では、語幹{ik}を認定し、語尾として{i}、{u}、{e}、{o"}を認定する。さらに、学校文法で助詞・助動詞とされているものの一部も活用に含め、{cba}、{ta;da}、{taro";daro"}、 {tara;dara}、{tc;dc}、{tari;dari}なども語尾として認められている(学校文法では、完了の助動詞「た」、接続助詞「ば」)。
これらの語幹・語尾が、それぞれ別の形態素と認定されている訳である。また、いわゆる未然形(ア段の)は活用形ではなく、「行かない」は、次のように分析される。

「行かない」の分析
{ik} ：「行く」の語幹
{ana}：否定の意味を持つ形態素
{i} ：形容詞系の活用語尾

「ない」に、強変化動詞に付く{ana}と弱変化動詞に付く{na}の2種を設けることになるが、「(ら)れる」, 「(さ)せる」も同様に処理される。
また、音便形の処理は次のような音韻変化として扱っている。

音便形の音韻変化
{kak}+{ta} → kaita
{kas}+{ta} → kasita
{sin}+{ta} → sinda

音便形の処理は規則的であるために、音韻変化として扱うとしているが、語幹{kak}と、語幹{kai}の両方を認める立場に立つとすると(処理の仕方の違いであって、根本的な原理の違いではない)、この{kak}と{kai}の両者は異形態の関係にあると認められる。
3の基準に従っているものとしては、同形の終止形・連体形の別をなくして、基本語尾としていることが挙げられる。構文的な機能を予定しながらも、形態的にまとまった活用表となっているのは、この 3の基準に従って、活用論は形態論のレベルのものと考えているからである。

		ムード	確言	概言	命令	条件	保留形	
							テ形	タリ形
動詞	基本語尾	強	-u	-o^	-e	-eba	-i	
		弱	-ru	-yo^	-ro	-reba	-φ	
		変格	suru kuru	siyo^ koyo^	siro koi	sureba kureba	si ki	
	タ系語尾	強	-ta -da	-taro^ -daro^	-------- --------	-tara -dara	-te -de	-tari -dari
		弱	-ta	-taro^	--------	-tara	-te	-tari
		変格	sita kita	sitaro^ kitaro^	-------- --------	sitara kitara	site kite	sitari kitari
形容詞	基本		-i	-karo	--------	-kereba	-ku	
	タ系		-katta	-kattaro	--------	-kattara	-kute	-kattari
判定詞	基本	ダ	da	daro^	--------	nara	ni de na no	
		デ	desu	desyo^	--------	--------	----------	
	タ系	ダ	datta	dattaro^	--------	dattara	--------	dattari
		デ	desita	desitaro	--------	desitara	desite	desitari
助動詞	(mas)		-u,-en*	-yo^	-e	-ureba	-,-ende*	
	(masi)		-ta*	-taro^	--------	-tara	-te	-tari
	desu		desu	deshyo^	--------	--------	--------	--------
	desu		desita	--------	--------	--------	--------	--------

*「読まない」，「読まなかった」の丁寧形は「読みません」，「読みませんでした」とするのが普通で，「読まないです」，「読まないでした」は不自然と感じる場合が多いため，助動詞の否定形を活用に含めている．

図 3.8: 寺村の活用表

活用形 / 語幹			(現在) / (完了)	−推量 / −推量	命令	−条件 / −条件	−中立 / −中立	並立	連接形	否定推量
ウ系	強変化	行	-く	-こう	-け	-けば	-き	-ったり	-か	-くまい
			-った	-ったろう		-ったら	-って			
	弱変化	食べ	-る	-よう	-ろ	-れば	-φ	-たり	φ	-るまい
			-た	-たろう		-たら	-て			
	カ変	φ	来る	来よう	来い	来れば	来	来たり	-こ	-こまい
			来た	来たろう		来たら	来て			
	サ変	φ	する	しよう	しろ	すれば	し	したり	-さ	-しまい
			した	したろう		したら	して			
イ系	赤		-い	-かろう	----	-ければ	-く	-かったり	連体	
			-かった	-かったろう		-かったら	-くて			
ダ系	ダ	φ	だ	だろう	----	なら(ば)	で	だったり	な	
			だった	だったろう		だったら だったなら	--------			
	デス	φ	です	でしょう	----	--------	--------	でしたり		
			でした	でしたろう		でしたら	でして			
助動詞	マス	φ	ます	ましょう	ませ	ますれば	--------			
			ました	ましたろう	----	ましたら	まして	ましたり		
	デス	φ	です	でしょう	----	--------	--------			
			でした	----------	----	--------	--------	--------		

否定推量形は，「くるまい，きまい，するまい」の形もある．

図 3.9: 本稿で用いる活用表

1．語基を構成する形	＝	派生
2．文節を構成する形	＝	屈折

図 3.10: 活用の現象の分類

文節構成と活用 図 3.9は、前記寺村の活用表を基にしてカナ書きの方針で作成した、この「文節の構成 (形態素解析)」で用いる活用表である。ここでは、この活用表を用いて、活用を文節構成の上から検討することにする。
文節構成の上からみると、活用の現象は次のように2つに分けることができる (図 3.10)。

1. の派生の機能は、語基 (相当) を構成するもので、「行かない」の「か」が、こ の典型である。活用形により、後接する接辞 / 助辞が異なるが、いずれにせよ、語 基構成の機能を担っている。 (連接形は助辞の側にこれを含ませる寺村の方法を取るのが妥当であるが、ここでは カナ書きという制約上、後方の接辞「ない, せる, れる」に「か」(実際には各行 にわたって、「さ, た…」とある) を含ませると、文法の記述が量的に多くなって しまうという処理上の問題から、本来は派生辞に含まれる形と考えられる「か」を 連接形として扱うことにした。カナ書きによる問題点は、この他に、活用助辞表の カ変、サ変を見ても分かるように、語幹のない、用言語基を認めてしまう点がある。但し、カ変、サ変用言は、事実上、「来る, する」の2語であり、文法内で記述し てしまうために、表面には表われない。サ変漢語系の「旅行する」などは「旅行」 という体言部分がそのまま用言語基の語幹となるという点では問題が無い。)
2. は、文節の定義で既に述べたように文節を構成する形で、「行けば」の「けば」が、この典型である。
図 3.9の活用表は、形態だけに着目して記述してあるが、これに機能を含めて分類すれば、各活用形は、次のように分類できる。

活用	派生辞	派生機能のみ	連接形
	助 辞	派生と屈折	現在形, 完了形, 現在中立形, 完了中立形, 並立形 連体形
		屈折機能のみ	現在推量形, 現在推量形, 命令形, 現在条件形 完了条件形, 否定推量形

図 3.11: 活用形の分類

派生の機能しか持たないものは「派生辞」であり、屈折の機能しか持たないものは「助辞」である。問題は、その両方の機能を持つものである。ここでは、零屈折を認めない立場に立っているので、形態的な同形に「派生辞」と「助辞」の2種が有ることになる。

次に活用の役割を次の例で考えてみよう (図 3.12)。
前述したように、活用形のあるものは、文節を構成することと、連接形と同様に、後接の接辞に接続することの両方の機能を持っている。
1から8までの番号は、ある語基、あるいは、接辞を後接させて語基相当になった段階を表している。そして、その各段階で、必ず一度活用を行っている。ウ系弱変化の連接形の場合は (2,3)、もとの弱変化の接辞がそのままの形で連接形という素性を持っている (活用表に則して言えば、弱変化系の活用をするものは、連接形がϕである) ので、一形態素に接辞と活用形の両方を記述している。
この活用形がϕであることと、上記の例文に、連接形が多く表われていることとは無関係ではないと思われる。すなわち、活用助辞がϕであるということは、文節構成の役割=屈折が、積極的でないと考えられるからである (強変化の場合は、本来、音韻変化とするか、接辞の側に含めるべきであることを既に述べた)。この屈折の役割は、活用形により積極性の段階があるが、ここでは触れない。
接辞の接続順序を考慮できる文法を考えた場合、上記の活用は、接続の順序とは別に考える必要がある。つまり、接辞の接続は順序を持つが、活用形は、積極的な順序性を持たないのである (ただし、順番を持った接辞が、何形に付くかは決まっているので、その意味での消極的な順序性が認められることも考えられる)。この接辞と活用の順序関係についても後述する。

活用の定義 以上、述べてきたことを活用の定義としてまとめておく。まず、活用の系列は図 3.13のようになる。
この他に助動詞 (接辞) の活用があるが、助動詞は何れも上記の活用系の変種で、ある活用形を持たなかったり、

	例文：「（彼らはまだ答案を）書かせられはじめていないようであった」		
1	書	用言語基	
	か	→ウ系強変化	連接形
2	せ	使役接辞→ウ系弱変化	連接形
3	られ	受身接辞→ウ系弱変化	連接形
4	はじめ	相接辞	
	て	→ウ系弱変化	完了中立形
5	い	テ形接辞→ウ系弱変化	現在中立形
6	な	否定接辞	
	い	→イ系	現在形
7	よう	ムード接辞	
	で	→ダ系ダ	現在中立形
8	あっ	断定接辞	
	た	→ウ形強変化	完了形

図 3.12: 例文 (活用形の役割)

語基	活用系	語基の例
用言語基用	ウ系┬強変化	行，死，書
	├弱変化	見，食べ，始め
	└混合変化┬サ変	φ（する），旅行
	└カ変	φ（来る）
形容語基用	イ系	赤，寒，新し
体言語基，情態語基用	ダ系┬ダ	静か，本，
	└デス	

図 3.13: 活用の系列

音便形を持っていたりという程度の違いであるため、上記に含めて考える。
活用の現象は次のように定義される。

活用とはある語基が，その語基用の活用形（形態素）をとり，
1．語基を構成する（派生）か
2．文節を構成する（屈折）か
のどちらかの働きをすることを言う．

図 3.14: 活用の現象の定義

さらに、活用形には次のものがある (3.15)。

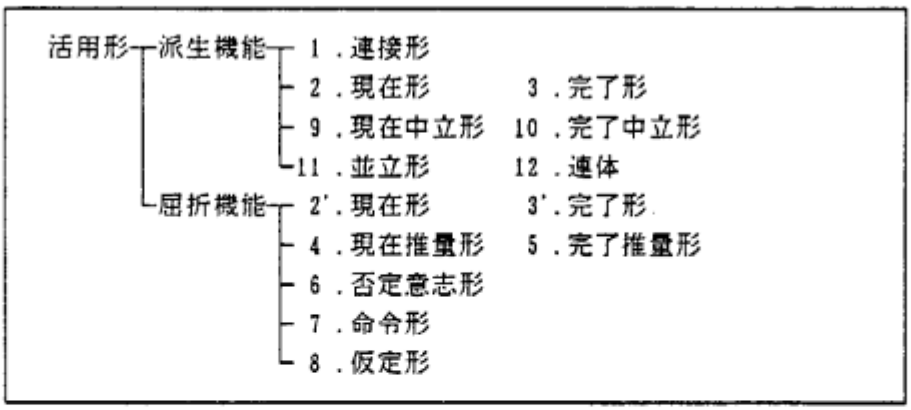


図 3.15: 活用形

3.2.3 各語基の派生 / 屈折形式

この章では、各語基毎に派生 / 屈折の形式を観察していくことにする。基本的には、森岡の語基分類に従い、各語基に付く接辞・助辞を網羅することを目的とするが、そのうち、あまり規則的でない部分は省き、また、接辞・助辞連接の順序を考慮して、記述する。
各語基の分類は 113頁の「文節を構成する語基の設定」で既に示したが、各語基間の派生や、ある語基に属していても他の語基の性質を持つものがあるため、その語基に特徴的なものを中心に観察する。例えば、「赤」は、体言語基に所属させたが、体言語基で記述するのは体言としての派生・屈折で、「赤−い」と接辞を取って、形容語基に派生した場合は、形容語基で記述する。
また、接辞 / 助辞は各語基に専用のものを中心にし、多くの語基に付き得るものは、次の「体言語基」以降で見ることにする。

［凡例］

［X 語基］（独立性＝○, ×, 素性）（一般的な後接情報）

○X 語基例
△X 語基用前接接辞
□X 語基用接辞
■X 語基用助辞
★X 語基用準用／準体辞

{語基, 接辞, 助辞の解説}
{語基に必要な素性情報}

体言語基 (独立性＝素性)(ダ系活用形、ただし連体形無し)

○体言　「花, 山, 文法, ペン」
○サ変体言　「心, 賞, 利, 愛, 旅行, ミス」　(ウ系サ変)
○ナ型体言　「味, 内気, 幸運, 健康, フリー」(ダ系連体形を含む)
○イ系体言　「赤, 四角, 黄色」　(イ系)
○形式体言　「後, 結果」
○不完全体言「嬢, か, と, じ, ば」

△接辞　「お, 御」　→体言
□接辞　「たち, ら」　→体言に派生
「らし−(い)」　→形容語基に派生（「男−らし−(い)」等で,
「行く−らし−(い)」とは異なる）
「のよう−(だ)」→情態言に派生
□準体辞　「の, への, との, よりの, からの, での」
■連体助辞　「の」
■格助辞　「が, を, に, へ, より, から, と, で, (連体従属節中の「の」＝「が」
ただし, 無用の曖昧性をさけるため助辞とはしない)」
■副助辞　「だけ, のみ, ばかり, など, くらい」
「と, や, やら, か, なり, とか, だの」(いわゆる並立助辞)
→同形の副派生辞がある
■係助辞　「は, も, こそ, さえ, でも, しか」
★準用辞　「のよう−(だ)」→情態言に派生
「なの−(だ)」　→情態言に派生
「らし−(い)」　→形容語基に派生

体言は上記6種に分類したが、この他に、「損-(な, する), 得-(な, する)」のようにナ型であると同時に、サ変でもある例が有るが、上記の分類を素性の付加と考えれば、ナ型とサ変の両者を素性として与えれば良いことになる。

体言に付く助辞の接続順序を考慮した、ゆるいアクセプタの例は次のようになる。

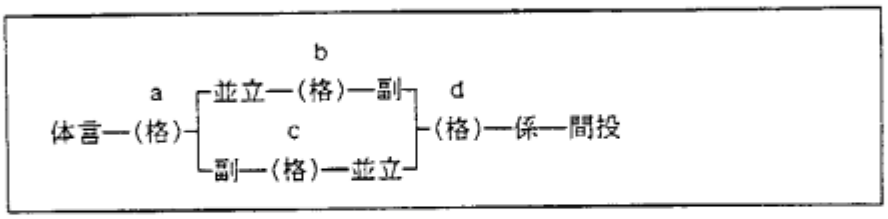


図 3.16: ゆるいアクセプタの例

1a. (彼にと) 僕_に_と_だけ_は_ね ([格]_並立_　副_　係_間投)
2b. (彼と)　僕_と_に_だけ_は_ね (　並立_[格]_副_　係_間投)
3d.　　僕_と_だけ_に_は_ね (　並立_　副_[格]_係_間投)

4a. (彼にだけと)　僕_に_だけ_と_は_ね ([格]_副_　並立_　係_間投)
5c. (彼だけにと)　僕_だけ_に_と_は_ね (　副_[格]_並立_　係_間投)
6d. (彼だけと) 僕_だけ_と_に_は_ね (　副_　並立_[格]_係_間投)

上記のアクセプタでは、格助辞の位置が明確でなく、見通しが悪いため、次の方針で整理する (図 3.17)。

1. 副助辞 / 並立助辞をまとめて副助辞とする。
2. 体言に直接下接する副助辞は、派生辞として、格助辞の後に付く副助辞と区別する。
3. 準体辞「の」と、連体助辞「の」を区別する。
   - 連体助辞は、いわゆる格助辞 (連体格) の「の」で、屈折専用
   - 準体辞「の」は、直前に弱格の格助辞のみを認め、「の, への, との, よりの, からの, での」の6種類とする。(屈折しない)

     ex. (約束は) 彼とのだけが→準体辞
     　　彼との (約束)　　→連体助辞
4. 用言が体言に準体する場合は、準体助辞の位置 (「の」) と、副派生辞の 位置 (副派生辞) とする。
5. 準体助辞 / 副派生辞は派生辞とし、格助辞 / 副助辞 / 係助辞 / 準連体助辞 は屈折辞として扱う。
6. 準用辞は特別の扱いをし、用言への派生とする。

「の」に関して、若干の補足を加えておく。

a. 彼の　本　　連体助辞 (必ず, 文節end)

b. 彼の—だけ(が)
　　に(だけ)
　　は　　準体辞　(endとならい場合)

c. 彼だけの　　連体助辞
(格助辞) (連体修飾中で, =「が」)

a. と b. とは、「の」の直後が新しい文節かどうかにより、判別できる。すなわち、連体助辞の「の」は、必ずそこで屈折が起り文節を構成し終えるが、準体辞の「の」は、end とならない。

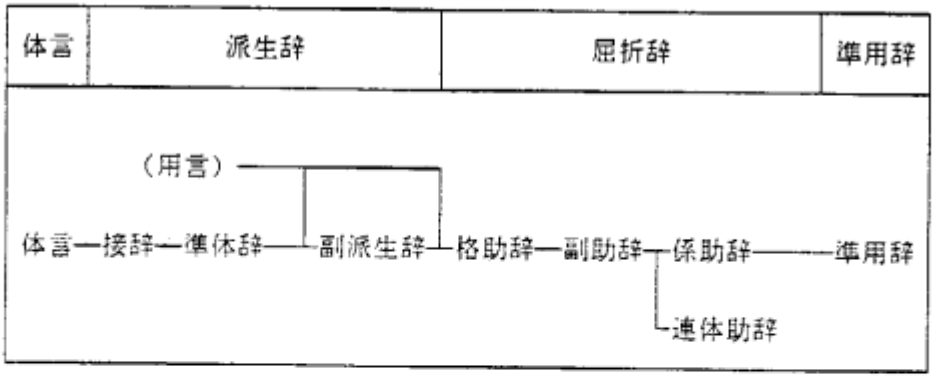


図 3.17: ゆるいアクセプタの例

c. の例は、「の」が体言に直接下接していないので、少なくとも準体辞ではない。従って、格助辞と連体助辞の可能性が有るが、文節構成の段階では、この区別は付かず、文節間の関係により決定するので、連体助辞としておく。a. の場合も後接の「本」が無ければ格助辞の可能性が有るが、b. の「の」には、格助辞の可能性はない。これは、格助辞「の」の後方接続素性は格助辞「が」と同じため（強調叙）、「だけ，は」などを後接させないからである（「に」は格助辞の重複となるため）。

d．彼からの 本　　連体助辞
e．（手紙は）彼からのが　準体辞

同様に、d. は格助辞＋連体助辞であり、e. は、「からの」で、準体辞である。

g．彼のようだ　　接辞　　（「のよう」で接辞）
g'．彼だけ（に）のようだ　準用辞
h．彼なのだ　　接辞　　（「なの」で接辞）
h'．彼だけ（に）なのだ　　準用辞

g.、h. は、接辞、準用辞の例である。体言に直接付く接辞は接辞、それ以外は準用辞と区別できる。h. の「なの」は、「な＋の」と考えることも出来るが、体言の後方接続素性としてのダ形活用形は、連体形「な」を持たないと考えるため(つまり、「なの」の接続においてのみ、「な」が表われる)、「なの」で一形態素と考える。

用言語基 (独立性＝×，弱変化はあり)(ウ系活用形)

○用言相対語基　「壊-(す，れ-る)，減-(る，ら-す)，閉-(ま-る，め-る)」
○用言強変化語基「太-(る)，話-(す)，聞-(く)，書-(く)，読-(む)」
○用言弱変化語基「痩せ-(る)，食べ-(る)，着-(る)，駆け-(る)」
○サ変用言語基　「接-，関-，呈-，排-」
○変格　「(来る)，(する)」
□自他派生辞　　各種（cf.「動詞自他の派生型」）
□態派生辞

「え，け，げ，せ，て，ね，べ，め，れ，-(る)」
（強変化用言語基に付いて可能を表す）　→弱変化用言語基
「れ-(る)」（弱変化に付いて可能を表す）　→弱変化用言語基

「せ-(る)，れ-(る)」　（強・連接形に付く）　→弱変化用言語基
「させ-(る)，られ-(る)」（弱・連接形に付く）　→弱変化用言語基

□派生辞

「た-(い)」　（現在中立形に付く）　→形容語基
「すぎ-(る)」（現在中立形に付く）　→弱変化用言語基
「がち-(だ)」（現在中立形に付く）　→情態語基
「べき-(だ)」（現在形に付く）　→情態語基

□相派生辞（完了中立形に接続，いづれも補助用言）

「い-(る，弱)，あ-(る，強)，しま-(う，強)」（aspect）
「あげ-(る，弱)，や-(る，強)，もら-(う，強)，くれ-(る，弱)」（voice）
「さしあげ-(る，弱)，いただ-(く，強)，くださ-(る，弱)](voice+style)
「み-(る，弱)，みせ-(る，弱)，お-(く，強)」（？）

□ムード派生辞

「そう(-だ)」　→情態語基
「よう(-だ)」　→情態語基
「みたい-(だ)」→情態語基
「らし(-い)」　→形容語基

□否定派生辞

「な-(い)」→形容語基へ派生
「なければならな-(い)」，「なければいけな-(い)」→形容語基へ派生
「なくてはならな-(い)」，「なくてはいけな-(い)」→形容語基へ派生

□スタイル派生辞

「ます」

■ムード屈折辞「だろう」
■係助辞　「は，も，こそ，さえ，でも，しか」
■接続助辞「と，けれど(も)，のに，ので，が，から，し」
■準副助辞「ながら，がてら」
■連体助辞「の」
★準体辞　「の，から」┐
★副派生辞　　└体言に準用

用言語基は、後接する活用形により、上記の様に分けられる。すなわち、強変化の語基は強変化の活用形に、弱変化の語基は弱変化の活用形に、サ変の語基はサ変の活用形に続く(強変化の場合は、何行かという情報も必要)。特殊なのは、変格活用の用言で、「来る,（本動詞の）する」は、語基と活用形が融合して分離出来ないため、活用形そのものの中に、語基が含まれることになる。用言相対語基は、相対自動 / 相対他動の基本となる語基で、そのままの語形、あるいは接辞が付いて、用言語基として用いられる。

用言語基の活用を考慮した文節構成の概略は次のようになる(図 3.18)。

- 「の」についての補足

  a. 行っての 話だ。　　連体助辞(完了中立形接続)
  b. 行くのが／行ったのが　　準体辞　(現在／完了形接続)
  c. 行くのだ／行ったのだ　　準体辞　(現在／完了形接続)

  a. は、体言語基につく連体助辞と同様に、「の」で屈折し文節を構成している。b. と c. は、体言語基に派生し、体言語基としての資格を得て、それぞれ、格助辞、ダ形活用形に接続している。この2つ「の」は、直前の活用形により判別できる。

- 「から」についての補足

  a. そんなに 食べたから　　接続助辞(現在／完了形接続)
  b. その話は 食べてからだ
     食べてからが　　準体辞　(完了中立形接続)

  a. は、接続助辞で、屈折して文節を構成、b. は体言語基に派生して、体言語基の資格で文節を構成している。これも、活用形の接続により判別できる。問題となるのは、次の形である。

  c. そんなに 食べるからだ
  d. そんなに 食べたからだ

  現在形 / 完了形に接続しているところからすると、接続助辞であり、ダ形活用形をとる点からは、準体辞と考えられる。ただし、意味的には、接続助辞と同じ原因・理由を表しており、「*食べるからが」と言えないように、後方接続はダ形活用形のみであり、体言としての資格を持っているとは考えられない。また、後接するダ形活用形も現在形、連用形に制限される。従って、ここでは接続助辞の特例と考えることにする。

情態語基 (独立性＝素性)(ダ系活用形)

	める	がる	がる
○情態語基 a	「あらた, あわれ, いや, うつろ, うぶ, たいら, まばら, はで」		
情態語基 b	「あざやか, おろか, おろそか, さわやか, すこやか」		
漢語系情態語基	「妙, 変, 急, 簡単, 有利, 残念」		
○ノ型情態語基	「新規, 僅か」		
○イ系情態言	「暖か, 柔らか, 同じ」		
○非自立情態語基	「静－, 遥－, ひそ－, ほの－, かす－, にぎ－」		
□非自立用派生辞	「か」	→情態語基に派生	
	「まる, わう」	→用言語基（強変化）に派生	
	「めく, める」	→用言語基（弱変化）に派生	
	「かさ, けさ」	→体言語基に派生	
	反覆	→副用語基に派生	
■副用助辞	「に」	→副詞に屈折	

上記の語基は、森岡の情態言、情態語基を含む。情態語基 a、b の区別は不要であるが、例えば b の「おろか」と非自立形の「静－(か)」との区別は必要である。この区別は、「か」あるいは「やか」をとった部分が造語するかどうかによる。特に、反覆して用いられれば、それは非自立形の情態語基である。

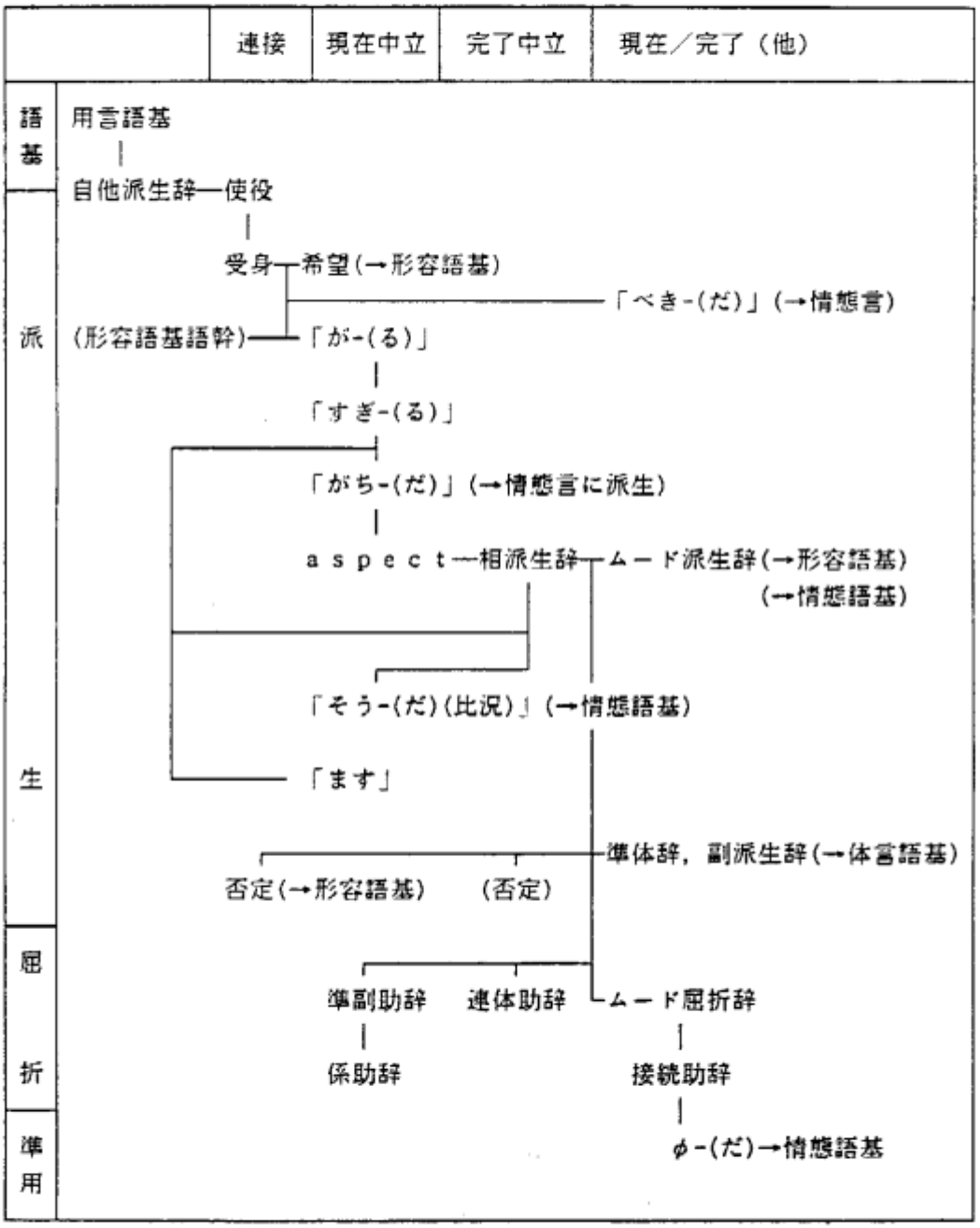


図 3.18: 文章構成の概略

非自立形の情態語基の派生を以下に示すが、これらは数も少なく、和語系の語基に限られているので、あらかじめ登録しておくか、接辞を付けた形で各語基に登録すれば良い (図 3.19)。

	やかか	まる	める	めく	わう	反（「と」が任意の情態 覆　副詞）
静	○	○	○			○
はる	○			○		○
ひそ	○○	○	○			○
ほの	○			○		○
かす	○		○			○（に）
にぎ	○				○	△（-し(い)，で形容語基）

図 3.19: 非自立系の状態語基の派生の例

情態語基の特徴は、ダ系活用系を取ることであるが、それ自体は体言と同じである。体言と異なるのは、副用助辞「に」を取ることである。ただし、体言は格助辞「に」を取るため、形態的に区別がつかない。むしろ格助辞一般を取らないという消極的な区別が有効である。また、体言が「だ」の連体形として「の」を取る (ここでは、連体助辞として扱う) のに対して「な」をとるという違いがある (ただし、ノ型は「の」あるいは「の」「な」両方を取る)。
また、イ系情態語基は形容語基と同様にイ系活用形をとる。

形容語基 (独立性＝×)(イ系活用形)

○形状性形容語基「高，低，広，狭，薄，固，深，浅，速，遅」
　感覚性形容語基「寒，暑，辛，甘，苦，痛，恐，悪，冷た」
　シク系形容語基「悲し，美し，苦し，淋し，嬉し，貧し，親し，等し」
○ナ型　形容語基「可笑し，大き，小さ」
□接辞「ま-(る)，が-(る)，む，ら-(ぐ)，ら-(む)」→用言語基 (強変化) に派生
　「め-(る)，げ-(る)，らめ-(る)」　→用言語基 (弱変化) に派生
　「さ，み」　→体言語基に派生
　「げ，な，らか，やか」　→情態語基に派生
■助辞「な」

形容語基は、イ系活用形を後接させる 3 種と、さらに助辞「な」を後接させるナ型の計 4 種がある。「な」は、ダ系活用形の連体 1「な」と同じであるが、ナ型形容語基には「な」以外のダ系活用形が後接しないため、「な」を助辞として取扱い、ナ型の語基は情態語基ではなく、形容語基として扱う (cf. イ系情態語基)。
形容語基の前者 3 種は森岡に従い分類してあり、種類により後接する接辞に特徴が見られるが、実際には各形態素ごとに後接辞の素性を与えなければならない。従って、形容語基の分類自体は必要無い。ただし、体言語基に派生させる「さ」は総てのイ系語基 (形容語基だけでなく) に接続するため素性を与える必要はない。また、この素性は、イ系語基の総てに与える。
図 3.20に、形容語基、その他のイ系語基の接辞 / 助辞連接の例をあげる。表中の記号の意味は次のとおり。

○：岩波国語辞典（第三判）に見出し語の有るもの。
△：見出し語に無いが，該当語が有りえると判断したもの。
×：接続しない。

ただし、「らか，やか，らぐ，らむ，らめる」は、各形態素の欄に接続する接辞を記した。また、"/"は「暖か，柔らか」で「か」を除いた部分「暖,柔ら」に接続することを示す。 畳語の部分は左側が元の形態素から意味を構築できると考えられるもの、右側が構築出来ないと思われるものに分けて記した。記号は派生先とその接辞連接のパターン別にしてある。

副用語基 ■：自立して文節を構成
●：自立して文節を構成，「と」をとる
▲：自立せず，「と」をとる
▼：自立せず，「と」；「(と)する」をとる
形容語基 ★：反覆＋「し-(い)」
体言語基 〒：

次ページの表の範囲からイ系語基に必要と考えられる接続素性は次のものである。

接辞｛まる，める，がる，げる，げ，む，み，め，らか，やか，らぐ，らむ，らめる｝
助辞｛な｝（ただし，ナ型形容語基を分類するなら不要）
反覆｛副用語基1，副用語基2，副用語基3，副用語基4，形容語基，体言語基｝

ただし、対象とする形態素数が増えれば当然、素性もさらに増えると考えられる。（注. 煙た-(い)、眠た-(い) は煙-(い)、眠-(い) とは別としたほうが楽か？）

副用語基（独立性＝素性）

○程度性副詞語基「すぐ，すこし」
○象徴性副用語基「きらきら，どんどん，さらさら」
■副用助辞「に，と」

ここでいう副用語基とは、森岡の副用語、象徴語基、副用語基をまとめたもので、副詞として自立するかどうかは、各副用語基に独立性の素性として付加されているものとする。
副用語基の後方接続は形容語基のとる接辞と同様に一般化が難しく、各語の接続素性により判断するしかない。副用語基に後接する典型は「に，と」の副用助辞であるが、これも各語の接続素性による。その他、副助辞、係助辞、サ変活用形、ダ形活用形にも接続する（図 3.21）。
「と」については、『「と」の消失についての資料』参照。

数詞語基（独立性＝×）

○和語系数詞語基「ひと，ふた，み，よ，いつ，む，なな，や，ここの，いく」
○漢語系数詞語基「一，二，三，四，五，六，七，八，九」
○数字系数詞語基「1,2,3,4,5,6,7,8,9,0」
○漢語系単位語基「十，百，千，万，億，京」
○記号系語基　「", ",". ", "、 ","。 "」
△前接助数辞　「数，何，幾」

			まる	める	がる	げる	げ	む	み	め	な	だ	畳語	他
形容詞	形状性	高	○	○	×	×	×	×	○	○	×	×	▲■	-らか
		低	○	○	×	×	×	×	○	○	×	×	×	
		広	○	○	○	○	×	×	×	△	×	×	▼	-やか
		狭(ば)	○	○	×	×	×	×	×	×	×	×	×	
		薄	×	○	×	×	×	×	×	○	×	×	■	-らぐ
		固	○	○	×	×	×	×	×	○	×	×	×	
		深	○	○	×	×	×	×	○	△	×	×	▲	
		浅	×	×	×	×	×	×	○	△	×	×	×	
		早	○	○	×	×	×	×	×	△	×	×	▲	
		遅	×	×	×	×	×	×	×	△	×	×	×	
	感覚性	寒	×	×	△	×	×	×	×	×	×	×	▼	
		熱	×	×	△	×	×	×	×	×	×	×	〒	
		辛つら	×	×	×	×	×	×	×	×	×	×	×	
		重	×	×	×	×	○	×	△	×	×	×	★	
		苦にが	×	×	×	×	×	×	○	×	×	×	★	
		痛	×	○	△	×	×	○	○	×	×	×	★	
		恐こわ	×	×	○	×	×	×	×	×	×	×	●	
		悪わる	×	×	×	×	×	×	×	×	×	×	×	
		冷た	×	×	×	×	×	×	×	×	×	×	×	
	シク系	悲し	×	×	△	×	△	○	○	×	×	×	×	
		美し	×	×	△	×	△	×	×	×	×	×	×	
		苦し	×	○	△	×	△	○	△	×	×	×	×	
		淋し	×	×	△	×	△	×	×	×	×	×	×	
		嬉し	×	×	△	×	△	×	×	×	×	×	×	
		貧し	×	×	×	×	×	×	×	×	×	×	×	
		親し	×	×	△	×	△	○	○	×	×	×	×	
		等し	×	×	×	×	×	×	×	×	×	×	×	
	ナ型	可笑し	×	×	△	×	△	×	×	×	○	×	×	
		大き	×	×	×	×	×	×	×	△	○	×	×	
		小さ	×	×	×	×	○	×	×	△	○	×	×	
体言イ系		赤	×	×	×	×	×	×	○	×	×	×	▲	-らむ
		四角	×	×	×	×	×	×	×	×	×	×	×	らめる
		黄色	×	×	×	×	×	×	×	×	×	×	×	
情態語基イ系		細か	×	×	×	×	×	×	×	△	○	○	×	
		暖か	／	／	×	×	×	×	△	△	○	○	×	
		柔らか	×	×	×	／	×	×	×	△	○	○	×	

図 3.20: 形容語基の例

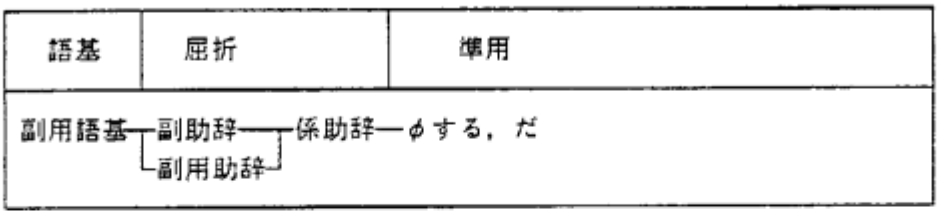


図 3.21: 副用語基の例

□漢語係助数辞　「個，回，匹，部，冊，段，人，重，年，月，日，…」→体言へ派生
□和語系助数辞　「つ，り」　→体言へ派生

実際には和語系語基が表われることはほとんど無く、「ひとーつ」、「ひとーり」は、それぞれ、「一つ」、「一人」と書かれることが多い。
「一人，一日，一重」などは、その語がどう読まれても関係なく、漢語系数詞語基＋助数辞として扱える。「一つ」の方は、漢語系数詞語基の内、「一〜九」にのみ付く。 和語系数詞語基は、必ず和語系助数辞に接続し体言語基となるが、それ以外の数詞語基はそのまま体言語基の資格を持つ。ただし、直接次の文節に続くことはなく、必ず、記号系語基に接続する。どちらの場合も助数辞に接続した場合は、体言語基としての資格を持つ。

ex. *ひとください，*一ください，*1ください
　　ひとつください，一個ください，1個ください，

それぞれの語基、助数辞の後方接続素性を次に示す。

和語系数詞語基［*和語系助数辞］
漢語系数詞語基［漢語系単位語基，漢語系数詞語基，漢語系助数辞，*和語系助数辞，記号系語基］
数字系数詞語基［漢語系単位語基，漢語系助数辞，*和語系助数辞，記号系語基］
漢語系単位語基［漢語系数詞語基，前接助数辞，漢語系助数辞，記号系語基］
前接助数辞　［漢語系単位語基，漢語系助数辞，記号系語基］

なお、数詞合成の規則は、CFGで簡単に記述できるが、3型文法では完全な記述が出来ないので、上記程度の記述に止めておく。*和語系助数辞への接続は、以下の通り。

指示語基 (独立性＝×)

○指示語基「こ，そ，あ，ど」
□指示接辞「(そ) こ，れ，ちら，っち，なた，いつ」
■指示助辞「の，んな」　→連体詞
　「んなに，う (あ)」　→副詞
　「ら」　→感動詞

指示語基では、接続の規則的な形態素のみを採用する。従って、規則から洩れるものは、個別に登録する。
接辞はすべて体言語基への派生である。助辞は「の，んな」が連体詞へ、「ら」が感動詞へ、「う」(ただし「あ」には接続せず) が副詞への屈折である。
上図で、助辞「う (あ)」には、ウ系サ変活用形「する」が接続する。また、「X のよう-(だ)」に接続する。

助数辞		和語系 ひと	ふたり	みよ	いつ	むな	なや	ここの	いく	漢語系 一―九	漢語系 単位	数字系 1―9	数字系 0
つ	っ	○	○		○		○		○	○	×	○	×
	っっ			○		○		○					
り		○	○	×	×	×	×	×	×	×	×	×	×

図 3.22: 和語系助数辞への接続

体言語基	→	指示語基　指示接辞
文節1	→	指示語基　指示助辞

	接辞 そこ	れ	ちら	っち	なた	いつ	助辞 の	んな	んなに	ら	うあ
こ	○	○	○	○	△	○	○	○	○	○	○
そ	○	○	○	○	△	○	○	○	○	○	○
あ	○	○	○	○	○	○	○	○	○	○	○
ど	○	○	○	○	○	○	○	○	○	○	○

図 3.23: 指示語基

3.2.4 まとめ

形態素の基本単位である語基分類を中心に、形態素文法について纏めた。派生 / 屈折による文節 (語) の解析では、単語という考え方が一般流通辞書のそれとは異なる。
形態素解析は、構文解析のための適当な前処理ではなく、統語論に対する形態論の部門であり、形態素を単位として文節 (語) の構成を記述することにある。計算可能である必要から、 3.2.2では派生 / 屈折を有限状態文法へ適用した。日本語では、基本的な文の要素は述部にあることからいわゆる活用形態に再考を加えた。この結果は、形態素を基本にしているにもかかわらず、意味あるいは構文を強く反映している。活用形態から定型述部であるかどうか、従属型述部であるかが判明する。
3.2.3では、区分した語基のそれぞれについて、派生と屈折の形式について記述した。辞は助詞に置き換えると、直感的な解釈が容易となるだろう。形態素結果は必ずしも特定の構文解析を想定していない。例えば、意味解析の入力は意味素の列であって、構文木でないように、形態素解析の入力は単語ではなく形態素である。従って、構文木が意味に対応しないように、形態素解析結果の文節 (語) が、構文解析の最小単位である単語に対応しないことは明らかである。
尚、助辞区分は詳細になるため、この報告では省略した。

和文索引

英文索引

LTB 操作説明書

– SAX 編 –

(LTB1.1 版)

ICOT 第二研究室

平成元年 3 月 31 日

本説明書は構文解析システム SAX の操作方法について記述している。

- SAX の概要
- SAX の操作方法
- SAX クラスをユーザプログラムから使うためのインタフェース
- 文法作成

汎用日本語処理系 LTB(Language Tool Box) の説明書は次のような構成になっている。

- LTB 概要編
- LTB マスタ辞書編
- CIL 編
- LAX 編
- SAX 編
- 文生成編

目次

図目次

第1章

SAXの概要

SAXは自然言語のための構文解析システムである。

SAXで構文解析を行なうためには、ユーザはまず「文法」を作成しなければならない。[⟹ 第4章 文法作成]

システムは与えられた「文法」に従って「入力」に対し構文解析を行なう。「入力」は、LAX[1]の形態素解析結果でもよいし、キーボードからタイプされた単語列でもよい－その両方のユーザインターフェースが提供される。[⟹ 実行ウィンドウでの操作]

またユーザはSAXシステムの中でCILの豊富な機能を活用し、意味処理を構文解析中に実行させることもできる。[⟹ 第4章文法作成]

システムを他のユーザプログラムから呼びだすためのインターフェースも解放されており本システムを使って開発した構文解析プログラムを別のシステムに組み込むことも可能であるし、システムで提供される各ツールもユーザプログラムからの起動が可能である。[⟹ 第3章ユーザインターフェース]

規模の大きな文法を効率よく開発するためにはどうしてもシステムのサポートが必要である。SAXは効率のよい文法開発を可能にするためデバッグツールを用意している。[⟹ 2.5トレーサ]; [⟹ 2.6ビジュアルデバッガ]

1.1 システム構成

SAXの構成を 図1.1システム構成に示す。

1. トランスレータ

   ユーザによって与えられた「文法」を計算機の上で実行可能な「構文解析プログラム」に変換する。

2. デバッガ

   文法のデバッグを支援する。

3. ツリーブラウザ

   解析結果をウィンドウ上にユーザに分かりやすい形で表示する。

[1] LTB 操作説明書 LAX 編参照

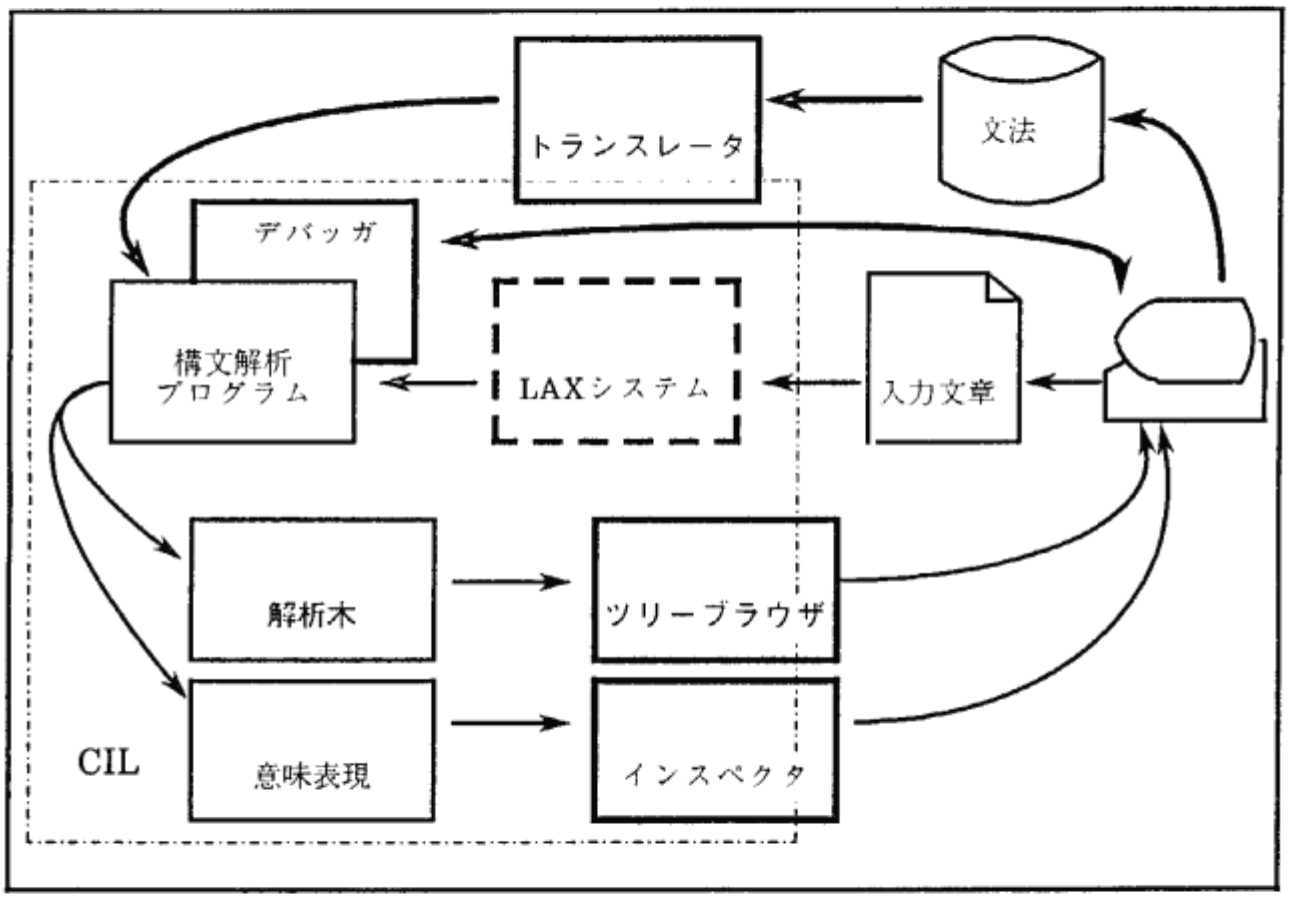


図 1.1: SAX システムの構成

第 2 章

SAX の操作方法

2.1 SAX の起動方法

起動—
SAX システムは、SIMPOS のシステムメニューまたは、LTB シェルから起動される。

システムメニューから システムメニューで sax を選択する。

すると、図2.1のようなSAX システムのトップメニューがスクリーン上に現れる。
終了—
SAX を終了するにはシステムメニュー (図2.1) のメニューの 終了 を選択する。

2.2 トップウィンドウでの操作

SAX の操作は全てこのトップウィンドウから、コマンドを入力することによって始まる。

トップウィンドウの構成

トップウィンドウ (図2.1) は、

- メインメニュー (a)
- エディタウィンドウ (b)
- 状態表示エリア (c) [1]

からなる。

コマンドの入力

コマンドはメニューから選択するか、またはキーボードからタイプ して入力する。
ここでは、メニューを使った操作のみを説明し、キーボードからコマンドを打ち込む場合の書式については、3.4でその一覧を示すことにする。

[1] SAX システムでは使用していない

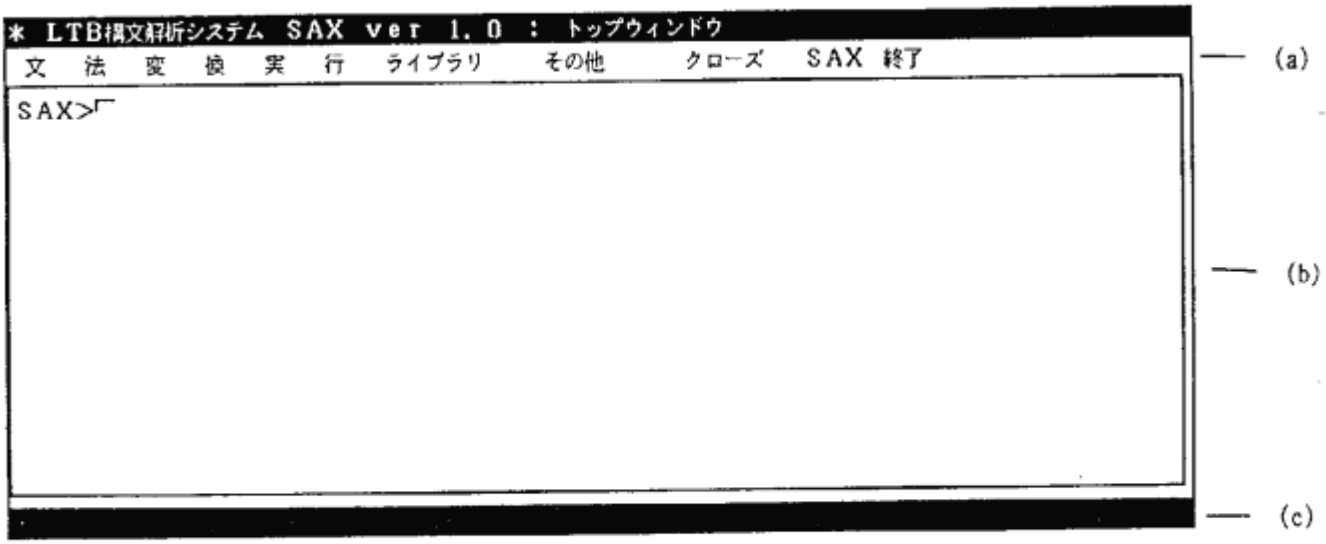


図 2.1: トップウィンドウ

メニューによる操作

メインメニューには、次の項目が、表示される。

文法　　変換　　実行　　ライブラリ　　その他　　クローズ　　SAX　　終了

機能一覧

- 文法: 文法規則のシステムへの登録や削除を行う。
- 変換: SAX トランスレータを起動し「文法」を構文解析プログラムに変換する。
- 実行: 構文解析を行うためのモードに移行する。
- ライブラリ: 構文解析プログラムのカタログ、セーブ [2]等、また「文法」の中に記述された CIL 手続きのコンパイルなどを行う
- その他: コマンドの別名定義などを行う。
- クローズ: ウィンドウのクローズ[3]を行う
- 終了: SAX システムを終了する。

文法

メインメニューの文法を選択すると、図2.2のようなサブメニューが現れる。

[2] SIMPOS 操作説明書 (I) 基本編を参照

[3] ウィンドウを消して代わりにアイコンを表示すること

図 2.2: 文法サブメニュー

登録

ユーザが作成した文法をシステムに登録する場合、この項目を選択する。すると登録するファイル名を聞いてくるので、キーボードからファイル名を入力する。ファイルは必ず"me:dcg>"の中に存在しなければならない。

削除

文法のシステムへの登録を抹消する場合、この項目を選択する。

変換

メインメニューの変換を選択すると、図2.3のようなサブメニューが現れる。
文法の変換は次のように行う。

1. サブメニューのなかの変換を行いたいファイル名の所をマウスクリックする。すると、その部分が反転表示になる。このメニューは、マルチセレクト[4]になっており複数のファイル名を選択することができる。

2. ファイルを選んだら次に実行という項目を選択する。すると図2.4のトランスレータオプションメニューが現れる。トランスレートオプションを適当に設定する。これらのトランスレータオプションは、文法の中で宣言して[5]おくとその設定でオプションメニューが現れる。また、文法ファイルの宣言部に

```
:- menu(off).
```

と、書いておくとこのオプションメニューは出ない。

オプション項目 Language 変換の目的言語を指定する。ただし ESP を選択したときのみデバッガ等のツールの使用が可能である。

ESP

Prolog Prlog を選択した場合は、以下のオプション項目は意味を 持たない。

ESP source file 変換した結果の構文解析プログラムのファイルを残すかどうかを選択する。

yes ファイルを残す

no ファイルを残さない

[4]SIMPOS プログラミング説明書 - 入出力編を参照
[5]第4章 文法作成 の宣言を参照

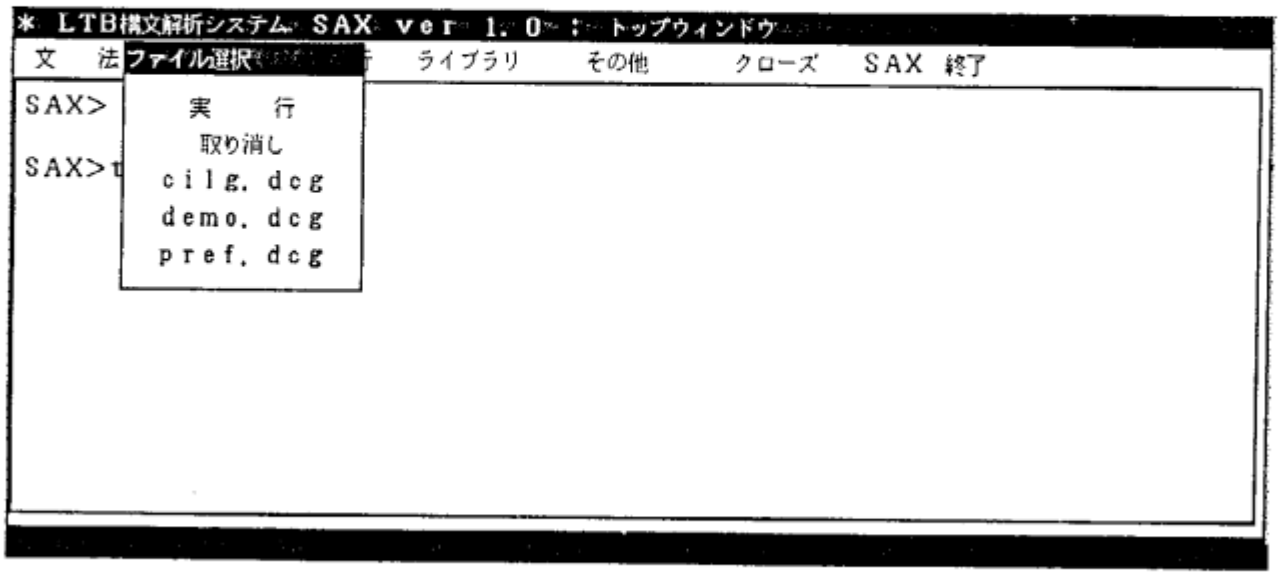


図 2.3: 変換サブメニュー

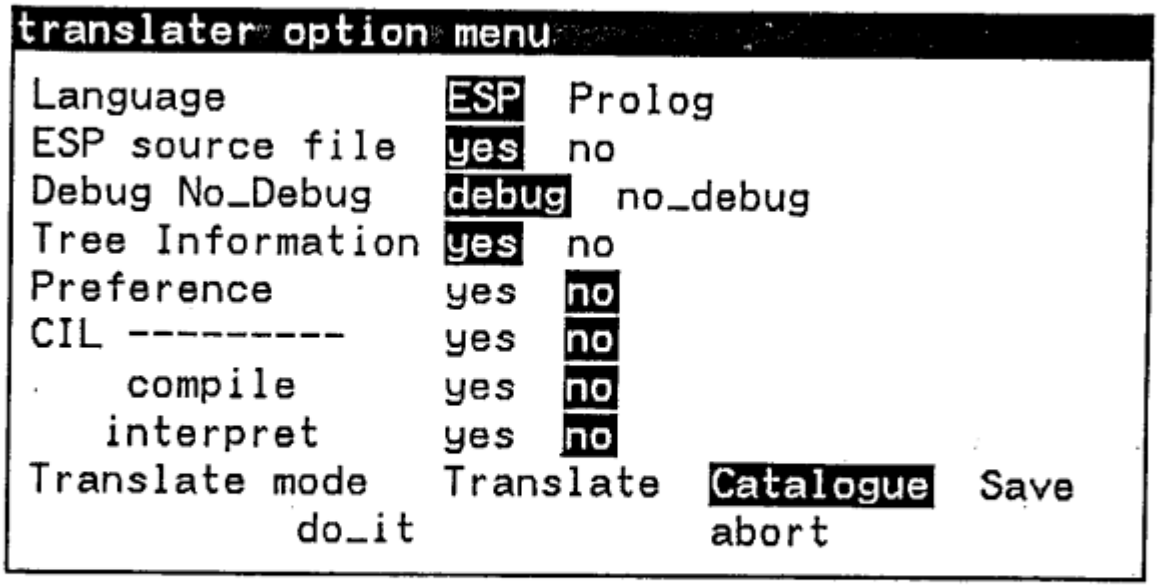


図 2.4: トランスレータオプションメニュー

Debug No_Debug トレーサ等のデバッグツールを使用するには、それに必要なコードを出さなくてはならない。この項目はデバッグ用のコードを出力するか、否かを選択するものである。

debug デバッグ用のコードを出す。(デバッガが使用可能だが解析実行速度が遅い。)

no_debug デバッガの使えないコードを出す。

Tree Information 構文解析実行中にシステムが解析木を構成するかどうかを選択する。

yes 解析木を作る。

no 解析木を作らない。

Preference SAX システムは解析中に (優先度) を計算する機能を提供する。[6]

yes 優先度規則を記述した文法を変換する。

no 優先度規則を記述していない文法を変換する。

CIL SAX システムからは CIL の機能が使用可能である。[7]

yes CIL を使用している文法を変換する。 こちらを選択した場合、オプション項目の compile か、または interpret の内どちらかで yes を選択しなければならない

no CIL を使用していない文法を変換する。

compile CIL をコンパイルモードで使用するか、否かを選択する。 CIL を使用しない場合は、本項目は無視される。

yes CIL をコンパイルモードで使用する。

no CIL をコンパイルモードで使用しない。

interpretive CIL をインタプリティブモードで使用するか、否かを選択する。 CIL を使用しない場合は、本項目は無視される。

yes CIL をインタプリティブモードで使用する。

no CIL をインタプリティブモードで使用しない。

Translate mode 変換したプログラムのカタログ等を行うかどうかを選択する。

Translate 文法の変換のみを行う。

Catalogue 変換したプログラムのカタログを行う。

Save 変換したプログラムのカタログとテンプレートのセーブ[8]を行う。

実行

メインメニューの実行を選択すると、図2.5のようなサブメニューが現れる。ここで、文法のファイル名を選択すると、解析実行モードに移行する。解析実行モードでの操作は、2.3で説明する。

ライブラリ

メインメニューのライブラリを選択すると、図2.6のようなサブメニューが現れる。サブメニューの項目を選択すると文法選択メニューが現れるのでそこで文法を選択するとその文法にたいして操作が行われる。

[6] 4の「文法作成」を参照

[7] 4の「文法作成」を参照

[8] SIMPOS プログラミング説明書 - 入出力編を参照

図 2.5: 実行サブメニュー

カタログ その文法に対応する構文解析プログラムのカタログを実行する。

セーブ その文法に対応する構文解析プログラムのテンプレートファイルのセーブを実行する。

CIL コンパイル その文法に関連する CIL プログラムのコンパイルを実行する。

その他

メインメニューのその他を選択すると、図2.7のようなサブメニューが現れる。

別名定義 コマンドの別名を定義する。

別名一覧 コマンドの別名定義を表示する。

別名定義削除 コマンドの別名定義を削除する。

環境ファイル更新 SAX システムの環境ファイルの更新を実行する。

SAX 強制終了 環境ファイルの更新を行わずに SAX を終了する。

クローズ

トップウィンドウをクローズする。

SAX 終了

SAX を終了する。このとき同時に環境ファイルの更新が行われる。

図 2.6: ライブラリ

図 2.7: その他

2.3 実行ウィンドウでの操作

実行ウィンドウの主な機能は, 解析文の入力および解析・解析結果の構文木情報の表示・意味情報の表示・トレースモードの切り替え・ビジュアルデバッガの起動である。
トップウィンドウで実行を選択するとトップウィンドウが消え, その位置に解析実行ウィンドウがあらわれる。(図2.8)
トップウィンドウ上ではSAXシステムに登録されている全ての文法ファイルが処理対象になるのと違い, 実行メニューで選択した文法ファイルのパーサクラスのみが処理対象となる。
プロンプトは最初'実行>'となっている。このときは構文木表示・意味表示は選択できない。選択した場合はコマンドエラーとなる。入力文の解析を行った後, プロンプトが'解析'にかわる。

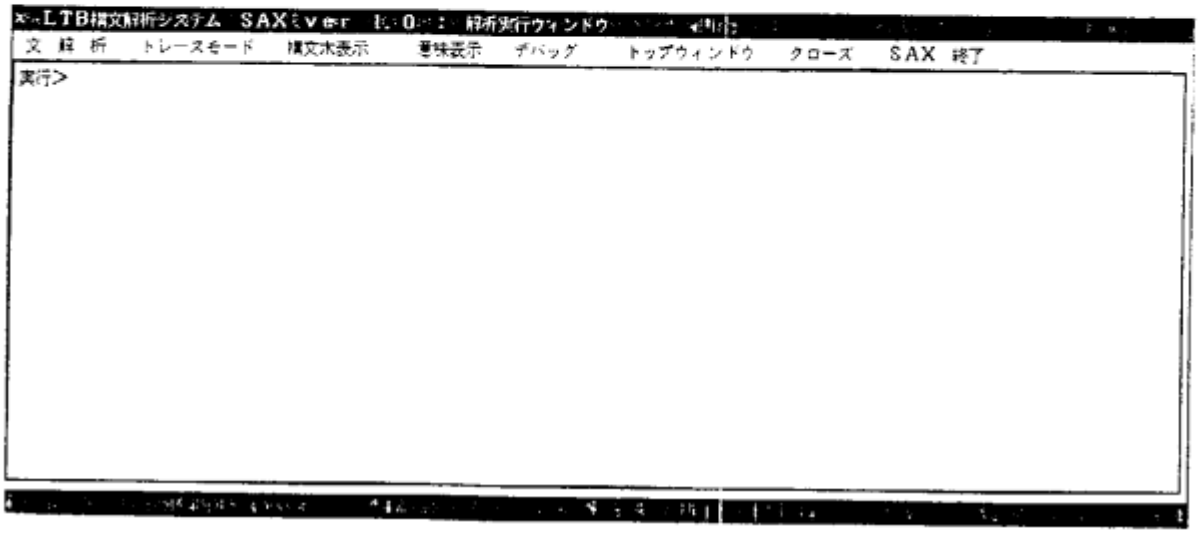


図 2.8: 解析実行ウィンドウ

コマンドの入力

コマンドはメニューから選択するか, またはキーボードからタイプして入力する。
メインメニューには次の項目が表示される。

文解析　トレースモード　構文木表示　意味表示　デバッグ　トップウィンドウ　クローズ　終了

機能一覧

- **文解析**
  解析する文を入力し構文解析の実行を行なう。
- **トレースモード**
  トレース / ノートレースモードの切り替えを行なう。
- **構文木表示**
  解析結果の構文木情報をグラフィック・テキスト表示する。
- **意味表示**
  解析結果の意味構造をプリティプリントする。
- ***デバッグ***
  ビジュアルデバッガを起動する。
- **トップウィンドウ**
  実行ウィンドウを終了してトップウィンドウにもどる。

- **クローズ**
  実行ウィンドウをクローズする。
- **終了**
  SAX システムを終了する。

文解析

解析文の入力および構文解析の実行を行なう。
メインメニューの文解析を選択するとサブメニューが表れるので, キーボード・LAX のどちらかを選択する。コマンド入力時はそれぞれ "keyboard","lax" と入力する。

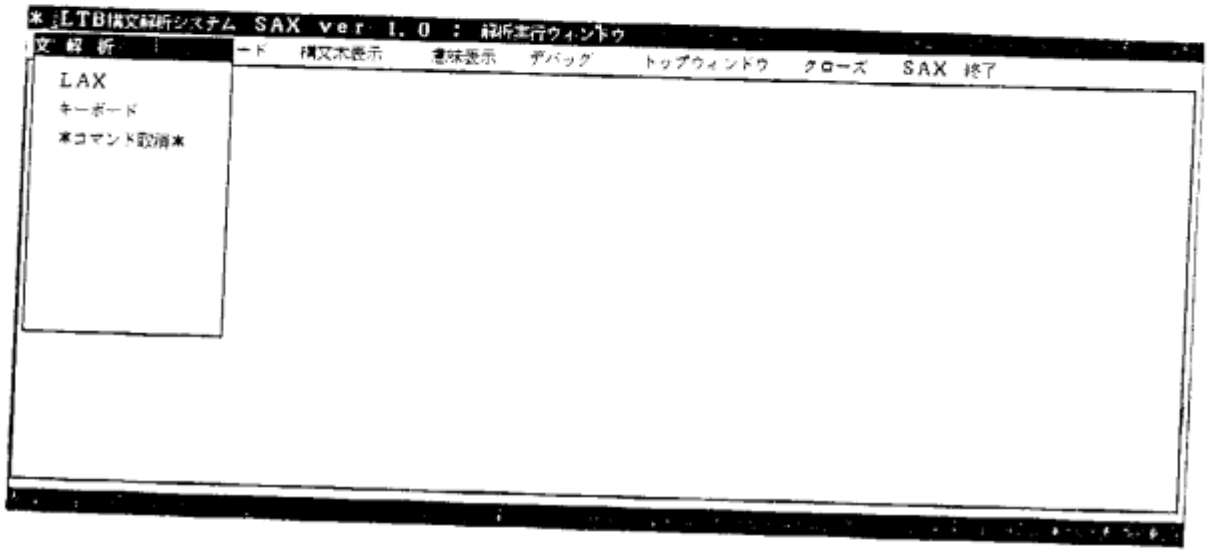


図 2.9: 文解析サブメニュー

キーボード
キーボードから解析したい文を入力する。プロンプトが解析文入力>となる。
入力は単語間を 1 つの空白で区切って入力する。改行キーを押すことで入力終了となる。

```
実行>keyboard
解析文入力>he saw the woman with the telescope
```

図 2.10: キーボード入力

入力後, デバッグモードであれば単語のチェックを行なう。入力文中に未定義語がある場合はエラーメッセージを表示し解析を中止する。

LAX
LAX インターフェースを通して解析ゴール列を読みこみ解析を行なう。
読みこんだゴール列はウィンドウに表示される。このときゴール列に欠陥があるばあいはエラーメッセージを表示して解析を中止する。

```
解析>keyboard
解析文入力>he saw the woman with the telescop
＊ 終端記号 telescop が未定義です. ＊ 解析を中止しました.
```

図 2.11: 未定義語エラーメッセージ

解析結果表示

解析が終了すると, 解析数, 解析所要時間, 優先得点 (優先記述がある場合) をウィンドウに表示する。解析が途中で失敗した場合はエラーメッセージを表示する。解析が途中で失敗するのは通常, ノーデバッグ時で入力文中に未定義語が含まれている場合である。

```
解析文入力>he saw the man with the telescope
 解析数 = 4 : 解析時間 = 28 msec.
 優先得点
 No. 1 : 7
 No. 2 : 37
 No. 3 : 7
 No. 4 : 47
```

図 2.12: 解析結果表示

```
実行>keyboard
解析文入力>who is that girl
＊＊  解析できません.

実行>
```

図 2.13: 解析失敗時のエラーメッセージ

トレースモード

メインメニューでトレースモードを選択するとトレースモードの切り替えとなる。選択の度にトレース, ノートレースとトグルする。コマンド入力の場合は trace でトレースモードに,no_trace でノートレースモードとなる。トレース状態はボトムラベルに表示されている。ノーデバッグ時にはコマンドエラーとなる。

トレース状態になってから最初の解析文入力後にトレース用ウィンドウの線画が表れるので, マウスで位置決めを行なう。解析が始まるとトレースウィンドウがアクティベートとなる。解析終了後デアクティベートとなる。CIL 記述がある場合は CIL デバッガを用いるので, トレースウィンドウの位置決めのあと CIL デバッガウィンドウの位置決めも行なう。トレースウィンドウの操作については 2.6 トレーサを,CIL デバッガの操作については LTB 操作説明書 CIL 編を参照のこと。

構文木表示

解析により得られた木構造を表示する。まだ解析がなされてない場合 (プロンプトが '実行>') に選択するとコマンドエラーになる。解析の結果, 解が 1 つもない場合は何もしない。

メインメニューで構文木表示を選択するとサブメニューがあらわれるので, グラフィック表示・テキスト表示のどちらかを選択する。コマンド入力時はそれぞれ g_tree , t_tree と入力する。

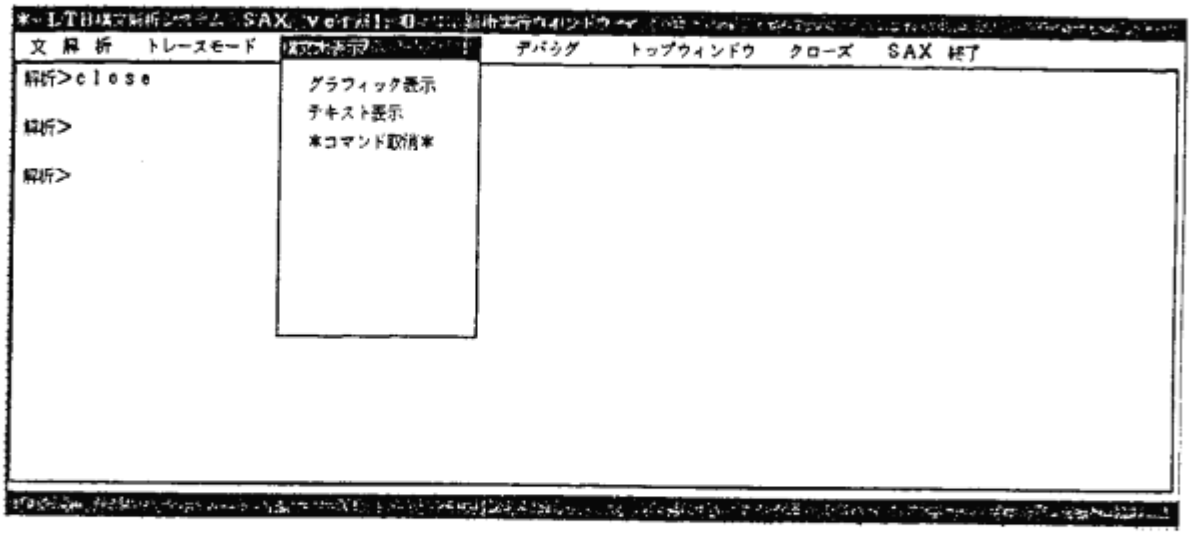


図 2.14: 木表示サブメニュー

グラフィック表示
構文木をツリーブラウザを用いてグラフィック表示する。ツリーブラウザについては 2.5 ツリーブラウザを参照のこと。

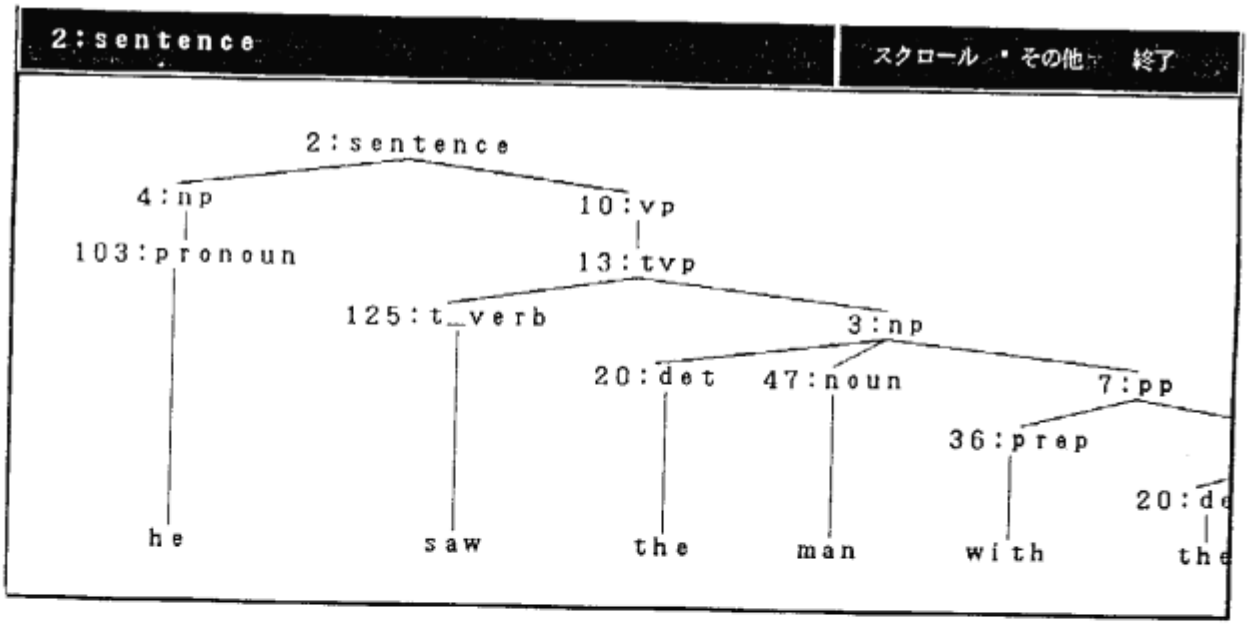


図 2.15: ツリーブラウザ

テキスト表示
構文木を実行ウィンドウにプリティプリントする。優先記述がある場合は得点表示も行なう。

意味表示

トップノードの引数のウィンドウ表示, ファイル出力を行なう。まだ解析がなされてない場合 (プロンプトが '実行>' の時) に選択するとコマンドエラーになる。解析の結果, 解が 1 つもない場合は何もしない。

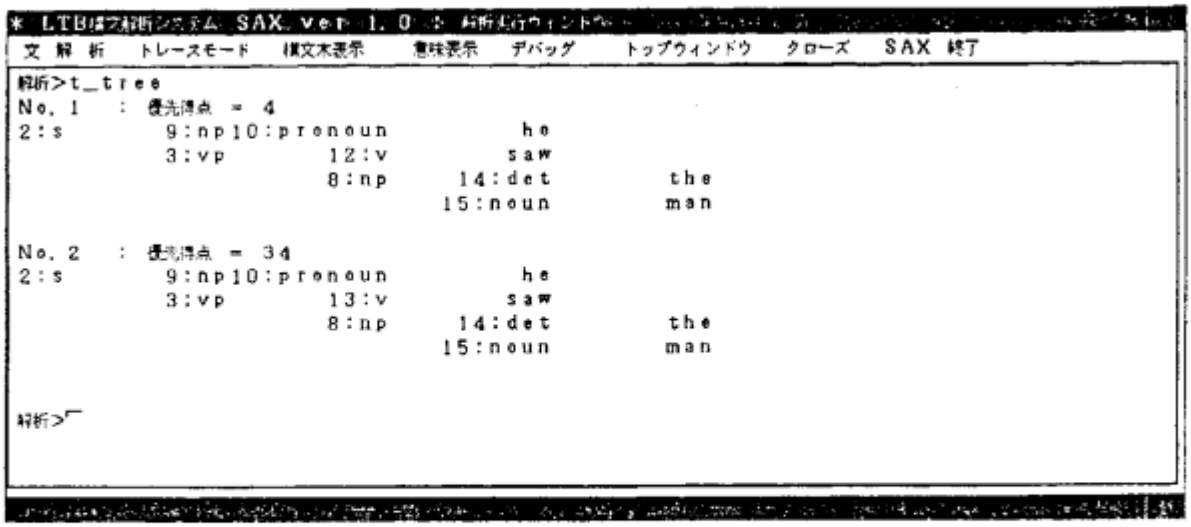


図 2.16: 構文木のテキスト表示

メインメニューで意味表示を選択するとサブメニューが表れるので, ファイル出力・pp ウィンドウ・pp ファイルから選択する。
コマンド入力はそれぞれ sem_out , sem_pp_w , sem_pp_f 。

図 2.17: 意味表示サブメニュー

ファイル出力
まず解選択メニューが表れるので出力する解を選択する。選択した解の引数情報をファイル "sax.out" に出力する。

pp ウィンドウ
トップノードの引数をウィンドウにプリティプリントする。
まず解選択を行い, 選択された解の引数情報を CIL プリティプリントする。

pp ファイル

図 2.18: 解選択メニュー

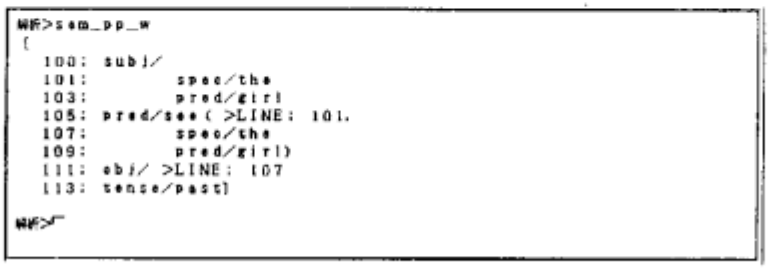


図 2.19: ウィンドウへの意味構造のプリティプリント

トップノードの引数をファイル "sax.pp" に CIL プリティプリントする。
まず解選択を行い, 選択された解の引数情報を CIL プリティプリントする。

デバッグ

構文解析結果をもとにビジュアルデバッガを起動してデバッグを行なう。まだ解析がなされてない場合 (プロンプトが '実行>') の時に選択するとコマンドエラーになる。
メインメニューでデバッグを選択する。コマンド入力時は debug と入力する。
ビジュアルデバッガの操作については ビジュアルデバッガを参照のこと。

トップウィンドウ

実行ウィンドウを終了しトップウィンドウに戻る。
メインメニューでトップウィンドウを選択するか,return とコマンド入力する。

クローズ

実行ウィンドウをクローズする。
メインメニューでクローズを選択するか,close とコマンド入力する。

終了

実行ウィンドウを終了する。
メインメニューで終了を選択するとサブメニューが表れるので, 終了・強制終了のいずれかを選択する。コマンド入

力はそれぞれ exit,abort である。

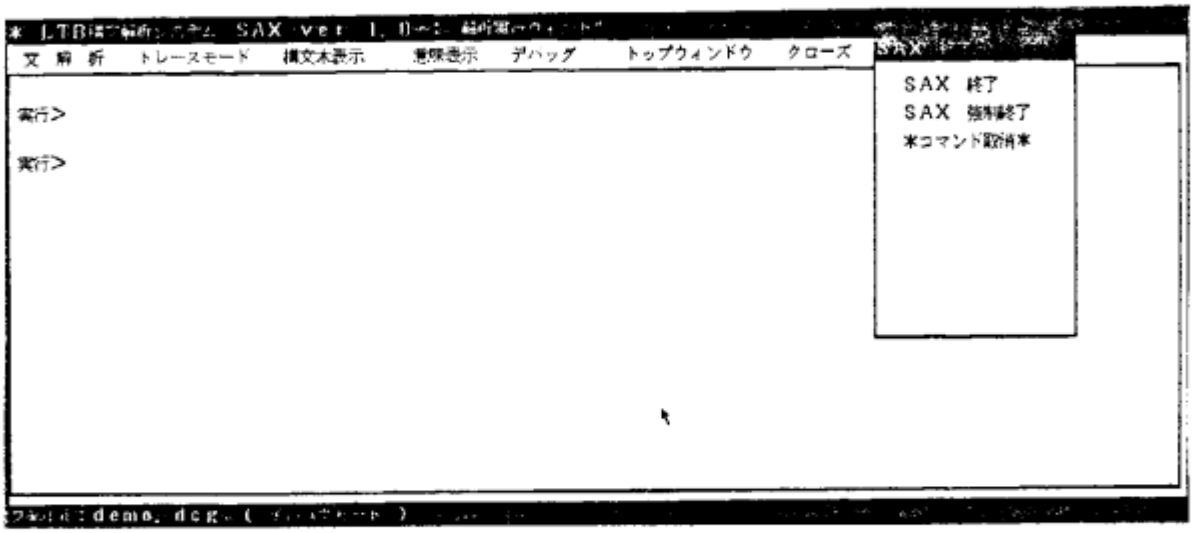


図 2.20: 実行ウィンドウ終了メニュー

終了

実行ウィンドウ終了後,SAX システムも終了する。実行ウィンドウが消えた後トップウィンドウがあらわれ終了確認メニューがあらわれる。ここで 'はい' を選択すると SAX システム終了となる。'いいえ' を選択するとトップウィンドウに制御がもどる。

強制終了

実行ウィンドウ終了後,SAX システムを強制終了する。実行ウィンドウが消えた後トップウィンドウがあらわれ強制終了確認メニューがあらわれる。ここで 'はい' を選択すると SAX システム強制終了となる。'いいえ' を選択するとトップウィンドウに制御がもどる。

その他のコマンド

メインメニューに表示されてないその他のコマンドとして

別名設定コマンド　define,
別名一覧コマンド　show_alias,
別名削除コマンド　undefine

がある。これらのコマンドの用法はトップウィンドウと同じである。
別名はトップウィンドウと実行ウィンドウで別々に管理しているので, トップウィンドウと実行ウィンドウで同じ別名を使用しても構わない。実行ウィンドウでの別名もトップウィンドウを通常終了 (強制終了ではない) することにより環境ファイルに保存される。

2.4 ツリーブラウザ

ツリーブラウザは構文解析の結果得られた構文木情報をグラフィック表示するツールであり実行ウィンドウから起動される。
まず木構造の縮小表示によるメニューウィンドウがあらわれる。ここで選択した木構造をブラウジングウィンドウに描画する。ブラウジングウィンドウはスクリーン上にいくつでもつくることができるので構文木が複数ある時など別々のブラウジングウィンドウに表示して比較することができる。

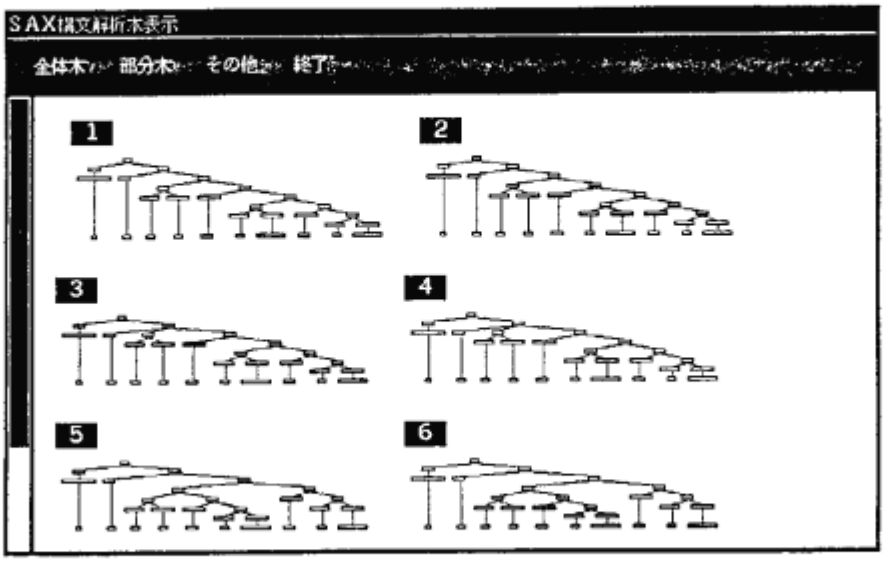


図 2.21: メニューウィンドウ (スクロール前)

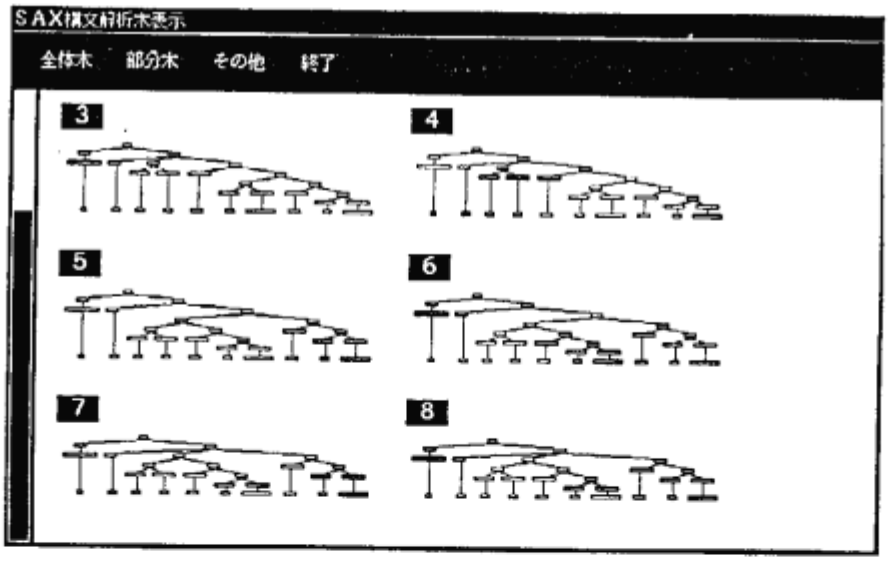


図 2.22: メニューウィンドウ (スクロール後)

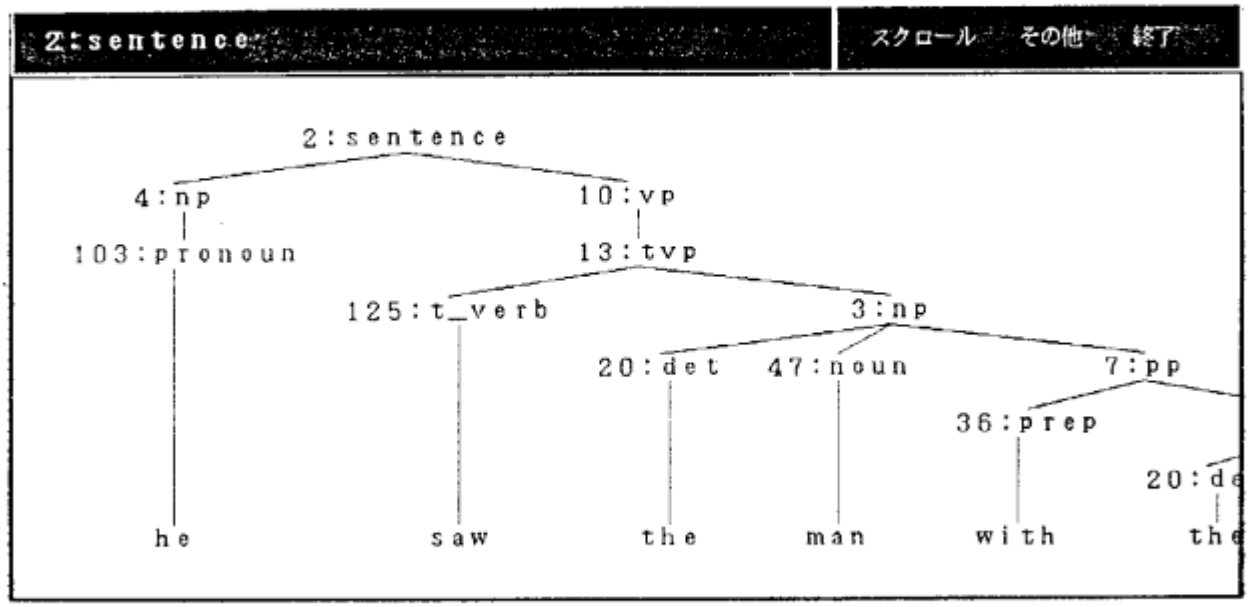


図 2.23: ブラウジングウィンドウ

2.4.1 メニューウィンドウ

メニューウィンドウは上部のコマンドメニューと下部の木メニューと木メニューをスクロールさせるためのスクロールウィンドウからなる。

木メニューには構文木が縮小表示されており, スクロールウィンドウには現在表示されている割合が反転表示されている。メニューをスクロールさせるにはマウスをスクロールウィンドウにもっていく。すると反転表示されている部分の線画がでるので, マウスを上下に動かしクリックすればよい。
メニューウィンドウではまずコマンドメニューでコマンド選択しなければならない。選択したコマンドは反転表示となる。
以下にメニューウィンドウでのコマンドを説明する。

全体木

木メニューで木構造の選択を行い, その木構造をブラウジングウィンドウに描画する。木構造の選択はメニュー上でマウスカーソルを木構造のある位置に移動させるとその木構造がボックス表示されるのでそのときマウスクリックをすればよい。
ブラウジングウィンドウが未生成であればウィンドウの生成を行なう。このときマウスカーソルが'「'となるのでマウスでウィンドウの位置と大きさを決定する。
もし選択した木構造がすでに表示中であればベルをならしその木構造が描画されているウィンドウをスクリーンの最上層に表示する。
なおブラウジングウィンドウ表示中でもメニューは入力可能なので別のコマンドを実行することが出来る。

部分木

木構造中の任意のノードをルートノードとする部分木構造をブラウジングウィンドウに描画する。ノードの選択はマウスカーソルを木構造が表示されている領域に移動させその中でノードをあらわすボックスをマウスクリックすればよい。
ボックス外でマウスクリックを行うと部分木コマンドの取消となる。

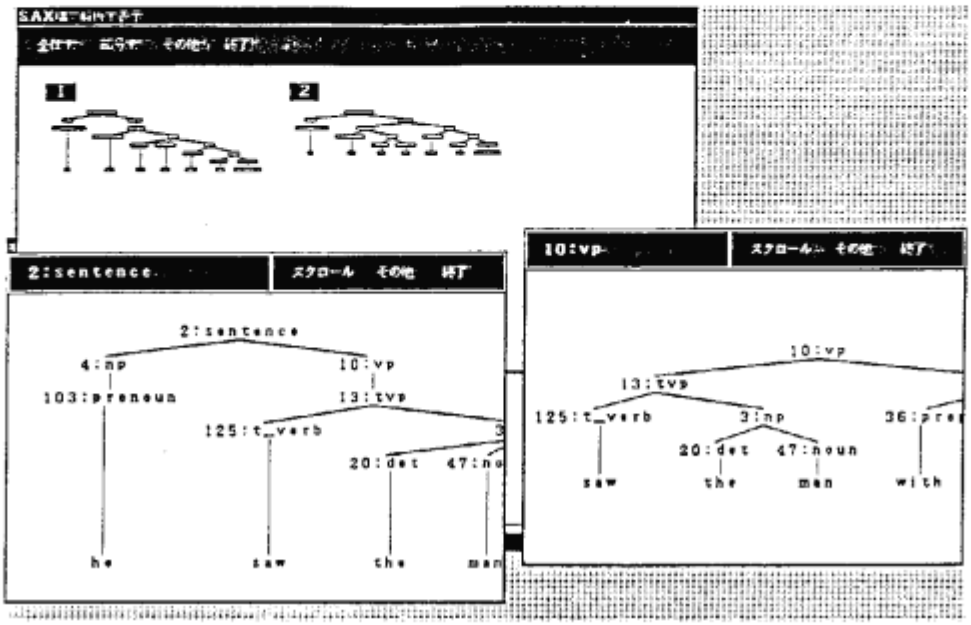


図 2.24: 全体木, 部分木の表示

その他

その他を選択すると下図のサブメニューがあらわれる。

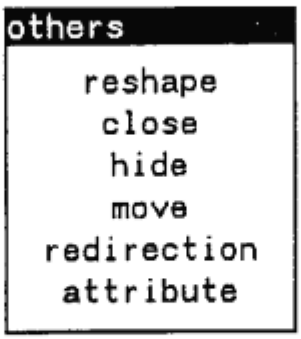


図 2.25: ツリーメニューのサブメニュー

サブメニューでのコマンドは以下の通りである。

- reshape

メニューウィンドウのリシェイプを行なう。
メニューウィンドウの再生成をおこなうので現在のウィンドウザイズよりも小さくすることができる。

- close
  メニューウィンドウをクローズする。

- hide
  メニューウィンドウとすべてのブラウジングウィンドウをスクリーンの最下層に置く。

- move
  メニューウィンドウの位置をマウスで移動する。

- redirection
  ウィンドウ出力様式を変更する。
  出力様式変更ウィンドウで値の変更を行う。

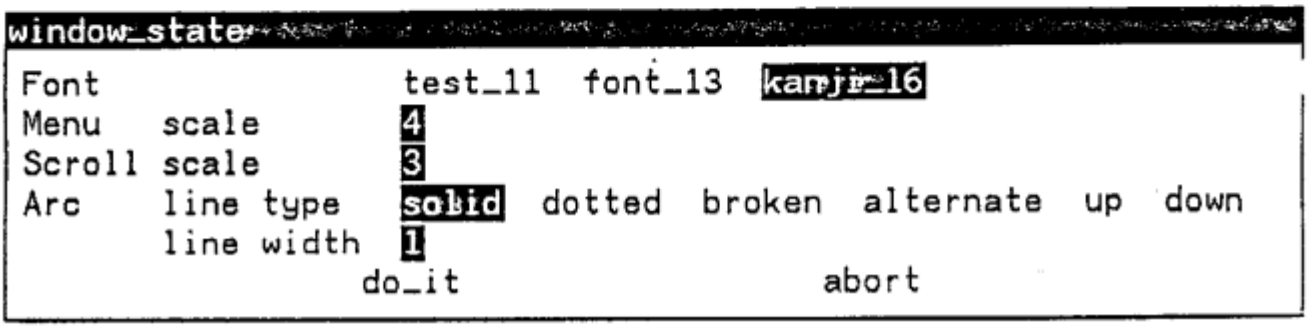


図 2.26: 出力様式変更ウィンドウ

- font
  メニューウィンドウ, ブラウジングウィンドウの出力フォントを変更する。

- menu scale
  縮小メニューの縮小率 (1/n) を変更する。
  値は 2 から 15 の整数で指定する。

- scroll scale
  ブラウジングウィンドウのスクロールウィンドウの縮小率 (1/n) を変更する。
  値は 2 から 15 の整数で指定する。

- arc line type
  ブラウジングウィンドウの各ノード間を結ぶアークの線種を変更する。

- arc line width
  ブラウジングウィンドウの各ノード間を結ぶアークの線幅を変更する。
  整数で指定する。

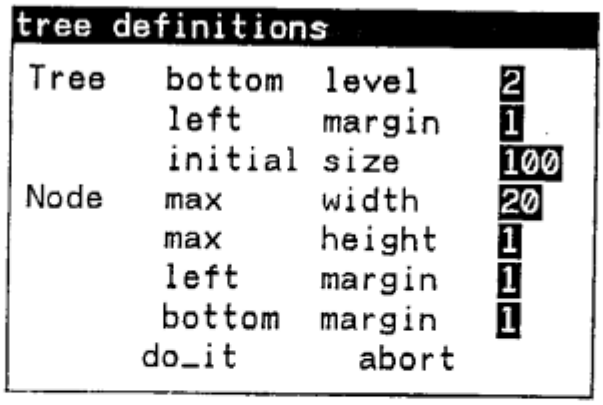


図 2.27: 定義情報出力ウィンドウ

- attribute
  木構造およびノードの出力定義情報を変更する。通常ユーザはこの情報を意識しなくてもよい。
- tree bottom level
  位置合わせを行なうノードのレベル
- left margin
  木全体の左側マージン
- initial size
  ノード情報を格納するプールの初期サイズ
- node max width
  ノード名出力最大幅 (文字単位)
- max height
  ノード名出力最大高さ (文字単位)
- left_margin
  ノード出力領域の左側マージン (文字単位)
- bottom_margin
  ノード出力領域の下側マージン (文字単位)

終了

ツリーブラウザを終了し実行ウィンドウにもどる。

2.4.2 ブラウジングウィンドウ

メニューウィンドウでの全体木, 部分木を選択するとブラウジングウィンドウがあらわれる。またビジュアルデバッガで部分木表示を選択した場合もブラウジングウィンドウを使用する。
ウィンドウはタイトルウィンドウ, コマンドメニュー, ビューポートからなる。
タイトルウィンドウにはルートノード名を表示する。全体木であればルートノードはトップノード指定した非終端記号となる。終端記号の前にはその文法規則番号を表示する。
ビューポートには部分木構造を表示する。木構造が大きくてウィンドウに一度に表示しきれない場合はスクロールコマンドで表示領域を変更することができる。
木構造の表示形式は bottom_level の値により異なる。bottom_lebel が 0,1,2 のときの表示をそれぞれ図 2.28, 図 2.29, 図 2.30 に示す。

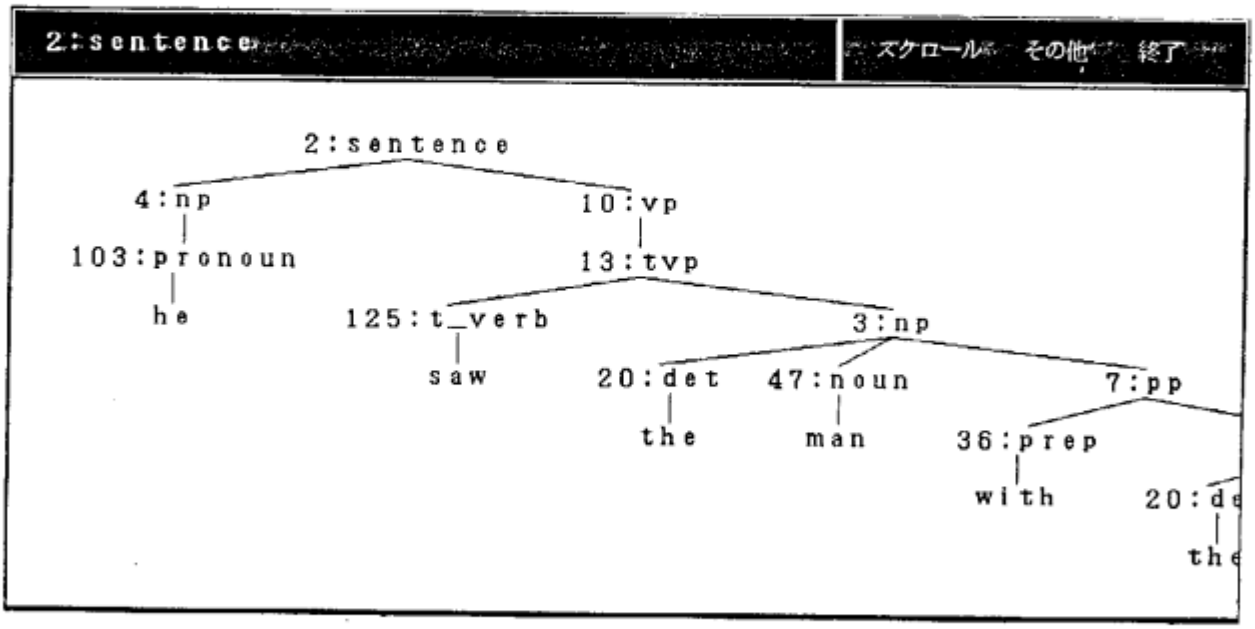


図 2.28: bottom_lebel が 0 のときの木表示

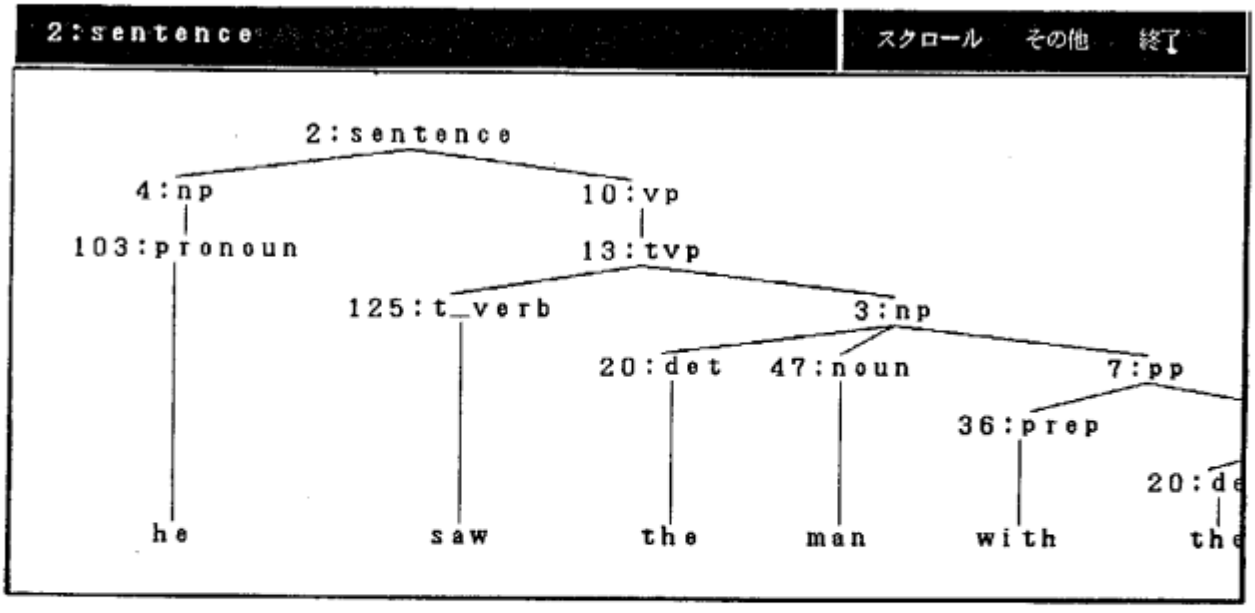


図 2.29: bottom_lebel が 1 のときの木表示

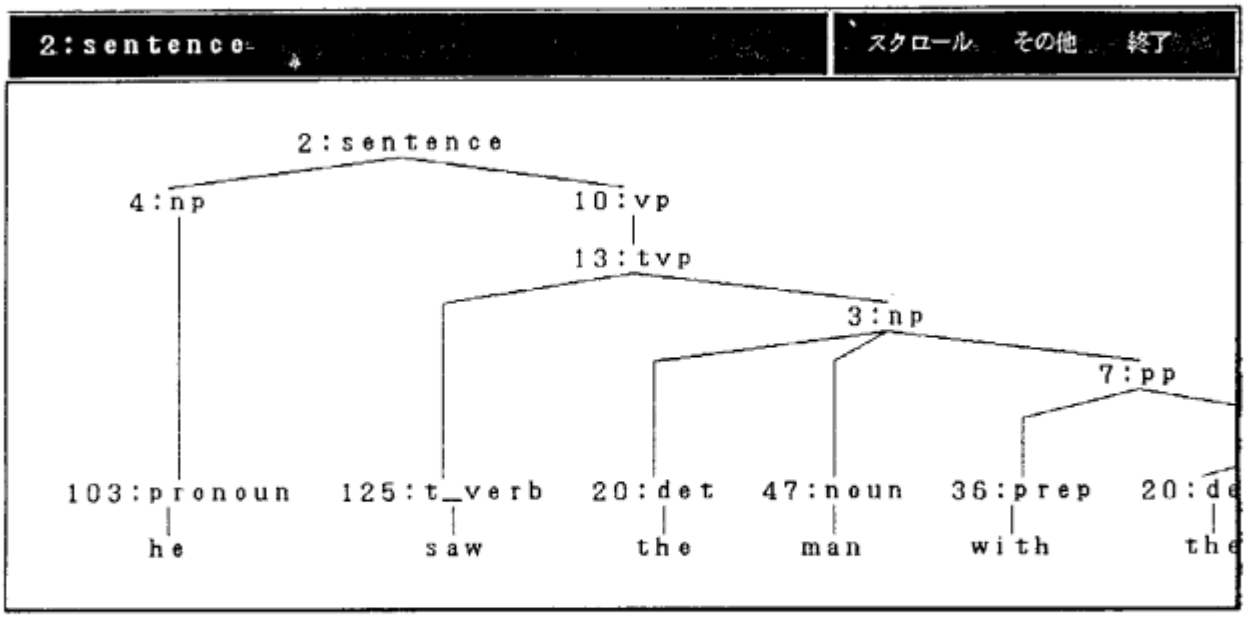


図 2.30: bottom_lebel が 2 のときの木表示

ブラウジングウィンドウでのコマンドを以下に説明する。

スクロール

スクロールウィンドウを用いてビューポートのスクロールを行なう。
スクロールウィンドウが未生成であればウィンドウの線画があらわれるのでまずマウスで位置決めを行なう。スクロールウィンドウにはビューポートに表示されている領域を反転表示している。マウスカーソルをスクロールウィンドウ上に移動さすと矩形のジャンプスクロールマーカがあらわれるので, マウスでこのマーカを移動させる。マウスクリックすることによりマーカで囲まれた領域をビューポートへ表示する。

その他

ウィンドウ状態を変更させるためのコマンドで, サブメニューで値の変更を行なう。

- reshape
  ブラウジングウィンドウのリシェイプを行なう。
- close
  ブラウジングウィンドウをクローズする。
- hide
  ブラウジングウィンドウをスクリーンの最下層におく。
- move
  ブラウジングウィンドウを移動させる。

終了

ブラウジングウィンドウをスクリーンから消す。表示上消えるだけでメニューで全体木・部分木を選択した場合はこのウィンドウを再利用する。

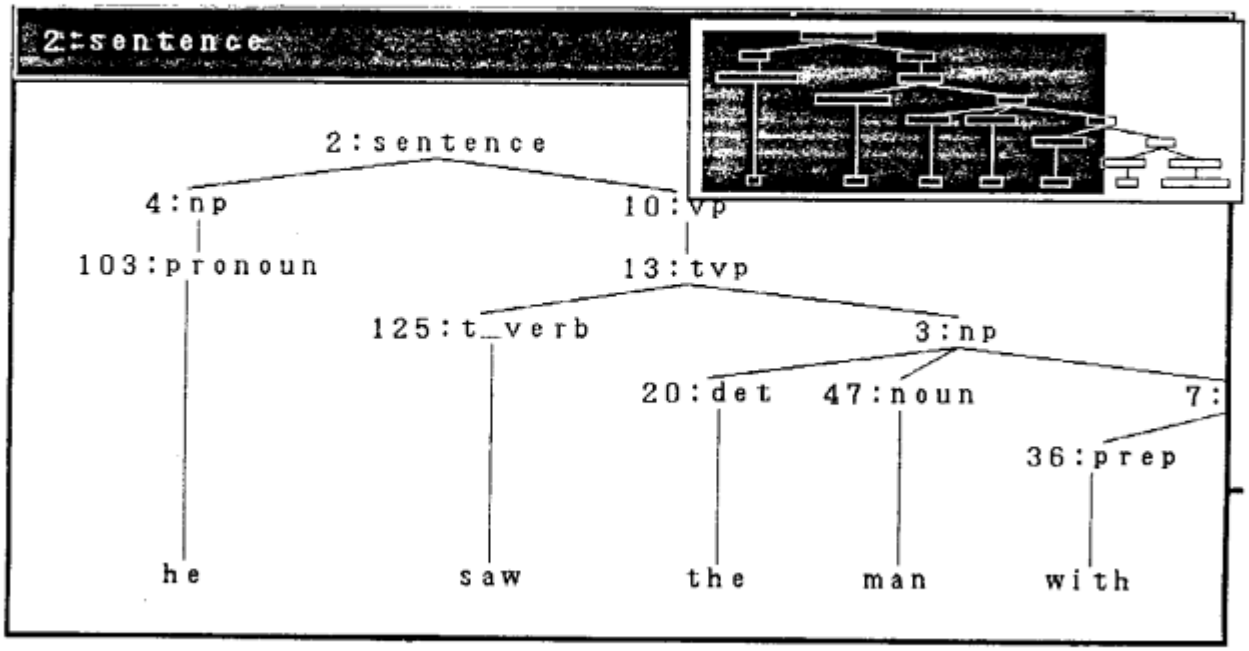


図 2.31: スクロール前のブラウジングウィンドウ

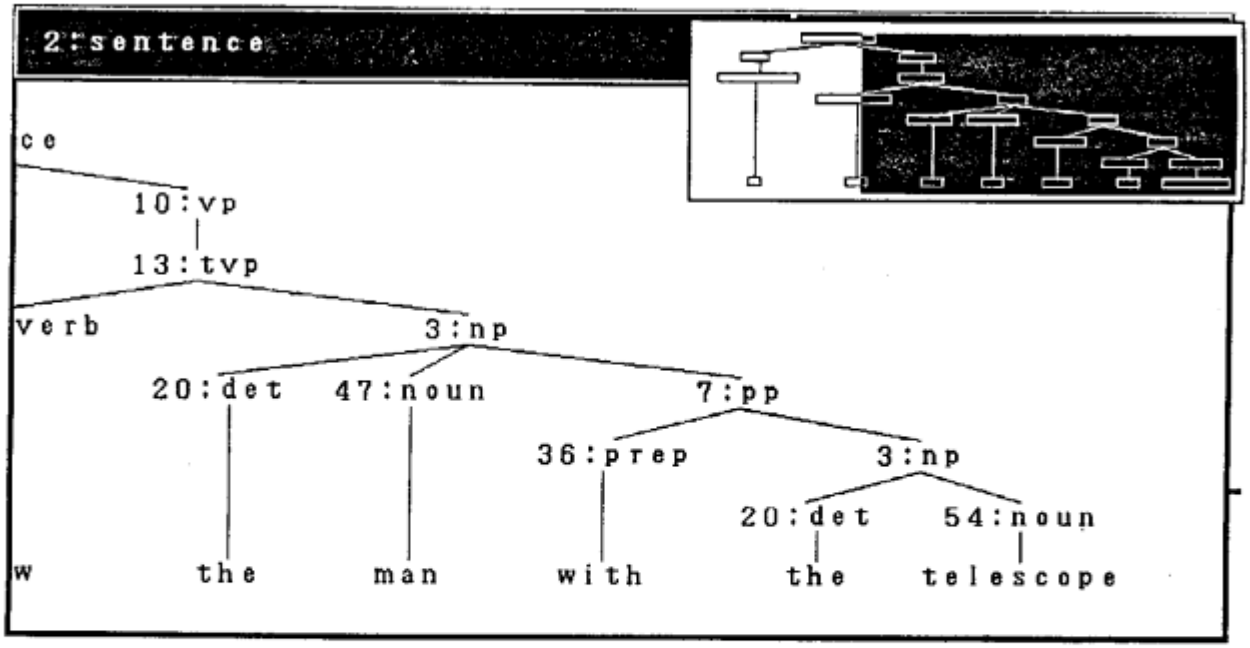


図 2.32: スクロール後のブラウジングウィンドウ

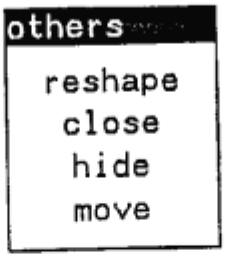


図 2.33: サブメニュー

2.5 トレーサ

SAX システムではデバッグ環境として動的デバッグ環境と静的デバッグ環境を備えている。トレーサは動的なデバッグ環境であり SAX の解析実行過程をトレースすることにより主に補強項評価の失敗を発見するために使用される。[9]

2.5.1 機能

トレーサでは以下の情報を得ることが出来る。

- 識別子出力情報
  タイプ1, タイプ2ミドルの識別子の出力時の識別子を表示する。識別子はその名前が文法規則中のユニークな位置を表すので, これを基にその識別子が存在する文法規則をとりだし, 識別子の位置に '*' 記号を埋め込んだ形で表示する。

- 終端記号呼び出し
  入力文中の実行位置を示す。 LAX からの入力を解析する場合は終端記号は存在しない。

- 非終端記号呼び出し
  非終端記号の引数情報をみることが出来る。 CIL プリティプリンタを用いての表示や CIL インスペクタ,SIMPOS のインスペクタによる引数のインスペクトも行える。

- 補強項評価
  評価前, 評価後の補強項を表示することが出来る。表示は標準出力 :putt によるが CIL 記述がある場合は CIL プリティプリントを行なうことができる。また CIL インスペクタ,SIMPOS のインスペクタによる補強項のインスペクトが行える。
  スパイポイントは文法規則, 補強項, 終端記号, 非終端記号, 入力文中の終端記号 (LAX からの入力を解析する時は入力ゴール中の非終端記号) に設定することが出来る。制御コマンドとしては step , leash ,no_trace がある。 SAX は決定的に解析を行いバックトラックすることはないので retry することはできない。ただし補強項の評価に失敗した場合のみ retry コマンドがあるが, 他の選択肢を実行するのではなく評価に失敗した補強項を再実行するだけである。

2.5.2 起動

実行ウィンドウでトレースモードに設定して最初の解析を行なう際に, トレーサウィンドウの線画があらわれるので, マウスで位置決めを行なう。補強項中に CIL 記述があり CIL インタープリタで補強項評価を行なう場合は CIL デバッガを使用するので,CIL デバッガの位置決めも同様に行なう。

[9] 静的デバッグ環境については 2.7 で述べる。

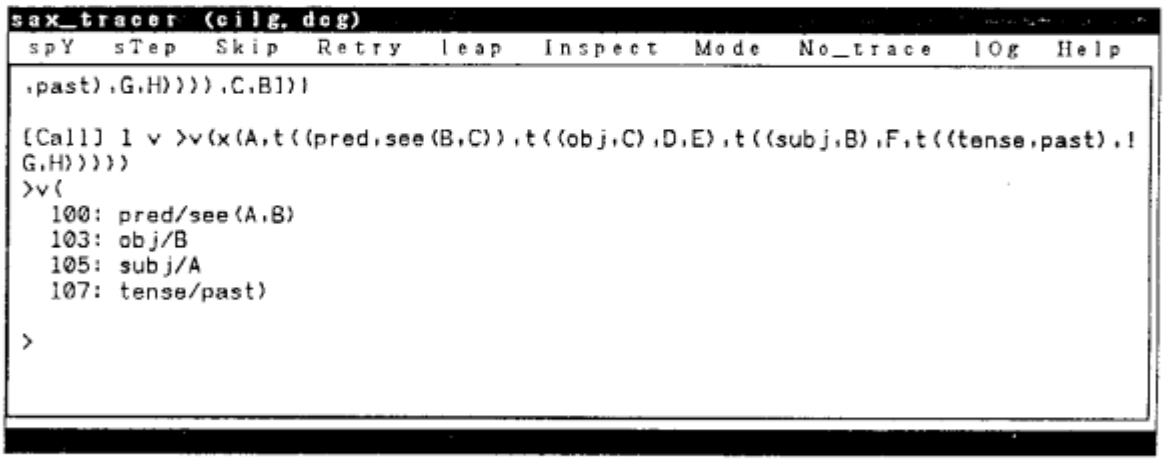


図 2.34: トレーサウィンドウ

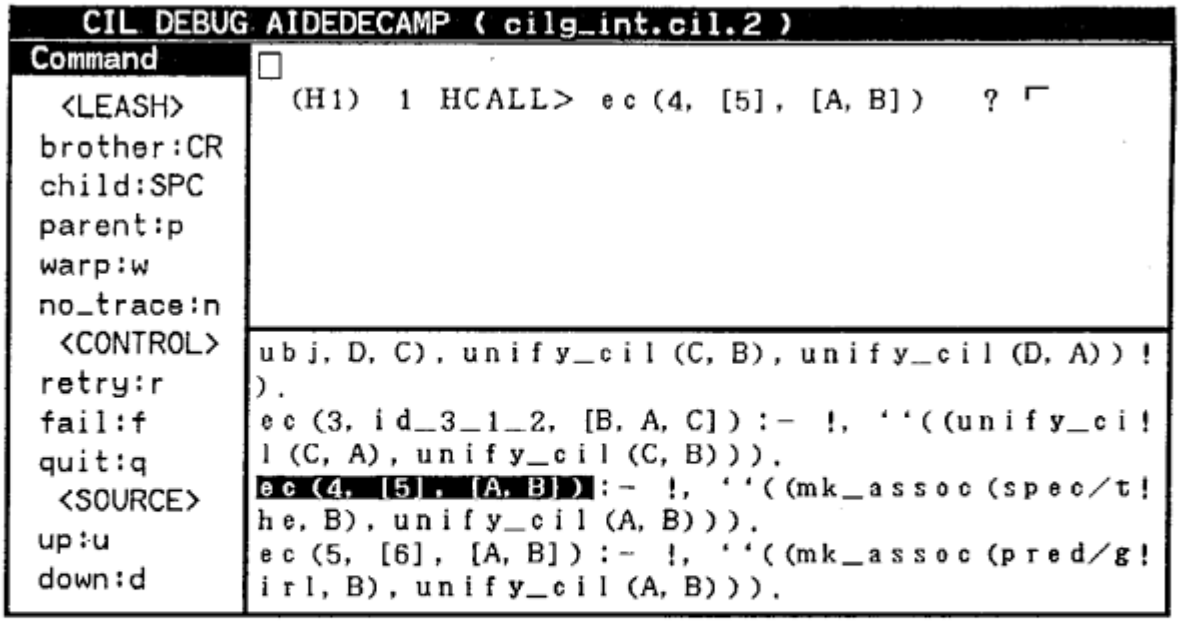


図 2.35: CIL デバッガ

2.5.3 トレースとリーシュ

SAX トレーサは, 識別子出力状況・終端記号・非終端記号・補強項評価でトレース表示・リーシュ (中断) をおこなう。トレース指定されている情報は実行中にその情報に相当するメソッドの呼び出し時にトレース表示する。トレース指定されてリーシュ指定されている場合はトレース情報表示後コマンド入力待ちとなる。

各情報のトレース表示を以下に示す。

識別子出力情報:
識別子が存在する文法規則を識別子の位置に '*' 記号を埋め込んだ形で表示する。
例:

```
np--> (det;[]),noun,((*pp;*coconj,np);[]),(relc;[])
```

終端記号:

[Terminal Call] N 終端記号 % メソッド呼び出し時
[Terminal Exit] N 終端記号 % メソッド終了時

N は入力文中の並びの通番をあらわす。

非終端記号:

[Head Call] N 終端記号 % メソッド呼び出し時
[Head Exit] N 終端記号 % メソッド終了時

N は呼び出しの深さをあらわす。

補強項:

[extC] ID 補強項 % メソッド呼び出し時
[extC Exit] ID 補強項 % メソッド終了時

ID は補強項の識別番号。
補強項の表示は詳細モードのとき行なう。
詳細モードでないときは識別番号のみ表示する。

2.5.4 コマンド

トレーサでのコマンド入力はコマンドメニューのマウス選択, キーボードからの一文字入力で行なう。ただし一部のコマンドはキーボード入力のみ有効。

- **制御コマンド**

  1. ステップ : メニュー で 'step' を選択。 スペース・リターンキー入力
     ステップ実行。次のリーシュポイントで中断する。
  2. リープ : メニューで 'Leap' 選択。 "L" 又は "l" の入力。
     次のスパイポイントまでトレース・中断なしで実行する。
  3. スキップ : メニューで 'Skip' 選択。 "S","s"の入力。
     Exit までトレース中断なしで実行する。
     適用は終端記号, 非終端記号の call 時のみ。
  4. ノートレース : メニューで 'No_Trace' 選択。 "N","n"の入力。
     トレース出力を中止する。トレーサウィンドウ,CIL デバッガは消える。

5. 再実行：メニューで 'Retry' 選択。"R","r"の入力。
   補強項の再評価を行なう。他の選択肢を実行するのではなく失敗したゴール列の再評価を行なう。
   適用は補強項失敗時のみ。

   トレーサ状態設定用コマンド

6. モード：メニューで 'Mode' 選択。"M","m"の入力。
   トレース, リーシュポイント, ホロフラスティング等の設定を行なう。

   2.6.5 モードメニューで後述。

7. スパイ：メニューで 'spY' 選択。"Y","y"の入力。
   スパイポイントの設定を行なう。6.3.5 スパイ設定ウィンドウで後述。
   情報表示用コマンド

8. インスペクト：メニューで 'Inspect" 選択。"I","i"の入力。
   非終端記号の引数情報, 補強項ゴール列のインスペクトを行なう。CIL 使用時は CIL インスペクタを, 未使用時は SIMPOS のインスペクタを利用する。
   適用は非終端記号, 補強項評価時。

9. プリティプリント：メニューで 'Pp' 選択。"P","p"の入力。
   終端記号・非終端記号の引数情報, 補強項の CIL プリティプリントを行なう。
   適用は識別子出力以外。

10. 引数表示："A","a"の入力
    終端記号・非終端記号の引数表示を行なう。表示は :putt による。
    適用は非終端記号のみ。

11. 全情報表示："F","f"の入力
    終端記号, 非終端記号でメソッド内の全ての引数を表示する。
    適用は終端記号, 非終端記号のみ。

12. ログ：メニュー 'lOg' 選択。"O","o"の入力。
    トレーサのログバッファ "sax_tracer.log" にトレース情報を出力する。
    終端記号・非終端記号では全情報を出力する。補強項では補強項を出力する。

13. ヘルプ："H","h","?"の入力
    コマンド一覧を表示する。

2.5.5 モードメニュー

モードメニューではトレース・リーシュの設定, 補強項表示モード, ホロフラスティングレベルの設定を行なう。
モードメニューは選択メニューでトレース・リーシュは反転表示が設定されていることを示す。マウスクリックすることにより on は off に,off は on になる。トレース・リーシュともに off のとき, リーシュを on にするとトレースも on となる。ともに on のとき, トレースを off にすると リーシュも off となる。
補強項表示モード (Extra condition detail) はオンオフメニューで,on のとき表示モード,off のとき非表示モードとなる。
ホロフラスティングの深さ・長さを変更するには, 各項目をマウスクリックするとキーボード入力可能となるので, 数値を入力したあと改行で終える。数値以外が入力された場合は無視する。

2.5.6 スパイ設定ウィンドウ

スパイ設定ウィンドウはトレーサでのスパイポイントの設定・解除を行う。
このウィンドウは図 のように 4 つのウィンドウから構成される。

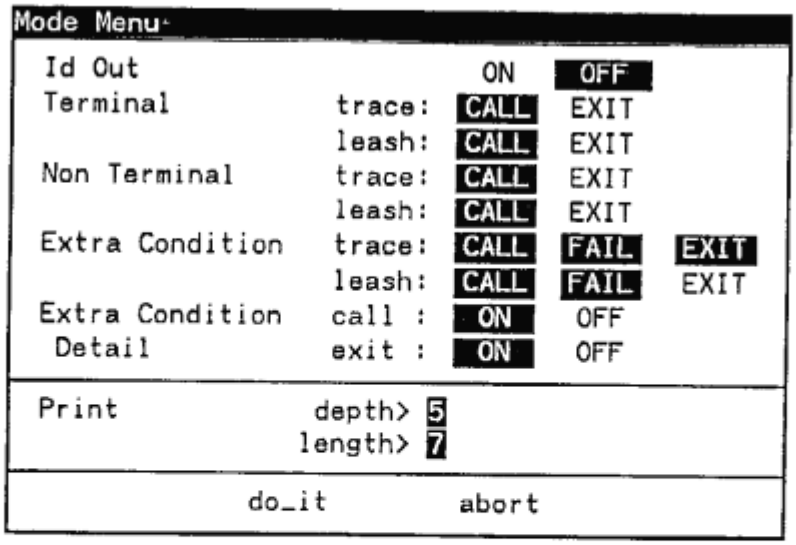


図 2.36: モードメニュー

スパイ設定ウィンドウは解除モードと設定モードがあり，解除モードでスパイポイントの解除，設定モードでスパイポイントの設定を行う。

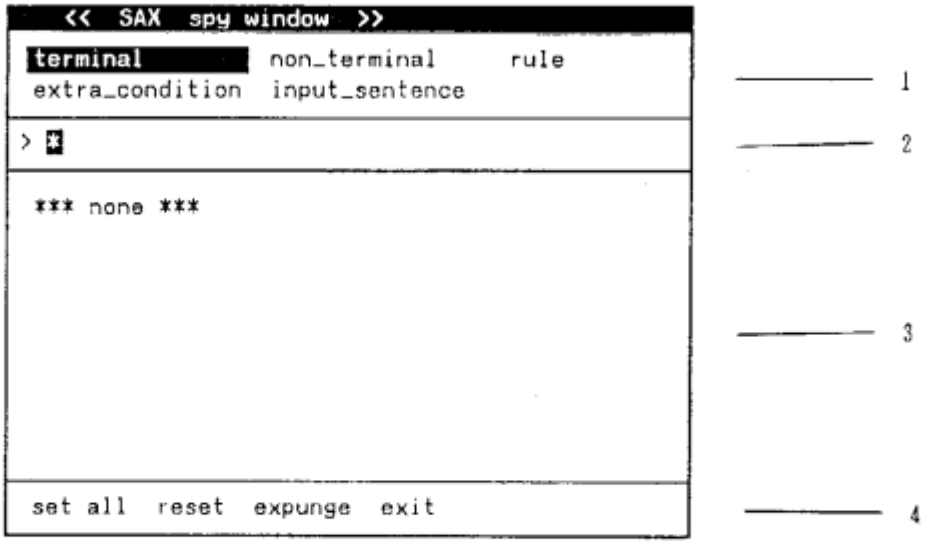


図 2.37: スパイ設定ウィンドウ

各ウィンドウの機能

1. スパイポイントの種類を選択する。

2. 非反転表示の時スパイポイント解除モードをあらわす。
   起動直後は解除モードになっている。スパイ名に対するワイルドカードを入力すると, スパイポイント設定モードとなり反転表示になる。

3. スパイポイントメニュー
   マウスクリックにより反転・非反転をくりかえす。
   解除モードのとき設定したスパイポイント名を表示し, 設定モードのときワイルドカードに対応するスパイ名を表示する。

4. コマンドメニュー

設定解除モードの時

- set_all メニュー中の項目をすべて反転表示する。
- reset
  メニュー中の項目をすべて非反転表示する。
- expunge
  反転表示されているスパイポイントの設定解除を行う。
- exit
  スパイポイント設定ウィンドウを終了する。

設定モードのとき

- do_it
  反転表示されている項目にスパイを設定し, 解除モードにもどる。

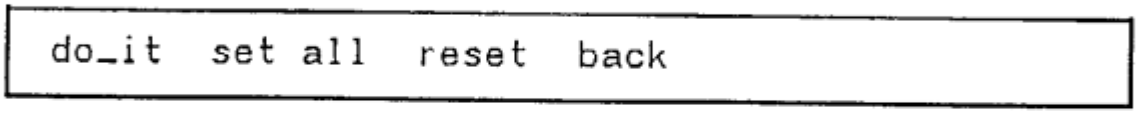


図 2.38: 設定モード時のコマンドメニュー

- set_all
  メニュー中の項目をすべて反転表示する。
- reset
  メニュー中の項目をすべて非反転表示する。
- back
  何もせずに解除モードに戻る。

スパイポイント設定方法

- (a) のウィンドウでスパイ情報を選択する。

項目名は、

terminal	: 終端記号
non_terminal	: 非終端記号
rule	: 文法規則
extra_condition	: 補強項
input_sentence	: 入力文

をあらわしている。

- (b) のウィンドウでワイルドカードを入力する。ただしスパイ情報が terminal,non_terminal 以外の時は * を入力すること。
- (c) のウィンドウにワイルドカードに一致するものが表示されるので,set_all , reset あるいはマウスクリックにより選択を行なう。選択状態にあるものは反転表示となる。
- do_it を選択する。
  解除モードとなり, メニューにはスパイポイント設定されたものだけが表示される。

スパイポイント解除方法

- (a) のウィンドウでスパイ情報を選択する。
- (b) のメニューにスパイ設定された項目が表示されるので,set_all,reset あるいは項目のマウスクリックにより反転表示させる。
- do_it を選択する。反転表示された項目が消える。

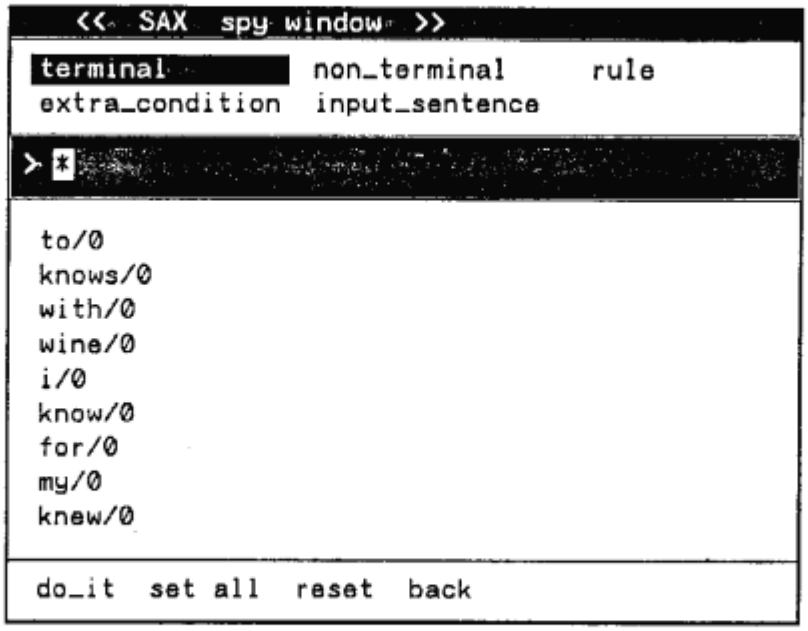


図 2.39: terminal 選択時のメニュー表示

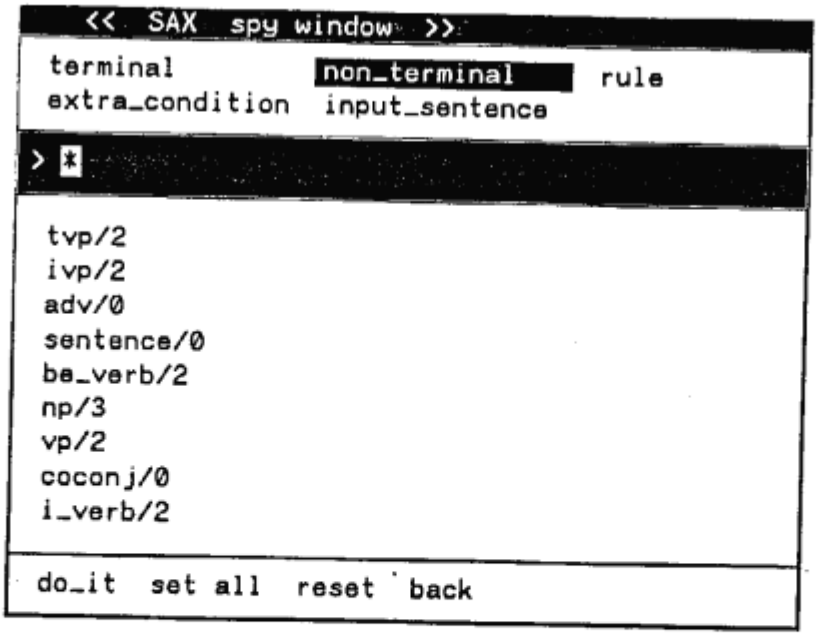


図 2.40: non_terminal 選択時のメニュー表示

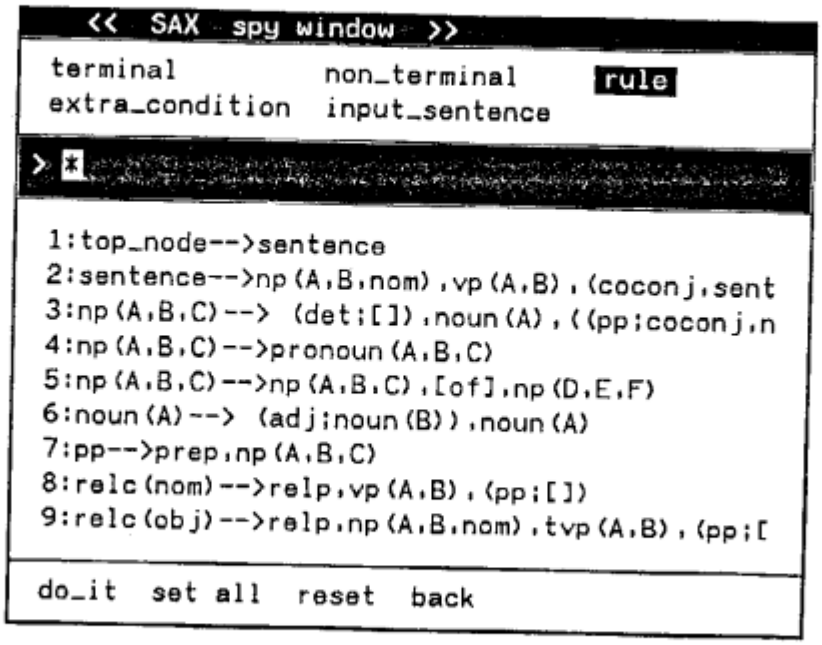


図 2.41: rule 選択時のメニュー表示

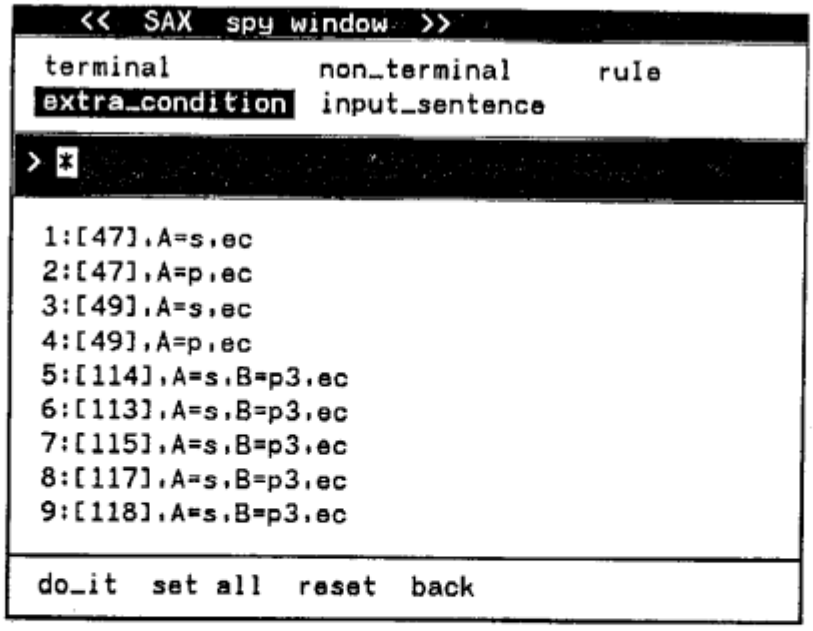


図 2.42: extra_condition 選択時のメニュー表示

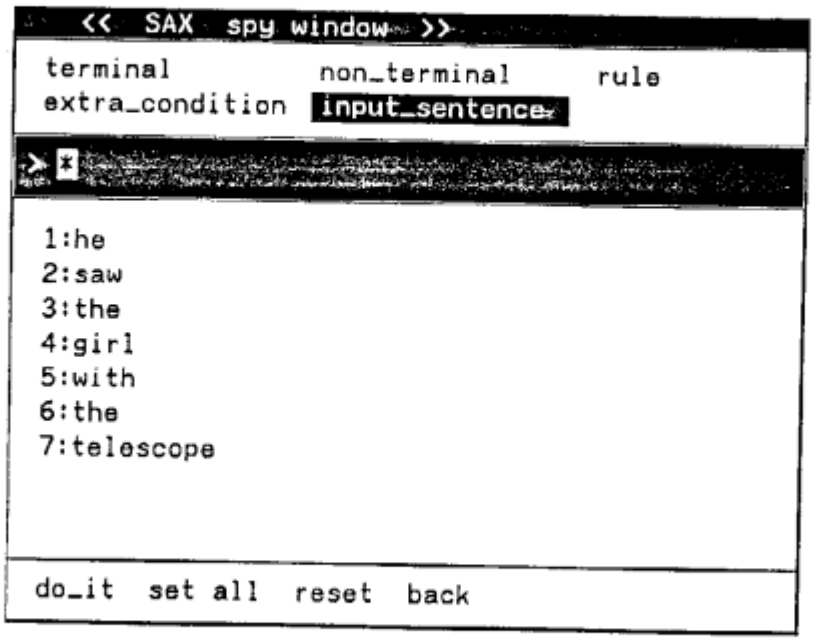


図 2.43: input_sentence 選択時のメニュー表示

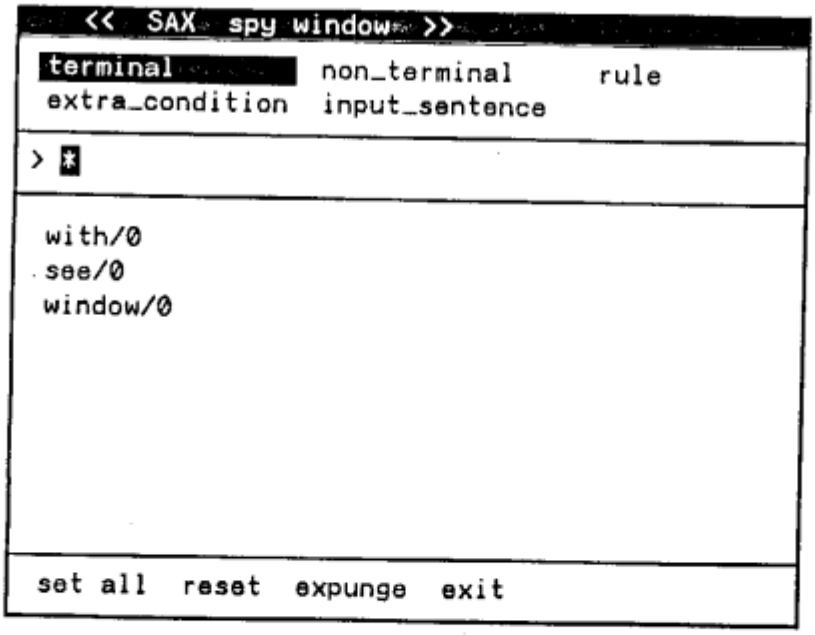


図 2.44: スパイ設定後の解除モード

2.6 ビジュアルデバッガ

ビジュアルデバッガは静的なデバッグ環境を提供するもので構文解析時の解の探索履歴を用いて, 部分木情報の表示やストリームのインスペクトを行なうものである。

2.6.1 起動

実行ウィンドウのコマンドメニューでデバッグを選択するかコマンド debug をキーボード入力することによりビジュアルデバッガの起動となる。起動すると実行ウィンドウの位置にビジュアルデバッガウィンドウを表示する。このウィンドウはコマンドメニューとグラフィック表示ウィンドウからなる。グラフィックウィンドウ下部には解析文を単語ごとに区切ってボックス表示する。このボックスはターミナルボックスと呼ぶ。LAX からの入力のときは, ボックス内上段に SAX コール非終端記号を下段にその表層語を表示する。この場合も便宜上ターミナルボックスと呼ぶ。
ボックス間の OX はボックス間のストリーム中の begin 識別子の有無を表す。O なら begin 識別子があり,X なら無いことを表す。これによりストリームの切れを発見することができる。

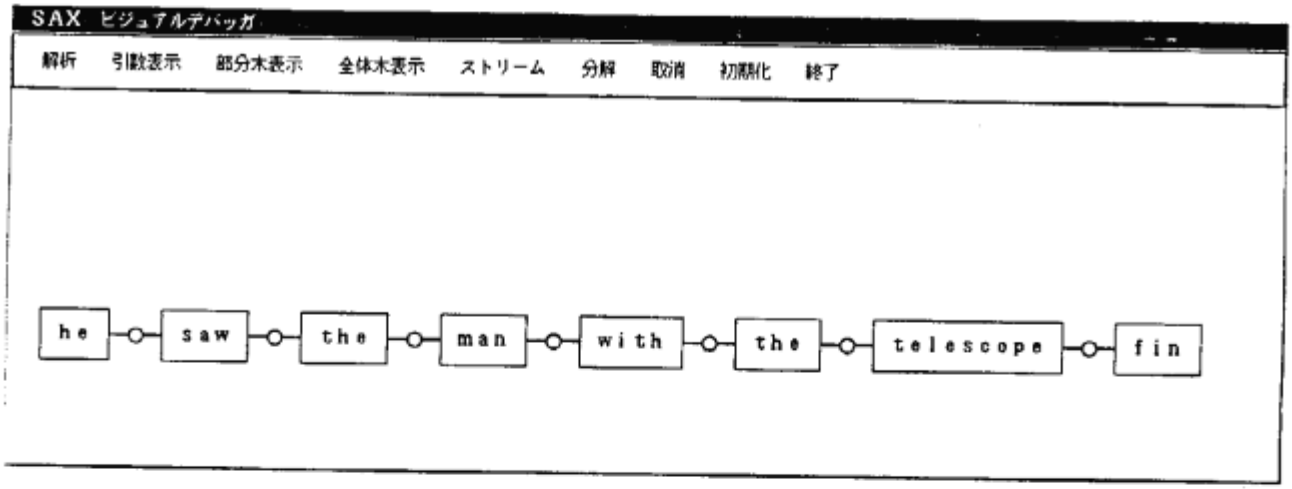


図 2.45: ビジュアルデバッガ

2.6.2 操作方法

ビジュアルデバッガでの操作は, まずコマンドメニューでコマンドを選択しグラフィックウィンドウで処理対象を選択する。以下にコマンドの説明を行なう。

解析

任意の範囲内で部分木構造の再構成をグラフィカルにおこなう。まずコマンドメニューで解析をマウスクリックした後, 解析範囲の始点・終点の 2 カ所のボックスをマウスクリックする。マウスクリックしたボックスは網かけ表

示となる。左側のボックスが始点, 右側のボックスが終点となる。範囲の指定を行うと, その範囲で完成する部分木構造のルートノード(カテゴリ)の選択メニューがあらわれる。メニュー中の項目は ' 文法規則番号: カテゴリ ' の形をしている。優先記述があるばあいはカテゴリの後にその木の構造の得点が付加される。選択メニューで選ばれたカテゴリが2つのボックス間の上に表示される。このカテゴリもボックス表示となる。このボックスはノードボックスと呼ぶ。

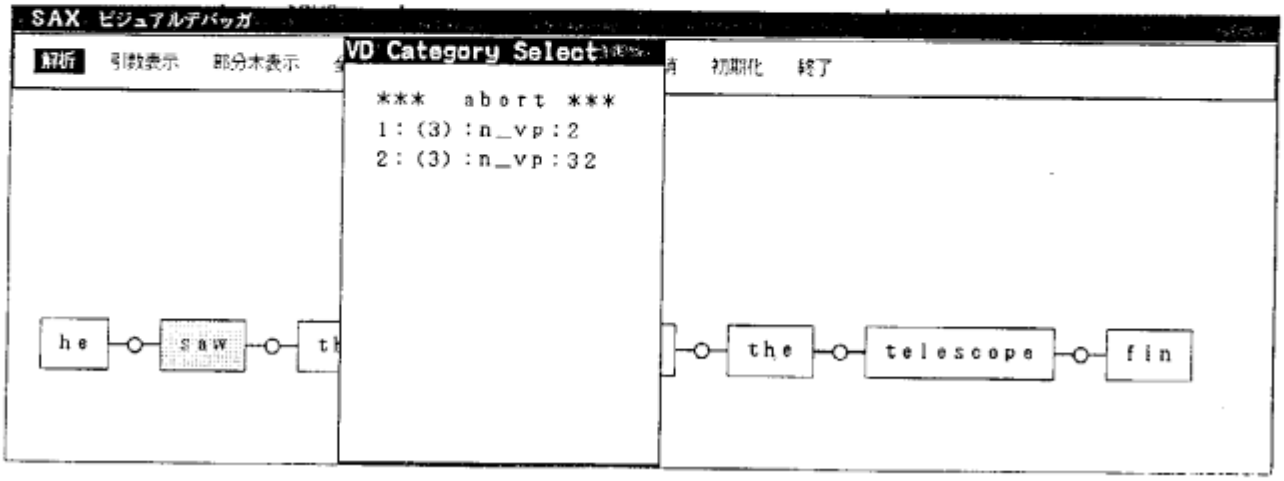


図 2.46: カテゴリ選択メニュー 1 (優先記述があるばあい)

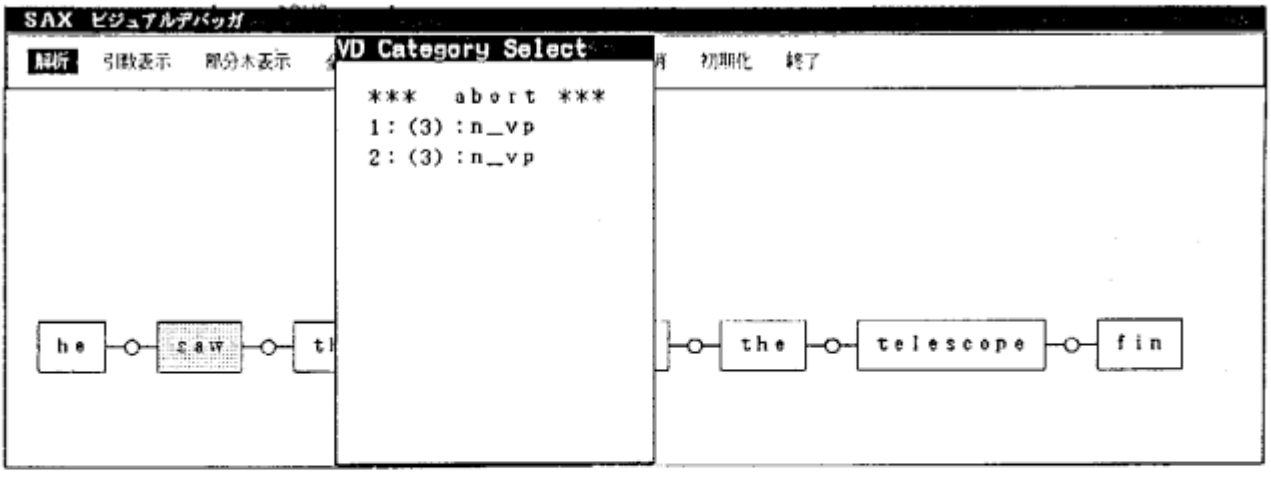


図 2.47: カテゴリ選択メニュー 2 (優先記述がないばあい)

解析でノードボックスを範囲指定した場合は, そのボックスであらわされる部分木構造を含む上位の部分木構造の構成をおこなう。このときもカテゴリ選択メニューがあらわれるので新たにボックス表示するカテゴリを選択する。カテゴリの表示は範囲指定したノードボックスを消去し, 新たな1つのノードボックスを表示する。

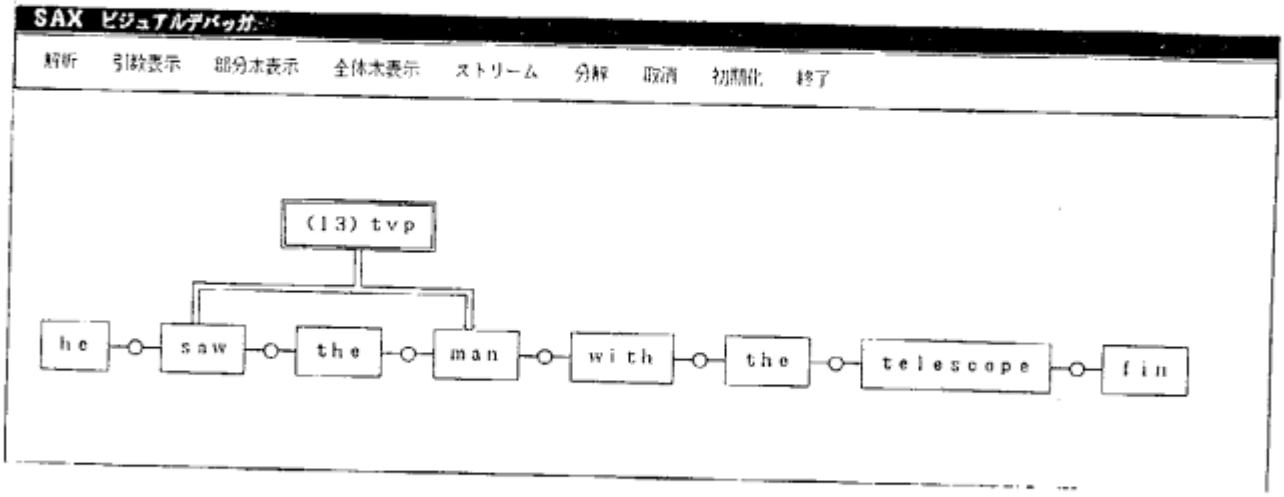


図 2.48: ノードボックスの表示

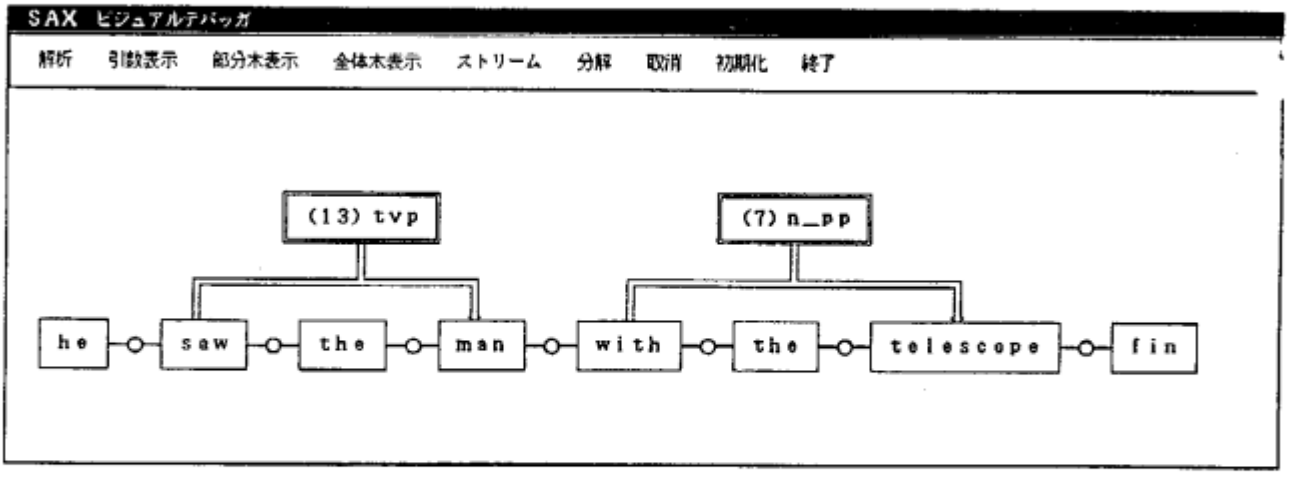


図 2.49: 2つのノードボックス表示画面

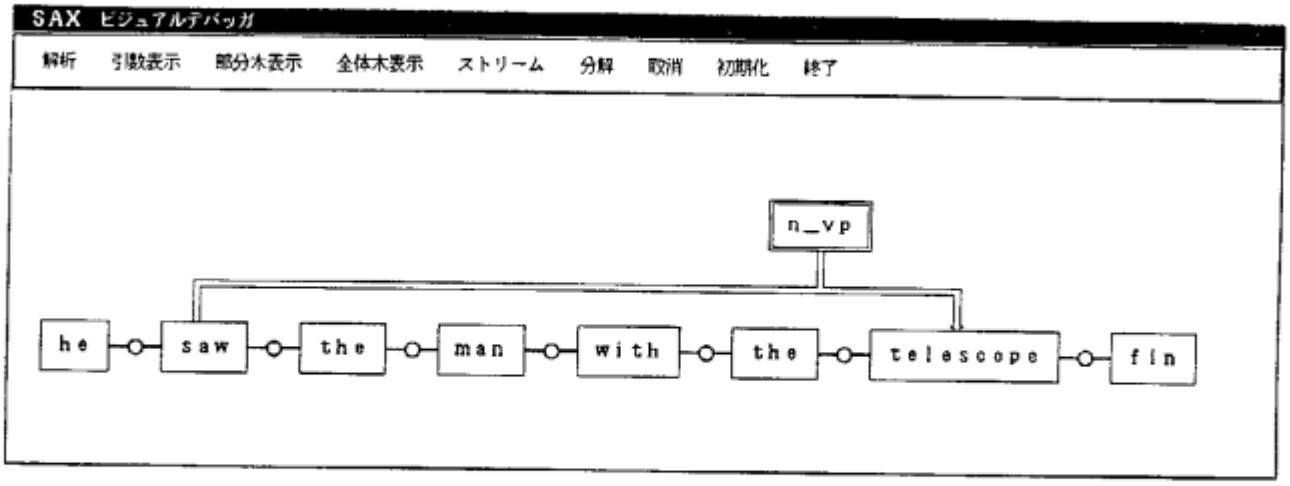


図 2.50: 新ノードボックスの表示画面

引数表示

部分木構造のルートノードの引数情報をインスペクタで調べる。LAX からの入力ゴール列のデバッグ時は SAX コールゴールのインスペクトも行える。CIL 使用時には CIL インスペクタを, それ以外の場合は SIMPOS のインスペクタを用いる。
操作はコマンドメニューで引数表示をマウスクリックし, 次にノードボックスあるいはターミナルボックスをクリックする。SAX トップウィンドウ起動後, 最初の引数表示であればインスペクタウィンドウの位置決めを要求してくるのでマウスで位置決めをおこなう。インスペクタの操作方法については CIL・SIMPOS の操作説明書参照のこと。

部分木表示

ツリーブラウザを用いて部分木構造の表示を行なう。
コマンドメニューで部分木表示をクリックし, 次にノードボックスをクリックする。するとそのノードをルートノードとする部分木構造をツリーブラウザに表示する。

全体木

全体の構文木構造をツリーブラウザで表示する。操作方法については 2.5 ツリーブラウザを参照のこと。

ストリーム

ストリームインスペクタでストリームのインスペクトを行なう。
ストリームインスペクタは, 文法規則選択メニューと文法規則表示ウィンドウから構成される。

コマンドメニューでストリームをクリックし, 次にターミナルボックス間の OX 印の領域をクリックする。するとその部分を流れるストリームの一番浅い部分の識別子の選択メニュー (ルール選択メニュー) があらわれる。識別子はその識別子の位置に * を埋め込んだ元の文法規則の形で表示される。文法規則の表示は引数を省略した簡素な形で表示される。 begin,end,right 識別子はそのまま表示する。項目の表示は, 通番: 文法規則番号: 文法規則の形式で行う。ここでの通番はメニュー中での表示順を表す。

このメニューで文法規則項目を左クリックすると選択となり, 文法規則表示ウィンドウにその文法規則を表示し, 選択

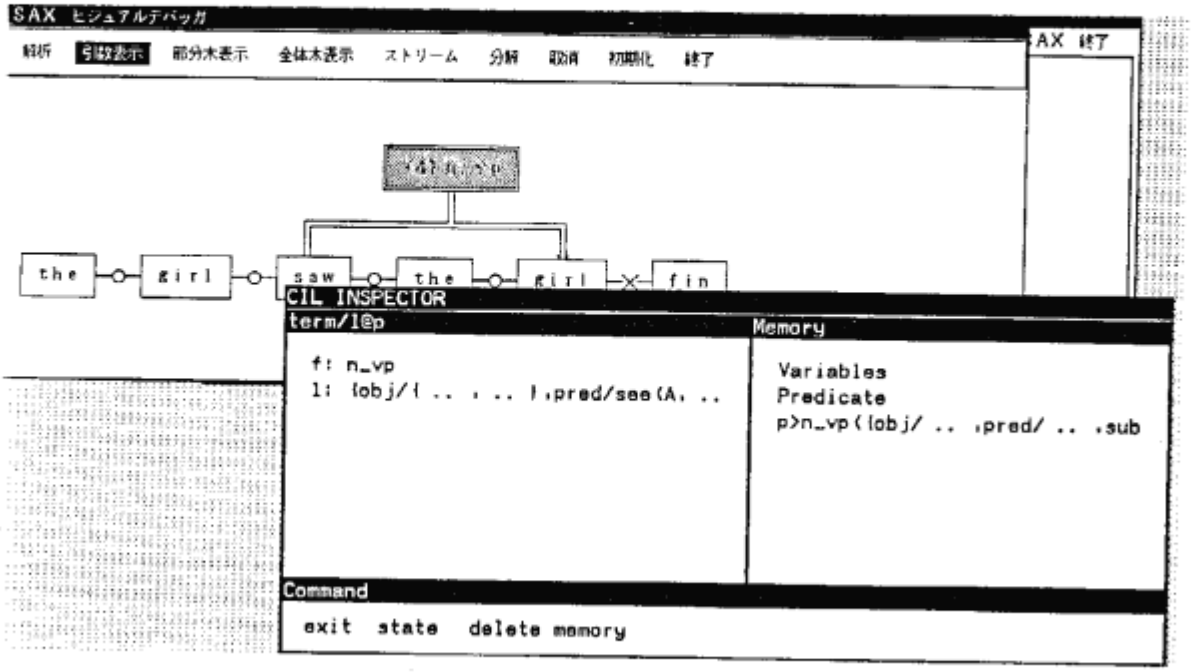


図 2.51: ビジュアルデバッガでの CIL インスペクタ

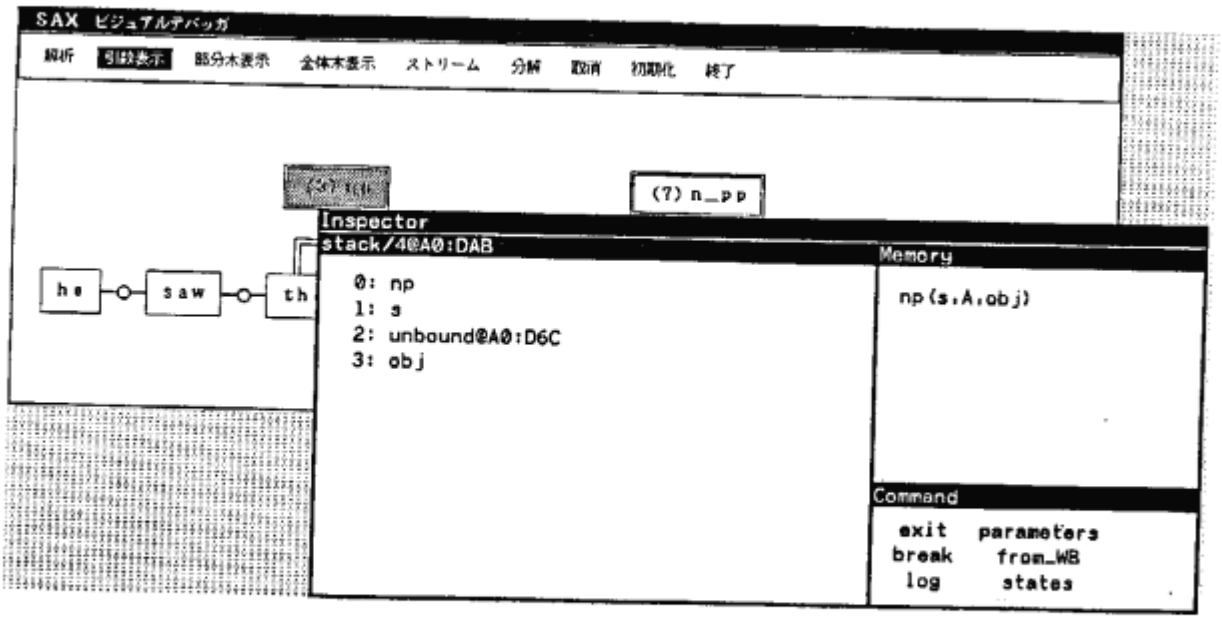


図 2.52: ビジュアルデバッガでのインスペクタ

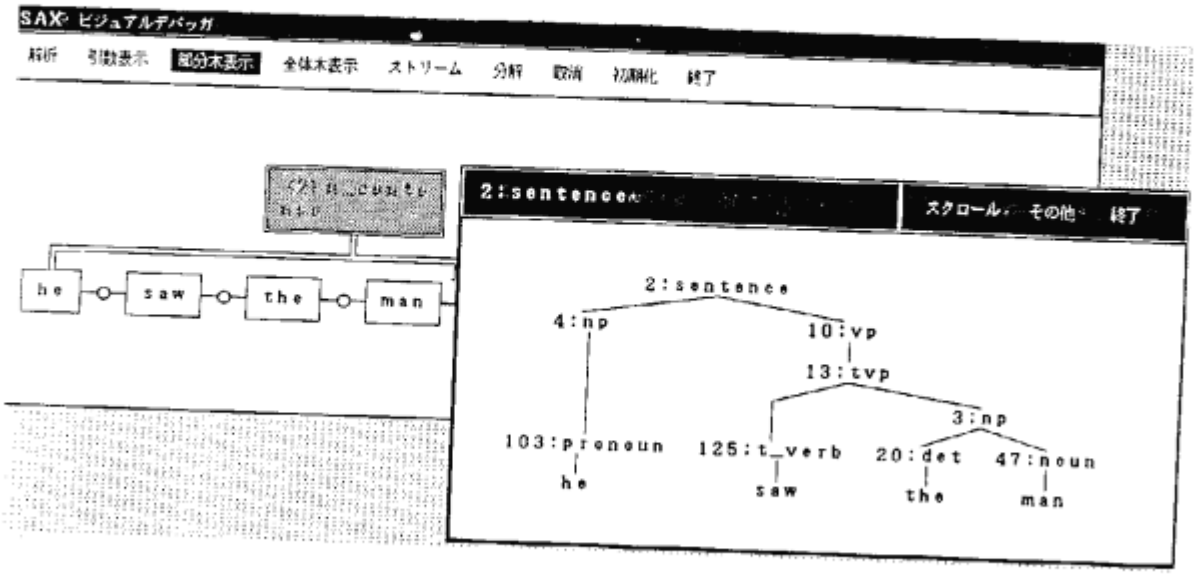


図 2.53: 部分木表示

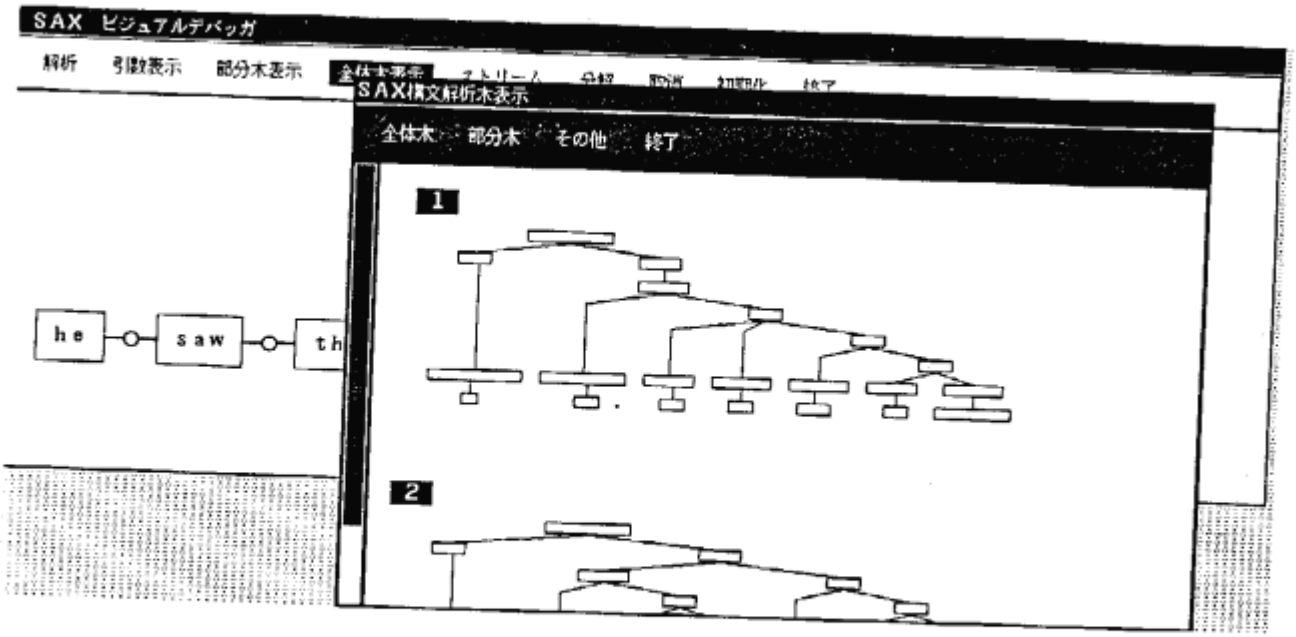


図 2.54: ビジュアルデバッガでのツリーブラウザ

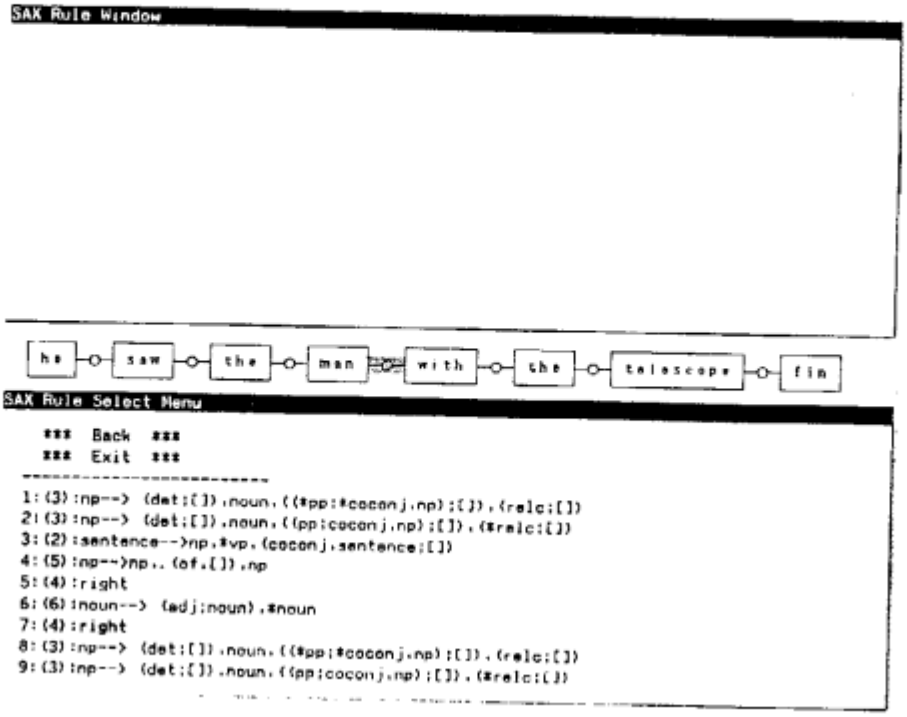


図 2.55: ルール選択メニュー

された文法規則に対応する識別子の下位ストリームを元に選択メニューを再表示する。ストリームが空になるまでこの処理を繰り返す。

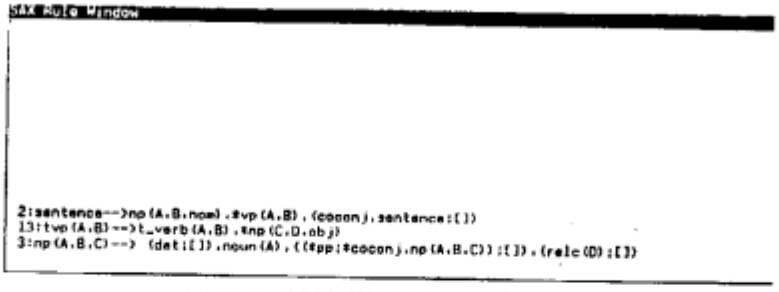


図 2.56: 文法規則表示ウィンドウ

選択メニューで Back を選択すると一つ上のストリームに戻り, Exit を選択するとストリームインスペクタの終了となる。
項目表示は簡素化して行っているが, 右クリックすることにより詳細表示ウィンドウにその項目の詳細表示をおこなう。マウスをウィンドウから外すかキー入力によりこのウィンドウは消える。

分解

ノードボックスをそのノードが構成される一つ前の状態に戻す。
コマンドメニューで分解をクリックした後, ノードボックスをクリックする。

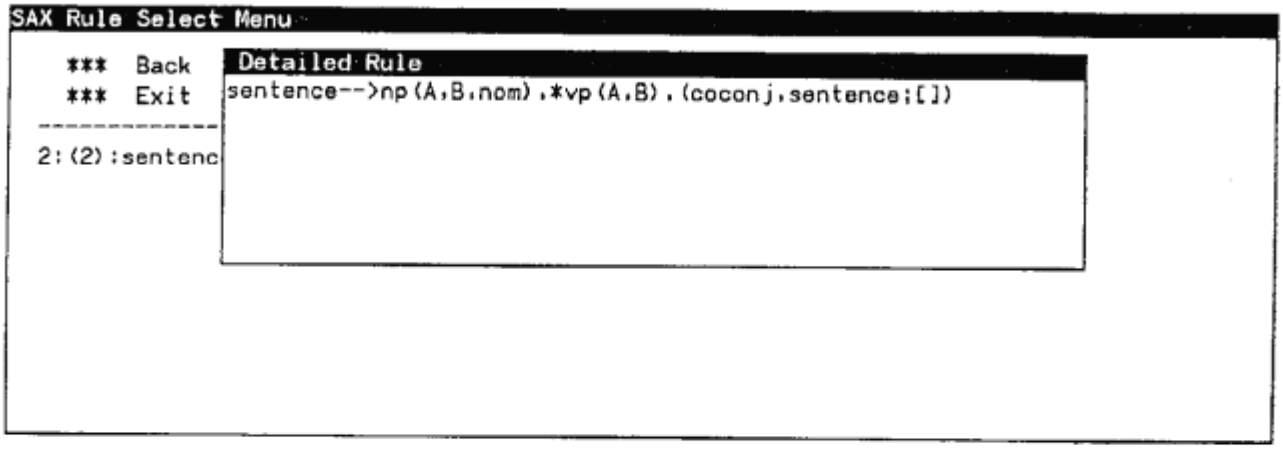


図 2.57: 詳細表示例

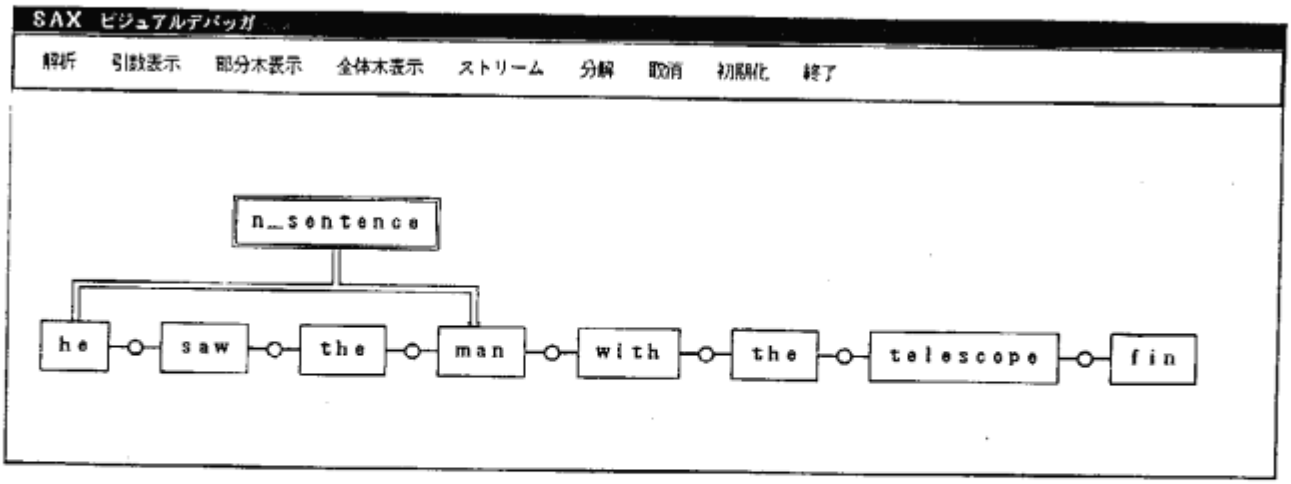


図 2.58: 分解前

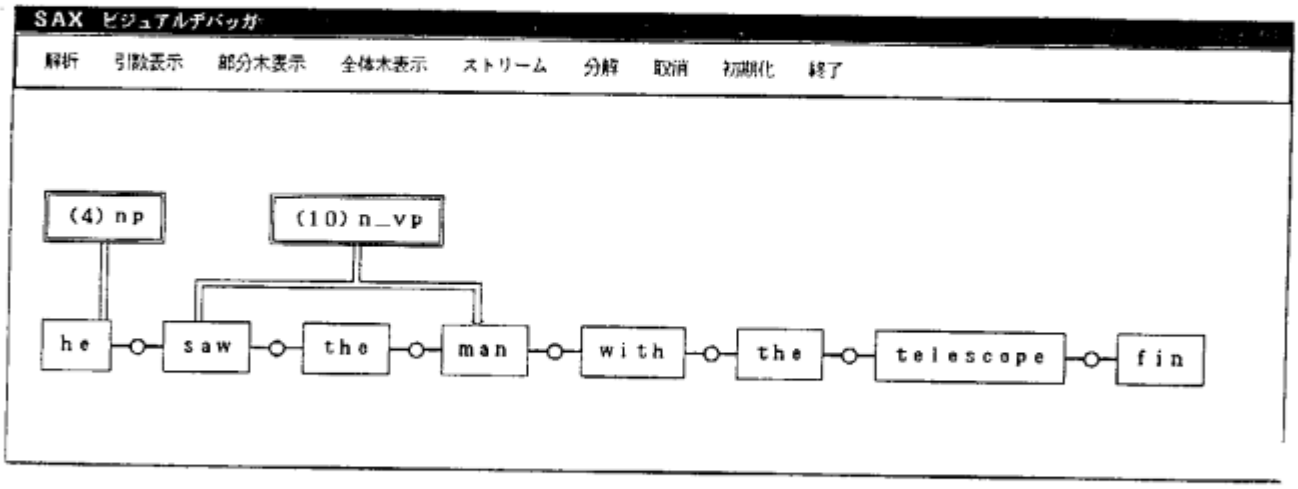


図 2.59: 分解後

取消

直前のコマンドを取り消す。
コマンドメニューで取消をクリックすると前回実行のコマンドの処理を取り消す。

初期化

ウィンドウ上のノードボックスを消去し, デバッガ起動直後の状態にする。
コマンドメニューで初期化をクリックする。

2.7 コマンド一覧

2.8 出力メッセージ

エラーメッセージ一覧

トップウィンドウ —

「ファイル File がみつかりません」

・文法ファイル登録時に, 指定したファイルがディレクトリ "me:DCG>¡" の下にない.

「クラス Class のテンプレートがありません. ファイル File をトランスレートしてください」

・その他のロードで指定した文法ファイルがロードできない. まずトランスレートを行なう.

「ファイル File の ESP ソースが無いのでカタログできません. トランスレートしてください」

・その他のカタログで指定した文法ファイルが ESP ソースがないのでカタログ出来ない. まずトランスレートを行なう.

「ファイル File の ESP ソースがカタログされてないか, またはトランスレートされてません」

・その他のセーブで指定した文法ファイルがセーブできない. まずカタログするかトランスレートする.

トップメニュー項目	サブメニュー項目	コマンド書式
文法	登録	add 「ファイル名]
	削除	delete [ファイル名]
変換		translate [ファイル名] , tran [ファイル名]
実行		execute [ファイル名] , exec
ライブラリ	カタログ	catalogue [ファイル名]
	ロード	load [ファイル名]
	セーブ	save [ファイル名]
クローズ	close	
その他	別名設定	define [別名, コマンド名]
	別名一覧	show_alias
	別名削除	undefine [別名]
	環境ファイル更新	sync
	強制終了	abort
終了		exit

図 2.60: トップウィンドウコマンド書式

トップメニュー項目	サブメニュー項目	コマンド書式
文解析	キーボード	keyboard
		lax
構文木表示	グラフィック表示	g_tree
	テキスト表示	t_tree
意味表示	ファイル出力	sem_out
	ウインドウ	sem_pp_w
	ファイル	sem_pp_f
デバッグ		debug
トップウインドウ		return
クローズ		close
終了	終了	exit
	強制終了	abort

図 2.61: 実行ウィンドウコマンド書式

「別名の登録に失敗しました．」
・別名設定時に，コマンド名を間違えたため登録に失敗した．

「DCG ファイルが見つかりません」
・トランスレート時に文法ファイルがみつからない．ディレクトリ "me:DCG¿" に文法ファイルをコピーする．

「N 個の非終端記号が見つかりました」
・トランスレート中に，文法ファイル中の左辺にあらわれて右辺にあらわれないカテゴリが N 個みつかった．このままで構文解析を行うと解析に失敗する可能性があるので文法をチェックする．

実行ウィンドウ —
「終端記号 Category が未定義です．」
・文解析のキーボード入力時に入力した文の中に未定義のカテゴリがあらわれた．入力し直すか文法ファイルを修正する．

「LAX データエラー」
・LAX からの入力構造が規定にあっていないので解析を中止する．データ構造を確認する．

「解析に失敗しました．Category が未定義です．」
・構文解析時に未定義の終端記号または非終端記号があらわれた．入力し直すか文法ファイルを修正する．

「解析できません．」
・構文解析が途中で失敗した．システムエラー．

「優先得点が整数ではない.(id = NN).」
・構文解析実行中の優先度計算時にアンバウンドの変数があらわれた．NN にその位置を示すので優先記述項のチェックを行なう．NN は整数または id_'N1'_'N2' の形をしたアトム.NN,N1 が文法規則番号をあらわす．エラーメッセージ表示後，キー入力待ちとなる．改行キーを押すとこのエラーが発生するのは主に予約語のスペルミスによる．

第 3 章

ユーザインターフェース

SAX では, トランスレータ, 解析実行支援, 構文木表示 (ツリーブラウザ) の各機能を, ユーザプログラムから利用できるよう公開している。
ここでは各機能の利用方法について説明する。

3.1 トランスレータ

3.1.1 概要

トランスレータでは,SAX 文法を記述したファイルから,ESP や Prolog での直接構文解析をおこなうソースコードを生成する。 ESP のコード生成を行なう場合は補強項での CIL の記述や, 優先得点の記述を行うことができ, トランスレータで, 生成されたパーサクラスのライブラリアンへのカタログ, セーブを行える。

3.1.2 必要なクラス

トランスレータを利用するには以下のクラスが必要である。すでに SAX がライブラリアンにセーブされているときは,sax_translater をロードすれば他の必要なクラスは自動的にロードされる。

ファイル名	クラス名
tran.esp	sax_translater
	pass0
	pass1
	pass2
	esp_code
	sax_extra_condition
	sax_preference_condition
	prolog_code
	sax_esp_file
	sax_prolog_file
	as_sax_file_manager
	sax_link
	sax_table
	sax_translater_option_menu
util.esp sax_io_buffer	
	sax_input_file
	sax_output_file
	sax_symbolizer
	sax_operator
ltb.esp	cil_syntax_sugar

その他, 補強項に CIL 記述がある場合は,CIL 3.6 版 , LTB 1.1 版 の各クラスが必要となる。

3.1.3 メソッド

クラス sax_translater のトランスレータ用クラスメソッドを公開する

1. :translate(Class,F_Path,F_Name,C_Name,O_List, ～C_List)

   (a) 機能
   トランスレート用述語。文法ファイルのパスを F_Path に、文法ファイル名を F_Name 、に, 生成されるパーサクラス名を C_Name で指定する O_List にはトランスレート時のオプションを指定する。
   C_List には生成されたパーサクラスのコード定義を出力したバッファのリストとファイル名のリストを要素とする 2 重のリストが返る。

   (b) 引数

   - Class
     クラスオブジェクト #sax_v10##sax_translater
   - F_Path
     ファイルパス名。ストリング。
   - F_Name
     ファイル名。ストリング。
   - C_Name
     クラス名。アトム。
   - O_List
     オプションリスト。
   - C_List
     変換言語が ESP なら出力バッファとトランスレートにより生成されるファイル名のリスト。
     Prolog ならファイル名。トランスレートに失敗した場合はその原因を以下のアトムで返す。

dcg_file_not_found	文法ファイルが見つからない。
menu_abort	オプションメニューで中止を選択した。
pass1_failed	パス 1 で失敗した。
pass2_failed	パス 2 で失敗した。

2. :translate_with_window(Class,F_Path,F_Name,C_Name,O_List, ～C_List)

   (a) 機能
   入出力ウィンドウつきのトランスレート用クラス述語。最初にウィンドウのポジショニングを求めてくるので, マウスで位置、サイズを決定する。入出力ウィンドウがある他は :translate/6 と同じ。
   入出力ウィンドウにはトランスレータの実行状況を表示する。
   トランスレート終了後キーボードからの入力まちとなるリターンキーを押下することによりウィンドウが消える。

   (b) 引数

   - Class
     クラスオブジェクト #sax_v10##sax_translater
   - F_Path
     ファイルパス名。ストリング。
   - F_Name
     ファイル名。ストリング。
   - C_Name
     クラス名。アトム。

- O_List,
  オプションリスト。
- C_List
  変換言語がESPならトランスレートにより生成されるクラス名のリスト。

3.1.4 オプションリストの指定方法

オプションリストは,項目名と値のベクタを要素とするPrologリストである。
オプションリストで与えた値の方が,文法ファイル中でのトランスレータ宣言値よりも強い{menu,off}がある場合は,
パス0後のオプション設定メニューはあらわれない。
項目名と設定可能な値を以下に示す。

- language
  変換先言語名

esp	ESPへの変換
prolog	Prologへの変換

- esp_source
  ESPソースファイル生成の有無

on	生成する
off	生成しない

- debug
  デバッグモード,ノーデバッグモードの別

on	デバッグモード
off	ノーデバッグモード

- tree
  構文木情報の有無

on	有
off	無

- preference
  優先解記述の有無

on	有
off	無

- cil
  補強項,遅延補強項のCIL記述の有無

on	有
off	無

- cil_ec_compile
  CIL コンパイルドクラス生成の有無

  on　生成する
  off　生成しない

- cil_ec_interpret
  CIL インタープリタファイル生成の有無

  on　生成する
  off　生成しない

- translate_mode
  トランスレータのモード

  translate　トランスレートのみ行なう
  catalogue　トランスレート, カタログを行なう
  save　トランスレート, カタログ, セーブを行なう

- menu
  パス 0 終了後のトランスレータオプションメニューの有無

  on　有
  off　無

変換先が prolog のときは,language 以外のオプションは意味を持たない。
CIL 記述が無い場合 ({cil,off}) は,cil_ec_compile,cil_ec_interpret は意味を持たない CIL 記述があり ({cil,on}),cil_ec_compile,

cil_ec_interpret が共に off の場合, デバッグモードの場合 cil_ec_interpret を on にし, ノ - デバッグモードの場合 cil_ec_compile を on にする。
translate_mode が translate の場合,esp_source を on にする。

3.2 実行支援クラス

3.2.1 概要

トランスレートにより生成されたコードだけでは, 構文解析を行うことはできない。
ここでは実行支援クラス sax_starter の標準的な構文解析の起動用メソッドを説明する。

3.2.2 必要なクラス

パーサクラスで構文解析を行うには, パーサクラスの他に起動クラスとして sax_starter が必要である。補強項, 遅延補強項中に CIL 記述がある場合は cil_interpreter , CIL 3.6 版 , LTB 1.0 版 が必要となる。

3.2.3 メソッド説明

1. `:parse(Class,Exe,Input,Info, ～Out, ～Time)`

(a) 機能
構文解析用述語。Exe にトランスレートにより生成された解析実行クラスのオブジェクト (クラス, インスタンスどちらでもよい),Input に解析実行する文をアトムのリストで与える。Info には CIL の変数情報を与える。文法規則中に CIL を含まない場合はアンバウンドのままでよい。構文解析結果を Out に, 実行時間をミリセカンド単位で Time にかえす。
遅延実行項の評価は, トップダウン, 左から右に行う。

(b) 引数

- Exe
  実行クラスのインスタンスオブジェクト。
- Input
  構文解析する文。アトムのリスト。
- Info
  CIL 評価時の変数情報。
- Out
  構文解析の結果。リストリスト中の各要素が 1 つの解析結果を表す一つの解析結果は 4 要素のスタックベクタで以下の構造をしている。
  {構文木情報, 引数情報, 遅延補強項, 優先得点}
  構文木情報 = n 分木構造のリスト
  引数情報 = トップノードの引数のリスト
  遅延補強項 = 構文木情報と同じ n 分木構造のリスト
  優先得点 = [トップノードの得点 —begin の得点]
- Time
  解析実行時間。構文解析に要した時間のみで, 前処理, 後処理の所要時間は含まない。
  ミリセカンド単位。整数。

2. `:parse(Class,LC_Mode,Exe,Input,Info,~Out,^Time)`

(a) 機能
遅延補強項の評価方法を指定して構文解析を行なう。他は :parse/6 と同じ。

(b) 引数

- LC_Mode
  遅延実行項の評価方法。アトム。
  tdlr トップダウン, 左から右。 tdrl トップダウン, 右から左。
  bulr ボトムアップ, 左から右。 burl ボトムアップ, 右から左。
- Exe
  実行クラスのインスタンスオブジェクト。
- Input
  構文解析する文。アトムのリスト。
- Info
  CIL 評価時の変数情報。
- Out
  構文解析の結果。
- Time
  解析実行時間。

3. `:parse_no_eval(Class,Exe,Input,Info,~Out,~Time)`

(a) 機能
遅延実行項を評価しないほかは :parse/6 と同じ。このときの遅延補強項の評価はユーザにまかせる。

(b) 引数

- Exe
  実行クラスのインスタンスオブジェクト。
- Input
  構文解析する文。アトムのリスト。
- Info
  CIL 評価時の変数情報。
- Out
  構文解析の結果。
- Time
  解析実行時間。

3.2.4 注意事項

CIL 記述がある場合は CIL インタープリタまたは CIL コンパイルドオブジェクトが必要となる。
トランスレータ時のパラメタ設定で cil_interpret が on であれば CIL インタープリタが,cil_compile が on であれば CIL コンパイルドインスタンスが必要となる両方とも on であればどちらでも構わない。
CIL インタープリタを使用する場合は解析メソッドを呼ぶ前に以下の前処理が必要となる。

- パーサクラスのインスタンスオブジェクトを生成する。
  :create(#'パーサクラス名',Parser)。
- CIL インタープリタを生成する。
  :create(#ltb_v10##cil_interpreter,InterPreter,Info)。
- CIL ファイル名を取り出す。
  :cil_file_name(Parser,FileName)。
- CIL ファイルをコンサルトする。
  :consult_on(InterPreter,FileName)。
- パーサインスタンスに CIL インタープリタをセットする。
  :set_cil_interpreter(Parser,InterPreter)。
  :set_cil_executor(Parser,InterPreter)。

CIL コンパイルドインスタンスを使用する場合は解析メソッドを呼ぶ前に以下の前処理が必要となる。

- パーサクラスのインスタンスオブジェクトを生成する。
  :create(#'パーサクラス名',Parser)。
- コンパイルドインスタンスのクラス名を取得する。
  :cil_class_name(Parser,ClassName)。
- コンパイルドインスタンスを生成する。
  :create(ClassName,CilInstance)。
- パーサインスタンスに CIL コンパイルドインスタンスをセットする。
  :set_cil_instance(Parser,CilInstance)。
  :set_cil_executor(Parser,CilInstance)。

3.3 ツリーブラウザ

3.3.1 概要

SAX ツリーブラウザーは，木構造データを SIMPOS ウインドウグラフィック表示し，ブラウジングするためのツールである。ここでいう木構造とは，根ノードが1本で枝が環状でないことが前提である。
ユーザは，出力時のノードのサイズや枝の線幅，線種，ウインドウサイズ等の様式をインスタンス生成地に設定でき，またウインドウサイズを越える木の場合でも，スクロールウインドウで2次元スクロールを行うことで，表示部分を自由に変更できる。
使用方法は，プログラムまたはデバッガ上より，

①クラス sax_tree_browser のインスタンス・オブジェクト生成
②ブラウジングメソッドの呼び出し
③ウインドウ操作
:
②③繰り返し
:
④資源の解放

を行う。

3.3.2 メソッド使用方法

1. インスタンス・オブジェクトの生成

```
:create(#sax_tree_browser, Initiations, ^Obj) \index{create}
    Initations  初期化項目リスト
                [項目_0, 項目_1,]
```

- 初期化項目
  初期化項目はウインドウサイズ，位置，出力様式の初期値を設定する。省略時はデフォルト値で初期化される（アンダーライン）。
- メニューウインドウ初期化項目

(a) title(Title)

Title : タイトル文字列 "SAX TREE BROWSER"

(b) size(Width,Height)
ウインドウサイズ

Width : 1以上の整数またはmanipulator
Height : 1以上の整数またはmanipulator

(c) position(X,Y)
ウインドウポジション

X : 1以上の整数またはmanipulator
Y : 1以上の整数またはmanipulator

(d) menu_scale(Scale)
グラフィックメニューのスケール

Scale : スケールの母数。木構造をタテ，ヨコを 1/Scale に縮小。
1以上の整数。_2

- ブラウジングウインドウ初期化項目

(e) tree_window(size(Width,Height),position(X,Y)) ブラウジングウインドウのサイズと位置
size(Width,Height)
ウインドウサイズ

Width : 1以上の整数またはmanipulator
Height : 1以上の整数またはmanipulator

position(X,Y)
ウインドウポジション

X : 1以上の整数またはmanipulator
Y : 1以上の整数またはmanipulater

(f) scroll_scale(Scale)
スクロールウインドウのスケール
Scale : スケールの母数。 木構造をタテ, ヨコを 1/Scale に縮小。
1以上の整数。 :3

- 出力様式

(g) font(Font_Name)
出力フォント名
Font_Name: test_11,font_13,kanji_16

(h) line_width(Width)
ノードとノードを結ぶアークの線幅。

Width : 1,2,3,4

(i) line_type(Type)
ノードとノードを結ぶアークの線種。

Type :	solid	実線
	broken	破線
	dotted	点線
	alternate	一点鎖線
	up	上方向矢印線
	down	下方向矢印線

(j) max_mode_size(Width,Height)
ノード名を表示する領域の最大サイズ（文字単位）

Width : 幅 20
Height : 高さ 1

図 3.2 参照。

(k) node_margin(Left,Bottom)
ノード出力領域のマージン

Left : 左側マージン 1
Bottom : 底側マージン 1

(l) standard_bottom(Level)
座標の位置合わせを行うノードのレベル

Level : 0 位置合わせなし
1 葉ノードを位置合わせする
2 葉ノードとその親ノードを位置合わせする
:
:

【 例 】"新世代コンピューター技術開発機構"（15 文字）の場合

Width = 5 Height = 1

```
      5
   <------>
1  新世代 . .                    【後ろ省略】
```

Width = 5 Height = 2

```
      5
   <------>
2  新世代コン
   . . 発機構                    【途中省略】
```

Width =10 Height = 2

```
        10
   <---------->
2  新世代コンピューター技
   術開発機構                    【幅 5 文字分カット】
```

Width =20 Height = 2

```
             15
   <--------------->
1  新世代コンピューター技術開発機構
```

図 3.1: max_mode_size

2. ブラウジングメソッド

(a) :brouse/2

同時に複数の木をブラウジングする時に使用する。
木構造を簡略縮小してグラフィックメニューウインドウへ表示。
メニューコマンドにて全体木，部分木をブラウジングウインドウへ表示。

```
:brouse(Obj,TreeL) \index{brouse}
  TreeL = [木構造データ _1, 木構造データ _2,...., 木構造データ _n]
  木構造データ = [title(Title), データリスト]
                        または
                    データリスト
  title(Title) = ストリング または アトム
  データリスト = [ノード名, アーク情報]
  ノード名      = アトム
  アーク情報    = (データリスト _1, データリスト _2,.... データリスト _n)
```

【データリストの例】

```
[sentence,
   [vp,
      ([t_verb,
         [time]],
       [np,
          ([np,
              [noun,
                 [files]]],
           [pp,
              ([prep,
                 [like]],
           [np,
             ([det,
                 [an]],
              [noun,
```

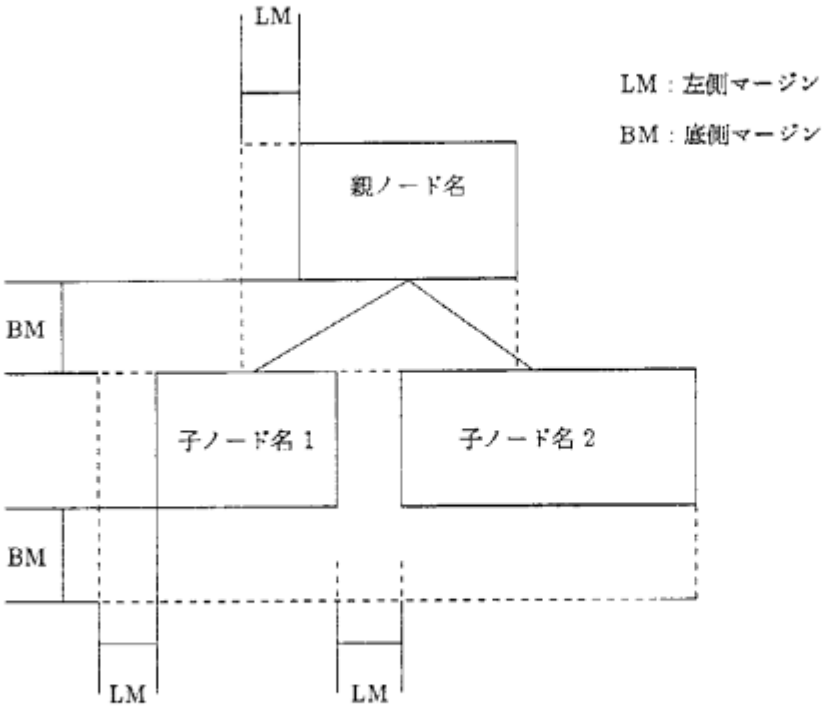


図 3.2: max_node_size ノード出力領域マージン

```
[arrow]]])])])])]]]
```

(b) :browse_without_menu/2
メニューウインドウを介さず直接ブラウズウインドウへ木を表示する。

```
:browse_without_menu(Obj, Tree_dataL)
    Tree_dataL = 木構造データリスト
```

3.3.3 操作方法

1. メニューウインドウ

(a) ウインドウ構成
メニューウインドウはインスタンス・メソッド:brouse/2 を呼び出すとスクリーンに表示され，終了コマンドでスクリーンへの表示をやめる。ウインドウはコマンドの選択を行うコマンドメニューウインドウ。木の選択を行う木メニューウインドウ，木メニューを縦スクロールするためのスクロールウインドウから構成されている。

(b) コマンド
コマンドを選択すると，コマンド名が反転し，そのコマンドモードに入る。コマンドモードは，そのコマンドを実行するか，もう一度コマンドの選択を行う事で解除する。以下，各コマンドについて説明する。
（カッコ内は，出力フォントが test_11,font_13 のときのコマンド名)

i. 全体木 (tree)
木メニューウインドウで木の選択を行う。ブラウジングウインドウが未生成ならウインドウの生成を行い，選択した木をブラウジングウインドウへグラフィック表示する。もし，選択した木がすでに表示中ならば警笛を鳴らし，そのウインドウをスクリーンの全面に表示する。

ii. 部分木 (part_tree)
選択したノードを根ノードに部分木をブラウジングウインドウへグラフィック表示する。
ノードの選択はマウスマーカを表すボックスの上に移動し，クリック入力する。ボックス外からの入力は部分木コマンドの取り消しとなる。

iii. その他
下図のメニューウインドウよりその他のコマンドを実行する。

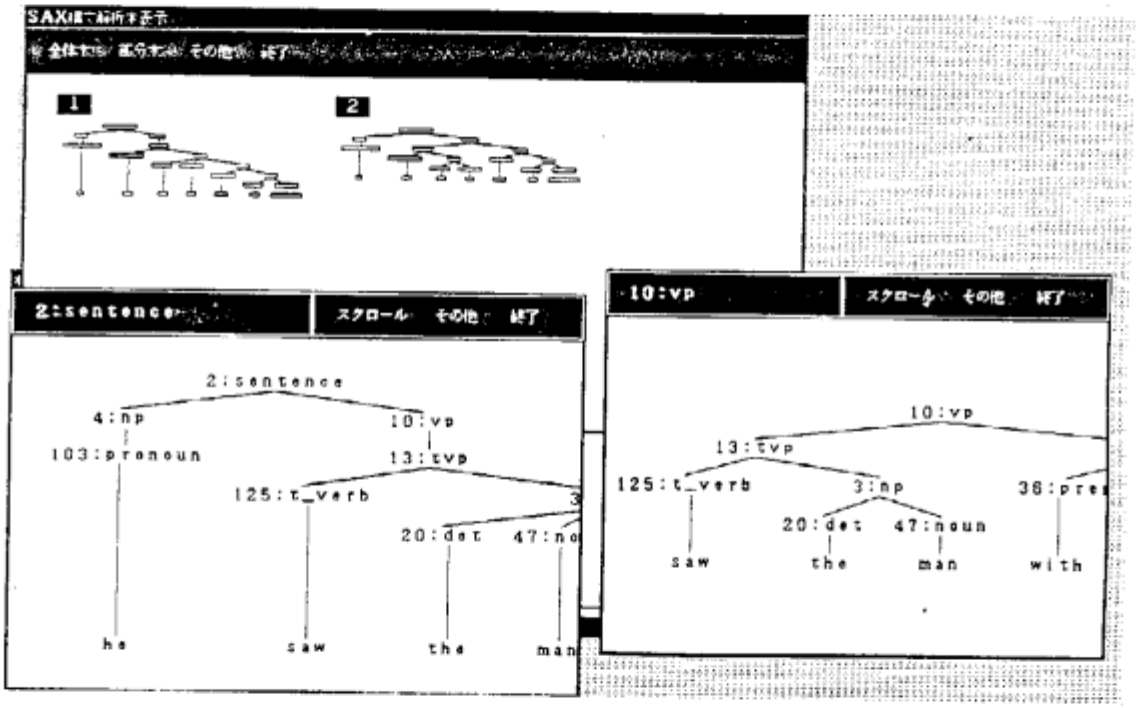


図 3.3: 全体木・部分木

- reshape
  メニューウインドウのサイズと位置をマウスで変更する。
- close
  全てのブラウジングウインドウの表示をとりやめ，メニューウインドウをクローズする。
- hide
  メニューウインドウとすべてのブラウジングウインドウをスクリーンの最下層に置く。
- move
  メニューウインドウの位置をマウスで移動する。
- redirection
  ウインドウ出力様式を変更する。

出力様式変更ウインドウ

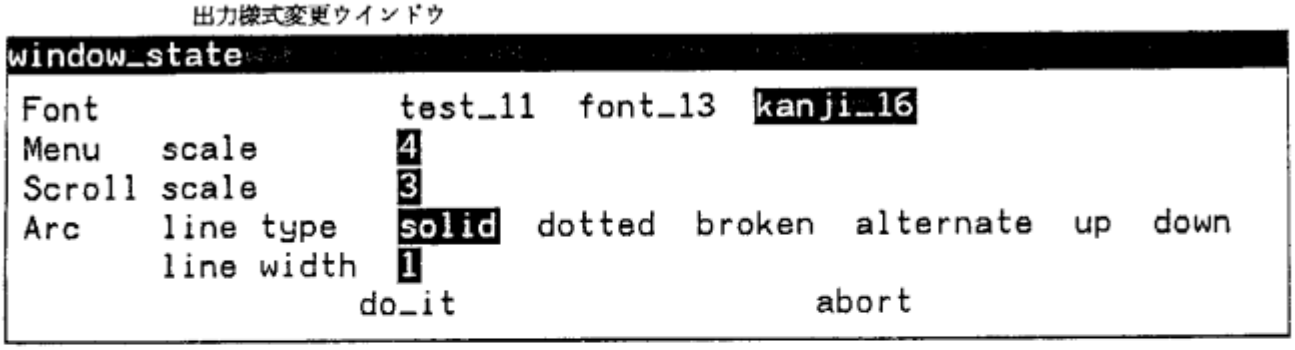


図 3.4: 出力様式変更ウインドウ

(1) **font** メニューウインドウ，ブラウジングウインドウの出力フォントを変更する。

(2) **Menu Scale** グラフィックメニューのスケール (1/Scale) を変更する。

(3) **Scroll Scale** ブラウジングウインドウのスクロールウインドウの縮小率 (1/Scale) を変更する。

(4) **Arc line type** ブラウジングウインドウのノードとノードを結ぶアークの線種を変更する。

(5) **Arc line width** ブラウジングウインドウのノードとノードを結ぶアークの線幅を変更する。

iv. attribute

木構造およびノードの出力定義情報を変更する。

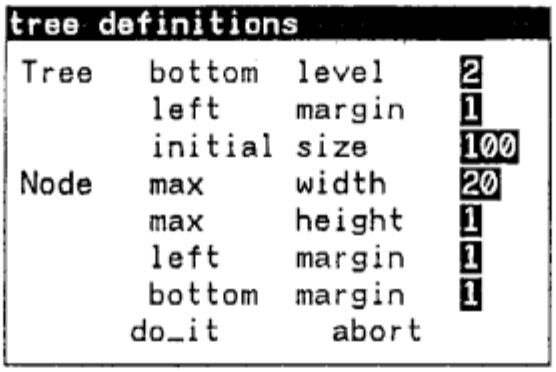


図 3.5: 出力定義情報変更メニュー

(1) **bottom level** 位置合わせを行うノードのレベル

(2) **left margin** 木全体の左側マージン

(3) **initial size** ノード情報を格納するプールの初期サイズ。

(4) **max width** ノード名出力最大幅（文字単位）。

(5) **max height** ノード名出力最大高（文字単位）。

(6) **left margin** ノード出力領域の左側マージン（文字単位）。

(7) **bottom margin** ノード出力領域の下側マージン（文字単位）。

v. 終了 (Exit)

メニューウインドウおよびすべてのブラウジングウインドウの表示をやめ，制御を呼び出し側に返す。

(c) ブラウジングウインドウ

i. ウインドウ構成

メニューウインドウの"全体木"または"部分木"コマンド，インスタンスメソッド:browse_without_menu/2 によりスクリーンへ表示される。

- タイトルウインドウ

  タイトルを表示する。タイトルが未定義のときは根ノード名を表示する。

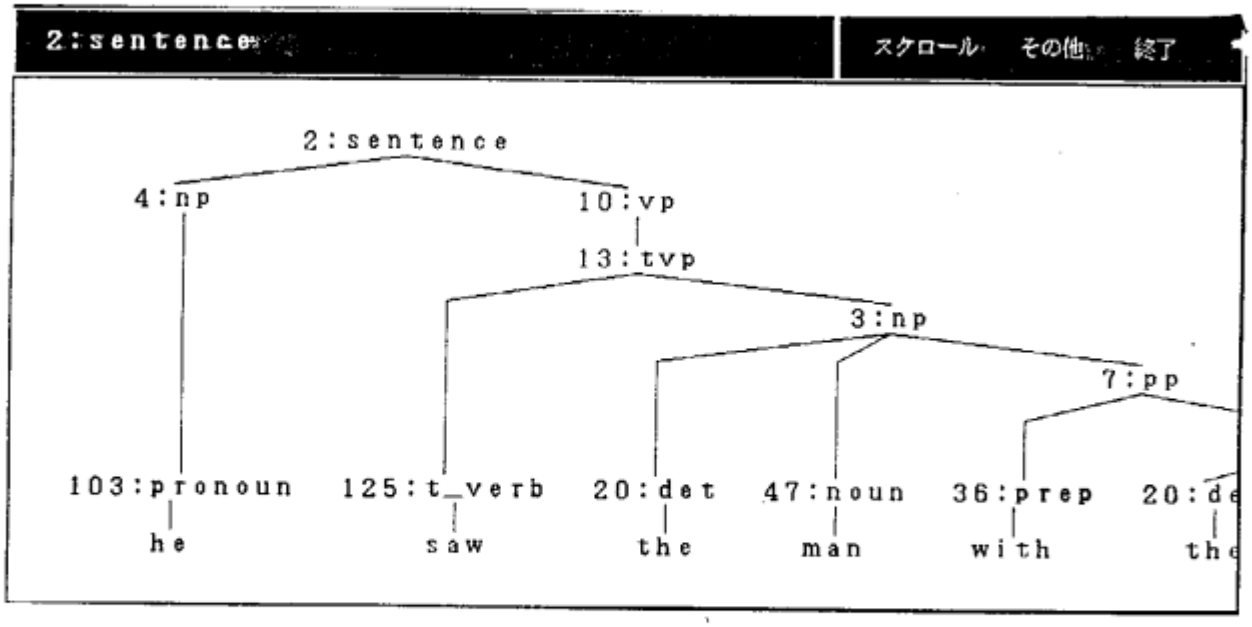


図 3.6: ウインドウ構成

- コマンドメニュー
  コマンドの選択と実行を行う。
- ビューポート
  木構造を実寸大でグラフィック表示する。

ii. コマンド

- スクロール (Scroll)
  スクロールウインドウを表示する。すでに表示済みなら，スクリーンの全面に表示する。
  - スクロール方法
    ビューポートに表示されている領域を反転表示し，マウスマーカがスクロールウインドウ上にあれば，矩形のジャンプスクロールマーカを表示する。クリック入力でジャンプスクロールマーカで囲まれた領域をビューポートへ表示する。
- その他 (others)
  下図のメニューよりウインドウ状態を変更する。
  - reshape
    ウインドウのサイズと位置をマウスで変更する。
  - close
    ウインドウをクローズする。
  - hide
    ウインドウをスクリーンの最下層に置く。
  - move
    ウインドウの位置をマウスで変更する。
- 終了 (Exit)
  ウインドウの表示をやめ，制御を呼び出し側へ返す。

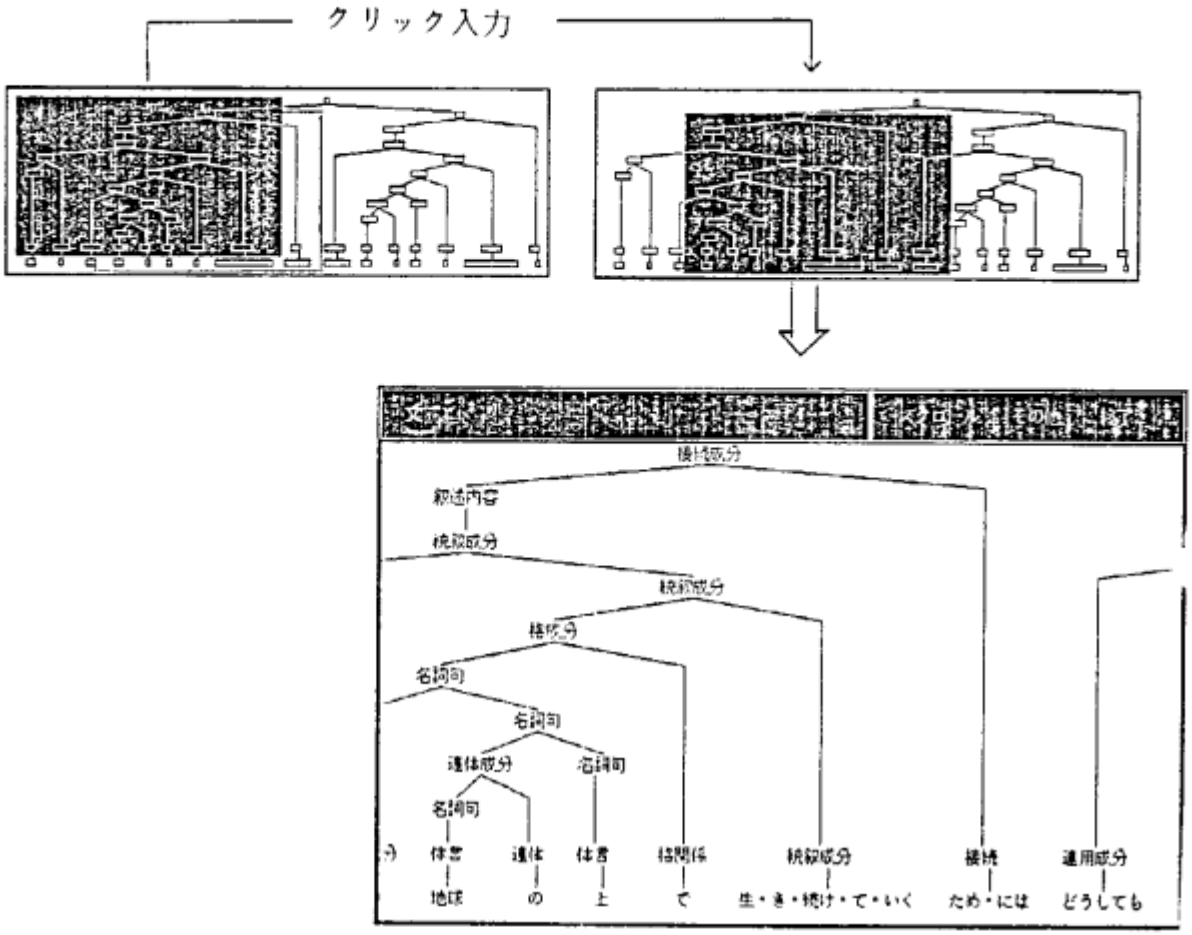


図 3.7: スクロール

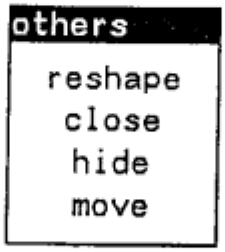


図 3.8: その他のメニュー

第 4 章

文法作成

本章では、SAX システムの文法記述言語と文法作成上の注意について述べる。

4.1 言語仕様

SAX の文法記述言語は、DCG を基本にしているが、自然言語の解析を効率良く実行するために拡張がなされている。また文法記述上、解析アルゴリズムに由来する制約がある。

4.1.1 解析方式と文法作成上の注意

期待通りに動作する解析を行なう文法を書くにはある程度、SAX での解析の進み方を理解する必要がある。ここでは、文法を作成する上で必要最小な範囲で解析の進み方を説明する。詳細については、文献[6]等を参照されたい。文法は図4.1のような DCG の形式で記述される。

```
sentence(Sentence) --> np(Np),vp(Vp),{sentence_construct(Np,Vp,Sentence)}.   (1)
np(Np) --> det(Det),noun(Noun),{np_construct(Det,Noun,Np)}.                  (2)
vp(Vp) --> vt(Vt),np(Np),{vp_construct(Vt,Np)}.                              (3)
noun(Boy) --> [boy],{dic(boy,Boy)}.                                          (4)
noun(Dog) --> [dog],{dic(dog,Dog)}.                                          (5)
vt(Hits) --> [hits],{dic(hits,Hits)}.                                        (6)
det(The) --> [the],{dic(the,The)}.                                           (7)
det(A) --> [a],{dic(a,A)}.                                                   (8)
```

図 4.1: DCG による文法記述の例

構文解析とは

構文解析の目的は、与えられた入力文と文法からあるカテゴリ [1] をトップに持つ解析木を求めることである。

上の文法に 'The boy hits a dog.' という入力文を与えた場合の解析木を図4.2に示す。

解析は、文法規則を入力文に適用して解析木を構築していくことによっていくことによって進行する。これを、次のような入力文に対して上記規則 (1) 〜 (7) がどのように適用されるかを示しながら説明する。

構文木は左隅から構成

解析木をどの位置から構築していくかという視点によって、構文解析手法を分類することが可能である。本システムでは左隅から規則を左から右に使いながら木を作っていく。文の先頭から読んでいってまず規則 (7) を使うと det をトップに持つ部分木が完成する。(a) のような木が出来る。ここで det が出来たのでこの det を右辺の先頭に持つ規則 (ここでは (2)) を適用することを試みる。

この状態は「不完全な部分木」が出来たと見ることができる。(b) この「不完全な部分木」を完成するためには、noun が必要である。2 番目の単語 dog に規則 (5) を適用することによって noun が得られる。

[1] (トップカテゴリ呼ばれる) 上の文法だと sentence

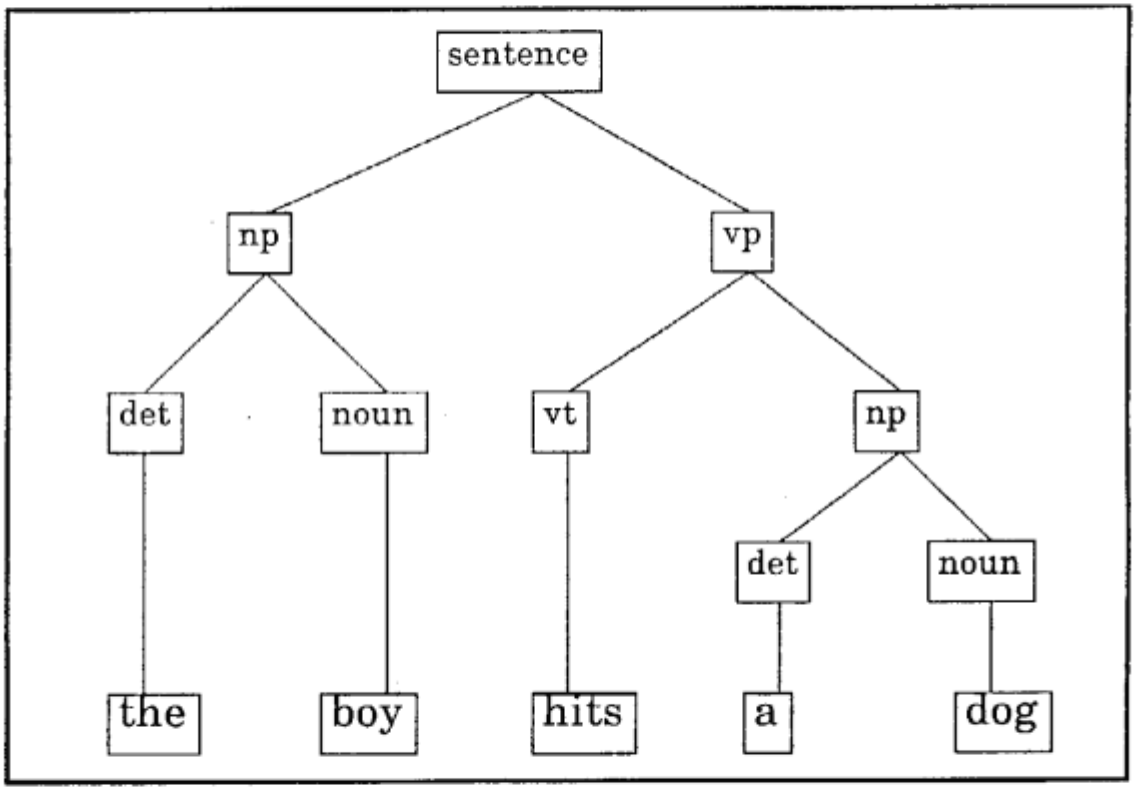


図 4.2: 解析木

(c) この「完成した部分木」を「不完全な部分木」に接続することによって規則 (2) の適用が完了し「部分木」が完成する。(c)

このような手続きを繰り返していき先頭の語から最後の語までで sentence が構成されれば解析に成功したことになる。この様子を図4.3に示す。

注意その 1:: 右辺の先頭には、補強項は書けない

このように SAX システムは上昇型の解析を行なうため右辺の先頭に補強項を記述することができない。

駄目な例 `np(Np) --> {check_np(Np)},det(Det),noun(Noun).`

複数の可能性を同時並行的に解析

一般にはある一つのカテゴリを左端に持つ規則は文法規則中に複数存在しうる。例えば、上の文法に次の規則を加えてみよう。

```
np(Np) --> det(Det),noun(Noun),wh_close(WhC).        (a.2)'
```

すると、det が得られた直後 (a) に (2) だけでなくこの規則も適用可能である。システムでは、複数の可能性を同時に試みる。この様子を図4.4に示す。

このように一つの部分木は複数の上位の木に共有される。

ということは、それらの部分木が持つ変数も共有されることを意味する。従って、図4.5のように変数 X を持つ tree が Tree1 と Tree2 に共有され Tree1 の規則中の中で変数 X が束縛されてしまった場合 Tree2 もこの束縛された X を持つことになる。つまり本来独立の解析が互いに影響しあい、文法作成者の意図してなかったような結果が得られることがあるのである。 これを防ぐために次のような注意が必要である。

注意その 2:: 右辺に含まれる変数には束縛を行なわないように規則を書く

これは、規則のなかで右辺から左辺へ、構文木のなかで下から上へのみ情報が流れるように文法を書くということである。

危ない例　`np(Np) --> det(Det),{Det= the},noun(Noun).`

注意その 3:: 補強項は再実行されない

SAX システムでは解析は決定的に実行され、バックトラックはおこらない。補強項についてもそうでありただ一度だけ評価される。次のような規則と手続き check_vt を考えて見よう。カテゴリ vt が完成するとこの規則の適用を試み check_vt の呼び出しが行なわれる。 ここで (b.1) が成功した場合、この後に np の構成を試みるがそれに失敗しても後戻りして (b.2) 以下を実行することはない。

```
vp(Vp) --> vt(Vt),{check_vt(Vt)},np(Np).
check_vt(Vt):- a1(Vt).        (b.1)
check_vt(Vt):- a2(Vt).        (b.2)
check_vt(Vt):- a3(Vt).        (b.3)
```

4.1.2 拡張

SAX システムでは、DCG に次の 2 つの機能－「遅延補強項」、「優先度計算」とそのための記法を導入した。また、CIL[2] とのインタフェースをサポートしており構文解析過程で、CIL の豊富な機能を用いて意味処理を実行させることができる。

[2] ユーザマニュアル CIL 編参照

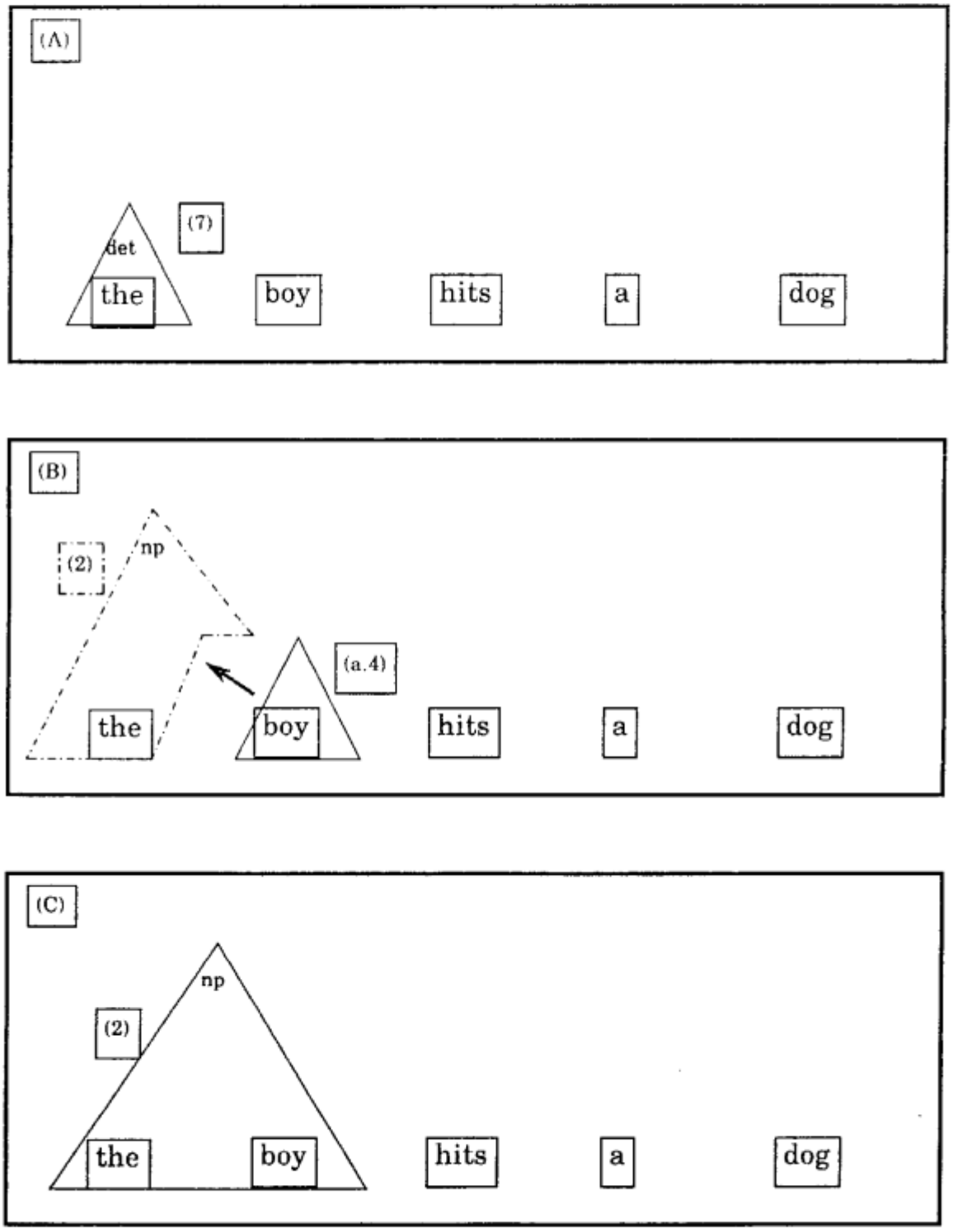

(A)
(7)
det
the
boy
hits
a
dog
(B)
(2)
np
(a.4)
the
boy
hits
a
dog
(C)
np
(2)
the
boy
hits
a
dog

図 4.3: 解析木の構築

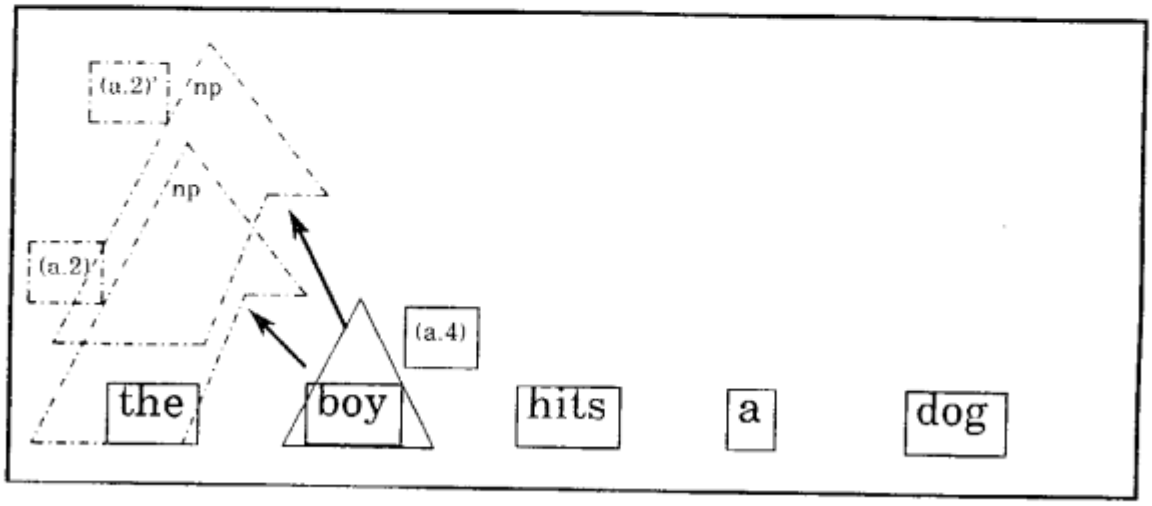


図 4.4: 横型解析

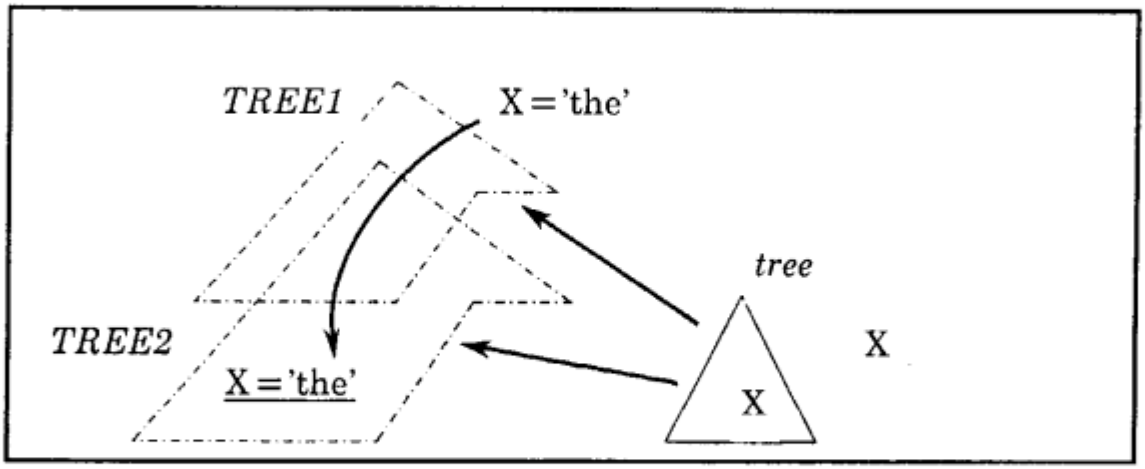


図 4.5: 変数の束縛

遅延補強項

機能

与えられたPrologの手続きを構文解析終了後に実行する。[3]

記述例

```
vp(vp(V,Np,Sem))  -->
        vt(V),np(Np),{agreement(V,Np)},
        &{construct_sem(V,Np)}.
```

[4]

遅延補強項は規則の最後に一つだけ"&"で区切って書く。

説明

構文解析過程において、文法的に不適当な解析木の生成を抑制するための条件を通常の補強項に記述し、枝刈りに役に立たない意味構造の生成などを遅延補強項に記述することによって構文解析中に無駄な手続きの処理を抑制することができる。

優先度計算

なぜ優先度が必要か？

一般に、自然言語文は複数の構文解析結果を持つことが多く、それらの間のもっともらしさの比較を行なう必要が生じる。SAXシステムでは、ユーザは文法規則の中にこのもっともらしさ(優先度)を計算する決定する規則を、記述することができ、システムはその規則に従って構文解析途中に優先度を計算する。またこの優先度規則のなかで、色々な情報を参照することが出来るしそのための記法も準備している。

優先度規則の書き方

図4.6に、優先度規則を含んだ文法規則の例を示す。下線を施してある部分(A),(B)が優先度規則である。

```
vp(vp(V,Np,Sem)) -->
  v(V)::{ eval_pref1(V,pref_cat,pref)},                    (A)
  np(Np)::{ eval_pref2(V,Np,pref_cat,pref)},               (A)
  &&{ pref_CAT = prefs(1) + prefs(2) }.                    (B)
```

図 4.6: 優先規則を含んだ文法規則

優先度の計算

SAXでの解析のやりかたを思い出していただきたい。上で述べたようにSAXでは図4.7のように、完成した部分木*tree*が上位の不完全な部分*TREE*に取り込まれることによって解析が進んでいく。

優先度の計算は次の二段階で行われる。

接合優先度

まず、文法規則の右辺に含まれる個々の文法カテゴリが、この文法規則に含まれるに当たっての優先度の計算を行う。ここではこの優先度を仮に接合優先度と呼ぶことにする。この接合優先度は、接合される下位の'完成した木'と上位の'不完全な部分木'との結びつきの良さと考えてよい。この接合優先度の規則は、文法規則中、上の例の規則のなかの(A)のように文法カテゴリのすぐ後ろに"::"で区切って記述する。接合優先度規則のなかでは、下位の木[5]の次に述べる部分木優先度が参照可能である。

接合優先度の値は予約語prefで返す。

次の二つの場合、接合優先度は、デフォルト値として下位の木の部分木優先度 をとる。

[3] 通常の補強項は、構文解析過程で実行される

[4] construct_sem は意味構造を構築する述語

[5] 4.7での*tree*

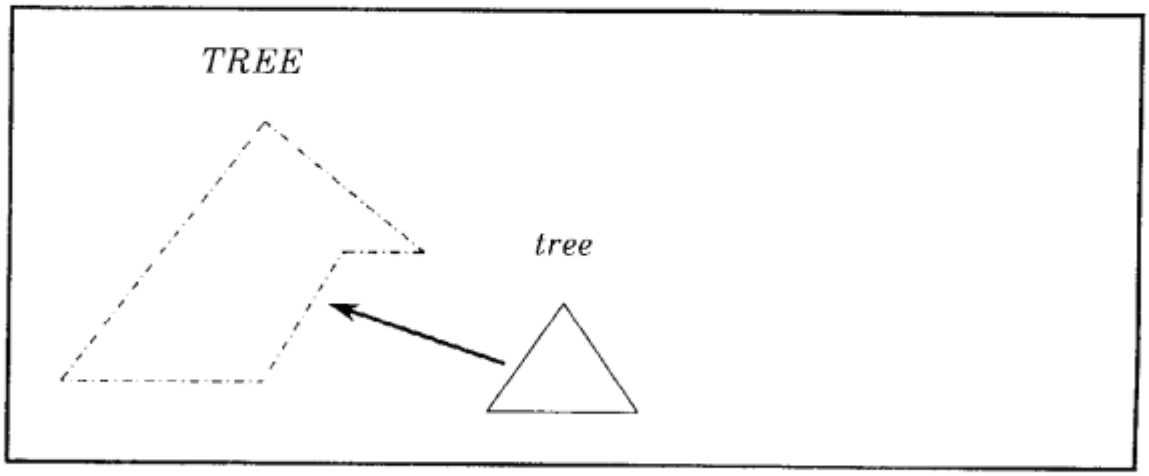


図 4.7: 解析木の構築 (その 2)

i. 接合優先度規則が書かれていなかった場合
ii. 接合優先度規則の実行が失敗した場合

部分木優先度

次に、有る部分木が完成した際にその部分木自身の優先度の計算を行なう。これを仮に部分木優先度と呼ぶことにする。この部分木優先度の計算には、自分に含まれる下位の i 番目の部分木との接合優先度を予約語*prefs(i)*で参照することができる。部分木優先度の計算規則は上の例の規則の中の (B) のように最後に "&&" で区切って記述する。

部分木優先度の値は予約語*pref_CAT*で返す。

次の二つの場合、部分木優先度は、デフォルト値として下位の木の部分木優先度 の合計をとる。

i. 部分木優先度規則が書かれていなかった場合
ii. 部分木優先度規則の実行が失敗した場合

優先度規則の中で参照可能な情報

優先度算規則のなかでは、次の情報が参照可能である。

二代上カテゴリ その規則の右辺のカテゴリを部分木として含むようなカテゴリを二代上カテゴリと呼ぶことにする。[6]この二代上カテゴリが予約語*super_cat_set*で参照できる。一般には、二代上カテゴリは複数個存在するので、その集合を Prolog のリストの形で返す。

右隣語品詞名 SAX に入力として与えられるのは、普通、形態素解析システム LAX の出力である。予約語*next_pos*により、現在処理を行っている語の右隣の語の品詞を参照することができる。

優先度機能まとめ

機能

文法規則に付加された優先度規則に従い各部分木の尤もらしさ (優先度) を構文解析途中に計算する。規則の中で優先度計算に有用な情報を提供する。

CIL の導入

[6]図 4.7の TREE を tree と見なしたときの TREE に相当する

参照可能情報	記法
自分より左のカテゴリが持つ引数	—
その部分木の優先度	pref_cat
二代上カテゴリ集合	super_cat_set
先読み情報を表す品詞名	next_pos
戻し値	pref

図 4.8: 予約語 (接合優先度)

参照可能情報	記法
自分より左のカテゴリが持つ引数	—
i 番目の tree の接合度	prefs(i),i= 1,2...
二代上カテゴリ集合	super_cat_set
解析木の文法カテゴリの集合	next_pos
戻し値	pref_CAT

図 4.9: 予約語 (部分木優先度)

SAX システムは補強項のなかに、CIL のゴール列の記述を許している。またこの補強項から呼びだされる手続きの中に CIL の記法を記述することも可能である。従ってユーザは、CIL で提供される機能により構文解析と意味処理を行ない、部分項の拡張された単一化を使うことができる。そうして単一化文法に基づいた見通しがよく柔軟な意味処理が可能である。部分項は部分意味情報を記述するのに適している。CIL を補強項に使った LFG 文法の例を下に示す。

```
sentence(Sent) --> np(Np),vp(Vp),{Sent!subj= NP,Sent= Vp}.

np(Np) --> det(Det),noun(Noun),{Np= Det,Np= Noun}.

vp(Vp) --> v(V),np(Np),{Vp= V,Vp!obj= Np}.

det(Det) --> [the],{Det= {spec/the}}.

noun(Noun) --> [girl],{Noun= {pred/girl}}.

v(V) --> [saw],{V= {pred/see(Subj,Obj),subj/Subj,obj/Obj,tense/past}}.

noun(Noun) --> [john],{Noun= {prepd/john}}.
```

図 4.10: CIL を使った LFG 文法記述の例

コメント

コメントの記法は、Prolog と同じであり、

i. 一行中で"%"より右の部分

ii. "/*"と"*/"でまれた領域

がコメントと見なされる。

宣言

オプション指定—

文法中に次の形式で宣言をしておくことで、トランスレータオプションメニュー[7]の初期値を設定することができる。

```
:- 宣言項目(値)
```

宣言項目	値	意味
esp_source	on off	ESP ソースファイル作成の有無
debug	on off	デバッグ，ノーデバッグモード
tree	on off	構文木情報の有無
preference	on off	優先記述の有無
cil	on off	CIL 記述の有無
cil_ec_interpret	on off	CIL インタープリターコード使用の有無
cil_ec_compile	on off	CIL コンパイルドコード使用の有無
translate_mode	translate catalogue save	カタログやセーブの実行

図 4.11: 宣言一覧

オプションメニュー表示の抑制—

また、次のように宣言しておくと、文法の変換の際にトランスレートオプションメニューが表示されず、ただちに変換が始まる。

```
:- menu(off)
```

[7] 2章を参照

参考文献

[1] M. Kameyama. *Zero Anaphora : The case of Japanese*. PhD thesis, Stanford University, 1984.

[2] D. Carter. *Interpreting anaphors in natural language texts*. Ellis Horwood Series in Artificial Intelligence. Ellis Horwood, 1987.

[3] V. Dahl. More on Gapping Grammar. In *Proceedings of the International Conference on FGCS '84*, pages 669–677, Tokyo, 1984.

[4] V. Dahl and H. Abramson. On Gapping Grammar. In *Proceedings of 2nd ICLP*, pages 77–88, Sweden, 1984.

[5] K. Kimura, R. Sugimura, T. Takizuka, and K. Mukai. Danwa Rikai Jikken System DUALS Dai 2-han no Sekkei to Jissô (Design and Implementation of Discourse Understanding System DUALS-v2 (*in Japanese*)). In *3rd Conference Proceedings of Japan Society for Software Science and Technology*, pages 33–36, Tokyo, 1986.

[6] Y. Matsumoto. Ronri Bunpô no Heiretsu Kôbun Kaiseki (Parallel Analysis of Logic Grammars (*in Japanese*)). *IPSJ*, 29(4):335–341, 1988.

[7] Y. Matsumoto and R. Sugimura. A Parsing System Based on Logic Programming. In *Proceedings of IJCAI 87*, 1987.

[8] Y. Matsumoto and R. Sugimura. Kôbun Kaiseki System SAX no tameno Bunpô Kijutsu Gengo (Grammar Description Language for the SAX Parsing System (*in Japanese*)). In *5th Conference Proceedings of Japan Society for Software Science and Technology*, pages 77–80, Tokyo, 1988.

[9] T. Okunishi, R. Sugimura, Y. Matsumoto, N. Tamura, T. Kamiwaki, and H. Tanaka. *Comparison of Logic Programming Based Natural Language Parsing Systems*, volume 2, pages 1–14. North-Holland, v. dahl edition, 1988.

[10] K. Morioka. Goi no Keisei (Formation of a Vocabulary (*in Japanese*)), volume 1 of *Gendai-go Kenkyuu*. Meiji Shoin, 1987.

[11] F. Pereira and D.H.D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231–278, 1980.

[12] H. Sano, K. Akasaka, Y. Kubo, and R. Sugimura. Go-kôsei ni motoduku Keitaiso Kaiseki (Morphological Analysis with Derivation and Inflection (*in Japanese*)). In *Proceedings of the 36th Conference of Information Processing Society of Japan*, 1988.

[13] R. Sugimura. Japanese Honorifics and Stuation Semantics. In *Proceedings of the 11th COLING*, pages 507–510, 1986.

[14] R. Sugimura. Ronri-gata Bunpô ni okeru Seiyaku Kaiseki (Constraint Analysis on Logic Grammars (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese society for Artificial Intelligence*, pages 427–430. Japanese Society for Artificial Intelligence, 1988. a prize-winning paper.

[15] R. Sugimura, K. Akasaka, Y. Kubo, H. Sano, and Y. Matsumoto. Ronri-gata Keitaiso Kaiseki LAX (Logic Based Lexical Analyzer LAX (*in Japanese*)). In *Proceedings of the Logic Programming Conference '88*, pages 213–222. ICOT, 1988. English version will appear in The Lecture Note on Computer Science.

[16] R. Sugimura, H. Miyoshi, and K K. Mukai. *Constraint Analysis on Japanese Modifying Relations*, volume II, pages 93–106. North-Holland, 1988.

[17] J. Tsujii. Bun-kaiseki system KGW+P no seigyo hôsiki (Controlling Methodology of Sentence Analysis System KGW+P (*in Japanese*)). In *4th Conference Proceedings of Japan Society for Software Science and Technology*, pages B–4–3, 1987.

[18] M. Watanabe. Kokugo kôbun-ron (Syntax of Japanese (*in Japanese*)). Hakama Shobô, 1971.

[19] S. Yamasaki, R. Sugimura, K. Akasaka, and Y. Matsumoto. Kôbun Kaiseki System SAX no Debug Kankyou (Debugging Environment of SAX System (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*, pages 411–414, 1988.

[20] T. Chikayama. ESP Reference Manual. Technical Report 044, ICOT, 1984.

[21] A. Colmerauer. Prolog-II: Reference Manual and Theoretical Model. Internal report, Group Intelligence Artificielle, Université d'Aix-Marseille II, 1982.

[22] ICOT. PSI/SIMPOS Editor Manual. 1988.

[23] T. Miyazaki and K. Taki. Multi-PSI ni okeru Flat GHC no Jitsugen Hôshiki (Installation of Flat GHC on Multi-PSI (*in Japanese*)). Technical Report 190, ICOT, 1986.

[24] K. Ueda. Guarded Horn Clauses. Technical Report 103, ICOT, 1985.

[25] J. Barwise. Recent Developments in Situation Semantics. In M. Nagao, editor, *Language and Artificial Intelligence*, pages 387–399, Amsterdam, 1987. North-Holland.

[26] J. Barwise and J. Perry. *Situations and Attitudes*. MIT Press, Cambridge, 1983.

[27]

[28] M. Dincbas, H. Simonis, and P.V. Hentenryck. Extending equation solving and constraint handling in logic programming. Internal Report IR-LP-2203, ECRC, 1987.

[29] T. Amanuma, T. Suzuki, T. Okunishi, and K. Mukai. CIL Programming kankyô (CIL Programming Environment (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*, pages 211–214, 1988.

[30] L. Cardelli. Typechecking Dependent Types and Subtypes. Technical report, DEC Systems Research Center, 1987.

[31] K. Mukai. Horn Clause Logic with Parameterized Types for Situation Semantics Programming. Technical Report 101, ICOT, 1985.

[32] K. Mukai. Unification over Complex Indterminates in Prolog. Technical Report 113, ICOT, 1985.

[33] K. Mukai. Anadic Tuples in Prolog. Technical Report 239, ICOT, 1987.

[34] K. Mukai. A System of Logic Programming for Linguistic Analysis based on Situation Semantics. In *Proceedings of the workshop on semantic issues in human and computer languages*. CSLI, 1987.

[35] K. Mukai. Partially Specified Term in Logic Programming for Linguistic Analysis. In *Proceedings of the International Conference on FGCS '88*, 1988.

[36] K. Mukai and H. Yasukawa. *Complex Indeterminates in Prolog and its Application to Discourse Models*, volume 3, pages 441–466. OHMUSHA, ltd. and Spring Verlag, 1985.

[37] J. Reynolds. Three approaches to type structures. *Lecture Note in Computer Science*, volume 185, Springer Verlag, 1985.

[38] K. Sakai and Y. Sato. Boolean Gröbner bases. Technical Memo 488, ICOT, 1988.

和文索引

英文索引

LTB 操作説明書

— 文生成編 —

(LTB1.1 版)

ICOT 第二研究室

平成元年 3 月 31 日

本説明書は文生成の操作方法について記述している。

- 文生成部の概要
- 文生成部の操作方法
- 文生成のための規則

汎用日本語処理系 LTB(Language Tool Box) の説明書は次のような構成になっている。

- LTB 概要編
- LTB マスタ辞書編
- CIL 編
- LAX 編
- SAX 編
- 文生成編

目次

図目次

表目次

第 1 章

文生成部の概要

本文生成部で考えている「文生成」とは、文のもつ構文の情報をもとにして表層文を生成することである。すなわち、表層文から文の係り受け関係や構文木などを生成する文解析のちょうど逆の働きをいう。しかしながら、文解析の入出力を逆に用いれば文生成が実現されるかというと、そう簡単なものではない。そこで、本文生成部には、文生成のためだけに特化したアルゴリズムを用意してある。また、文生成を行うためのさまざまなソフトウェア環境も文生成に専用のものを設計した。

本章では、本マニュアルを使用するために、ユーザが最小限知っておくべきことについて説明する。

1.1 文生成の概念

本文生成部における文生成の概念を図1.1に示す。文生成のもととなる構文的情報を担う表現を「中間表現」と呼ぶことにする。入力した中間表現を表層文に変換するには対象言語に関する知識が必要となる。文生成は、対象言語に関する知識 (言語知識) を参照しながら行われる。

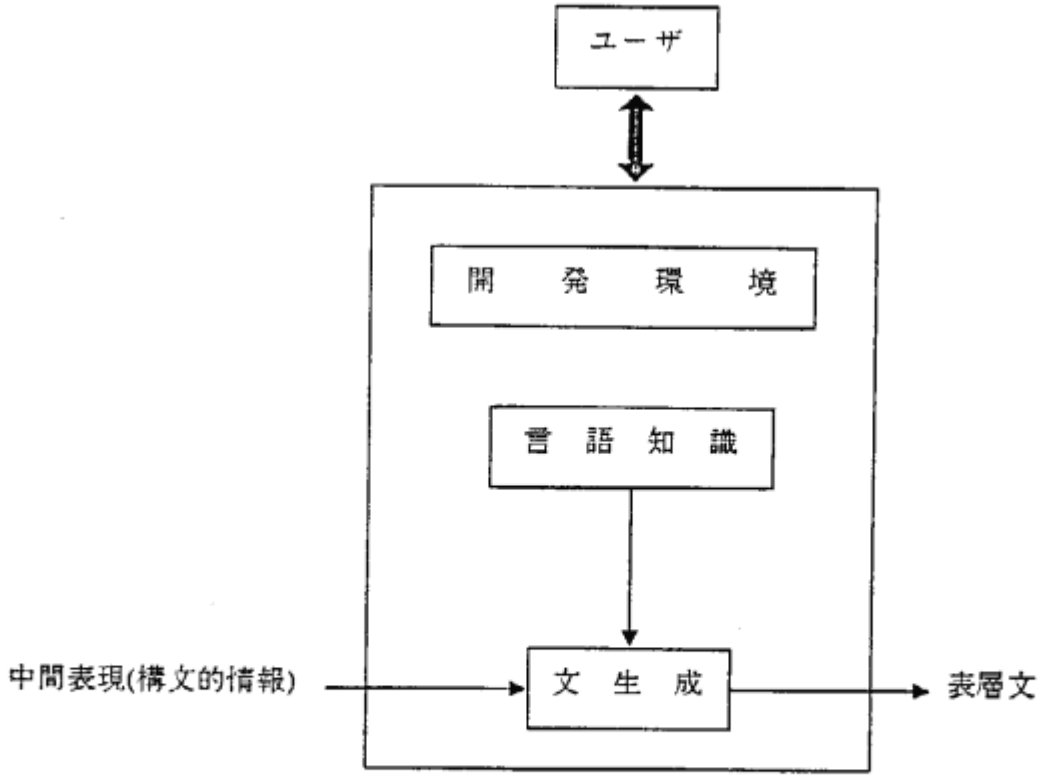


図 1.1: 文生成の概念図

1.2 文生成部の目的およびその実現

本文生成部の目的は、以下のとおりである。

1. 中間表現から表層文を生成する機構を提供する。

2. 言語知識を計算機上にソフトウェアとして実現するための 開発環境を提供する。

1の実現のために文生成を行う生成エンジンを、 2の実現のためにさまざまな文生成用ツール を用意した。

1.3 本文生成部における文生成の基本的な考え方

汎用的な文生成システムを設計する上で、大きな障害となる要素のひとつに、「入力となるデータを予め定めておくことが難しい」ということがある。ひとつの文が持つ情報を表現するにもいく通りかの表現形式が考えられる。たとえば、「太郎が花子を殴る」の情報を表現するのにも、次のようないく通りかの表現が考えられる。

1. { right/{ right/ 殴る,
   left/{ right/ を,
   left/ 花子}},
   left/{ right/ が,
   left/ 太郎}}

2. { 述語 / 殴る,
   格 /[{が, 太郎} ,{を, 花子}]}

3. { 関係 / 殴る,
   ロール /{ 行為者 / 太郎,
   対象 / 花子}}

1は、二分木による表現であり、係り受けの構造および文要素が表層に現れる順序を明示したものである。 2は、述語「殴る」に対して「太郎」、「花子」が格要素として現れる順序と格助詞とを明示した表現方法であり、1より抽象度が高い。 3は、述語が「殴る」であるということは明示しているが、格要素となる「太郎」、「花子」に付加する格助詞や、それらの表層上での順序を明示しない表現方法であり、2より抽象度が高い。

このようなさまざまな表記法に対応できるように、本文生成部は、次のような方針のもとに設計を行った。

> 抽象度が低い、すなわち、情報が詳細にわたって明示的に表現されている表現 (「基本表現」と呼ぶ) のみをシステム側で定義しておき、抽象度の高い表現を用いたい時は、システム定義の上にマクロを定義する。

この「マクロ」の考え方が、本生成方式の特徴となっている。図1.2に、この「マクロ展開」の概念を示す。生成すべき文の情報を抽象的に表現したマクロ記述を、マクロ展開機能はマクロ定義を参照しながら展開し、基本となる表現に変換する。このうち、マクロ展開機能が文生成の機能の一部であり、マクロ定義が言語知識の一部として計算機上に「マクロ展開規則」として記述される。マクロ定義を書き換えたり、すでにあるマクロ展開規則を利用してさらに新しいマクロ展開規則を付加することにより、中間表現の仕様をユーザの望み通りに決定することができる。

マクロ定義は、宣言的な記述により行われるならばユーザにとって設計しやすいものとなるが、現在のところそうはなっていない。現在マクロ定義は CIL上に記述されるプログラムとして組み込む枠組になっている。辞書を引くためのプログラムなど、マクロ定義を追加・変更しても、共通に使用できるユーティリティについては、マクロ展開機能のひとつとしてユーザに開放されている。また、システム側で予め作成した標準的なマクロ定義を「標準マクロ」として用意してある。

マクロ展開規則以外の言語知識としては、辞書の形で表現される単語に関する知識、および形態素生成用テーブル類の形で表現される単語の活用・接続形態に関する知識がある。

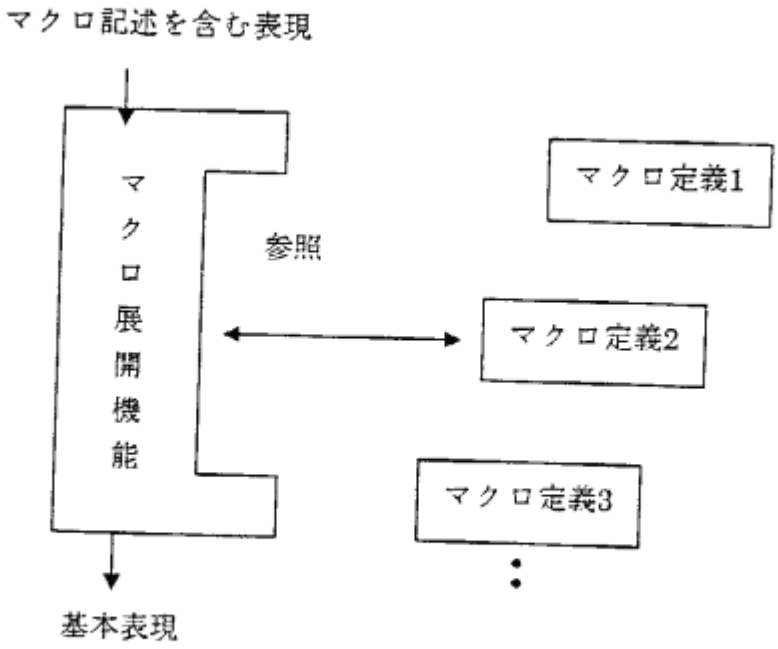


図 1.2: マクロ展開の概念図

1.4 文生成の流れ

生成の流れについて説明する。

文生成は、まず、入力された中間表現に含まれるマクロ記述を展開し、基本表現に変換することから始まる。このとき、文生成用辞書に記述された単語知識を利用して表層に現れる単語の語彙情報を構文構造とともに組み上げていくことにより、「表層構造」を生成する。表層構造には、表層文に現れるすべての単語についての語彙情報と、それらの単語どうしがどのように構成素を成しているかなどの情報が含まれている。つぎに、接続表および活用表を参照しながら、マクロ展開の結果得られた表層構造に含まれる各単語に活用を施し、表層文として形を整える。

1.5 文生成部の機能

文生成部の目的およびその実現の項で述べたとおり、本文生成部の目的のひとつは、文生成の各局面でシステムが参照する種々の知識をソフトウェアとして開発するための開発環境を提供することである。そこで、本生成部では前項に述べた機能、

- マクロ展開機能
- 形態素生成機能

のほかに、対話的に開発作業をすすめるためのつぎの機能を提供している。

- 文生成に使用する種々の規則を選択する機能
- 文生成の各局面でシステムの入力となるデータを選択 する機能
- マクロ展開、形態素生成のシステム動作を追跡・制御 する機能
- 入力された中間表現の構文や内容が正当なものかどうかを チェックする機能
- 中間表現や表層構造を、画面上に見易く表示し、 編集を支援する機能
- 表層表現を視覚的にとらえやすい形態で表示する機能
- 以上の機能を統括する マンマシン・インタフェース機能

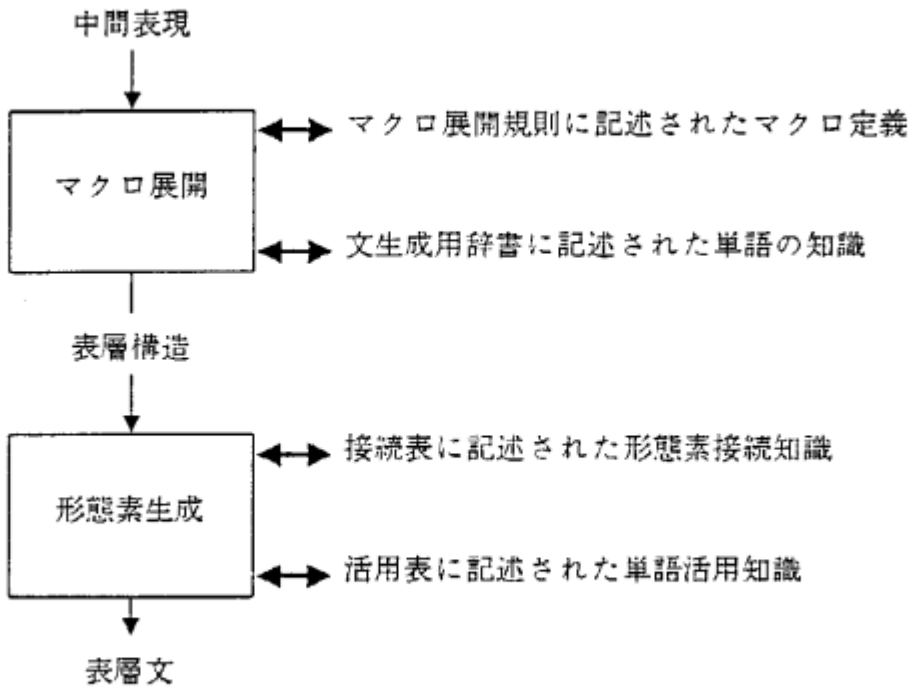


図 1.3: 文生成の流れ

1.6 文生成部の構成

文生成部は大きく分けて、文生成用ツールと文生成機構とから構成される。文生成部の全体構成を図1.4に示す。文生成用ツールは、ユーザに対して文生成機構とのインタフェースを提供する。即ち、ユーザは文生成ツールを通して、入力データ (中間表現、表層構造) の選択・編集、実行制御(文生成機構のトレース、実行規則の選択)、結果表示(表層構造、表層文) を行う。

- 入力データの選択・編集
  デバッガや中間表現 / 表層構造の編集・表示機能は、入力データのデータ・ディレクトリ内からの選択機能を提供する。更に、中間表現 / 表層構造の編集・表示機能は対象データを編集画面 (pmacs ウインドウ) 上にプリティ・プリントし、ユーザに pmacs エディタの機能を提供する。

  中間表現チェッカは、対象中間表現に対して、文生成機構内部のマクロ定義表を参照して構文、辞書登録の有無等のチェックを行う。

- 実行制御
  デバッガはユーザに CIL デバッガの機能を提供する。それにより、文生成機構のトレースや実行規則の選択が可能となる。その他、複数の入力データの一括処理機能も提供している。

- 結果表示
  中間表現 / 表層構造の編集 / 表示機能はマクロ展開の結果できた表層構造や形態素生成の結果である表層文を文生成用ツールの トップ・ウインドウ (→2.1) (又は、ユーザが指定したファイル) にプリティ・プリントする。木構造表示機能は表層構造のグラフィック表示機能を提供する。

文生成機構は、入力中間表現を表層構造に展開し、更に表層構造に形態素処理を施して表層文を生成する。

- 入力中間表現の展開
  入力中間表現の展開はマクロ展開部で行われ、マクロ展開部はマクロ展開エンジンとマクロ展開規則から構成される。

  マクロ展開規則はマクロ表記を含んだ中間表現 (マクロ表現) を基本表現に展開する。即ち、ユーザは中間表現として自分が使いたいマクロ表記を、マクロ展開規則の形でマクロ定義表にあらかじめ定義しておく必要がある。

マクロ展開エンジンは文生成用辞書の検索等を行って語彙データを構成し、基本表現を表層構造に変換する。但し、現行のマクロ展開規則は宣言的には記述できず、プログラムとして手続き的に記述しなければならない。

- 形態素処理
  形態素処理は形態素生成部で行われ、形態素生成部は形態素生成エンジンと形態素生成用テーブル類から構成される。

  形態素生成エンジンは、マクロ展開部より受け取った表層構造から形態素生成用テーブル類を参照して活用語の語尾の決定等を行う。形態素生成用テーブル類には接続表と活用表があり、いずれも活用語の活用情報が記載されている。

1.7 本マニュアルの構成

本マニュアルに掲載するおおまかな内容と掲載箇所は、以下のとおりである。

- 文生成部全体の操作 →2 章
  - 文生成部の起動・終了方法 →2.1
  - 文生成用ツールの核となる文生成デバッガの機能 →2.2
  - 文生成デバッガのメニュー項目別操作方法 →2.3
  - コマンド一覧 →2.4
  - 出力メッセージ一覧 →2.5
- 文生成用規則類の作成手順 →3 章
  - マクロ展開規則の作成手順 →3.1
  - 文生成用辞書の作成手順 →3.2
  - 形態素生成用テーブル類の作成手順 →3.3

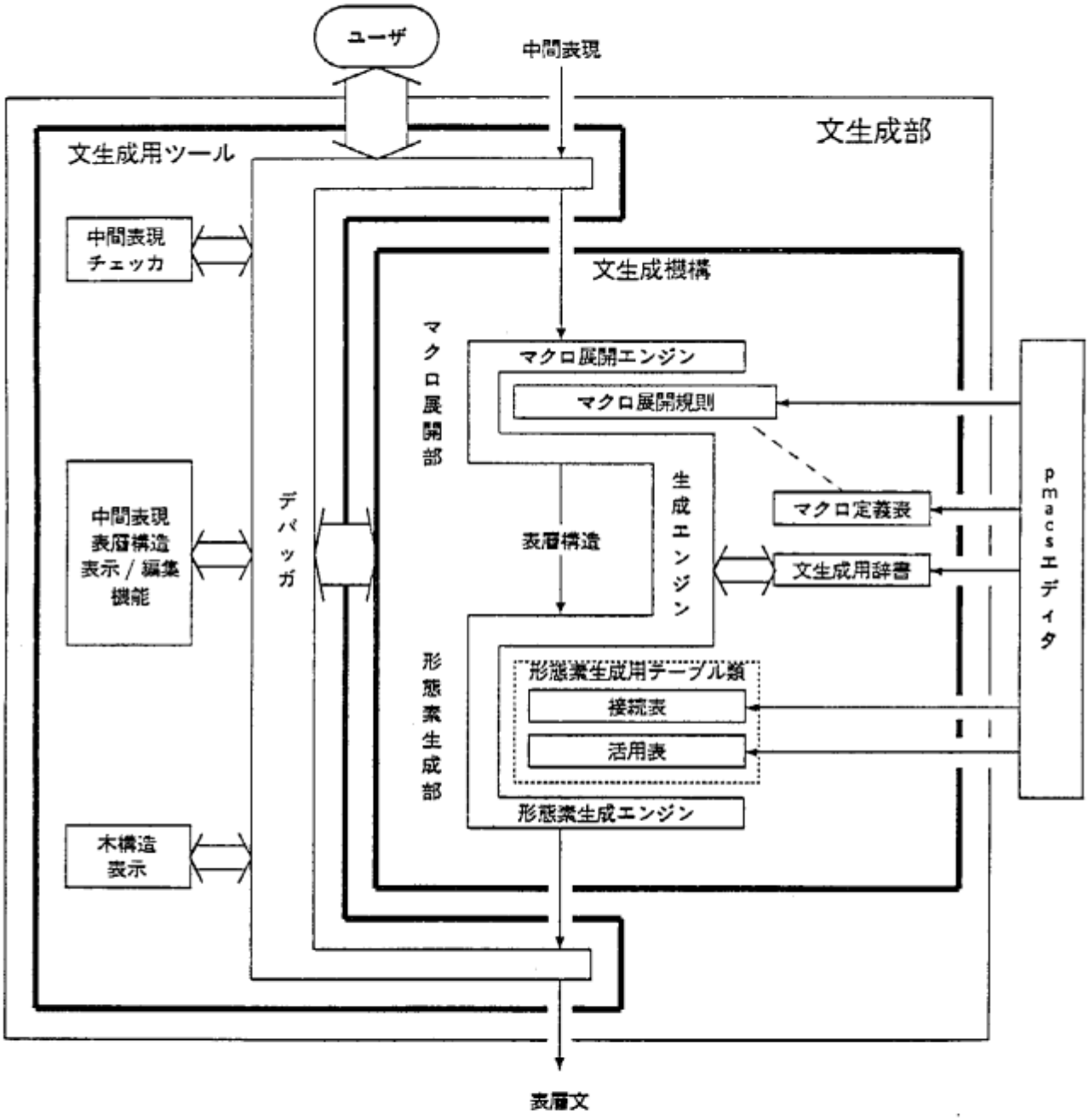


図 1.4: 文生成部の全体構成

第 2 章

文生成部の操作方法

文生成用ツールは、ユーザと文生成機構とのインタフェースであり、

- 文生成デバッガ
- 中間表現表示
- 基本表現表示
- 中間表現チェッカ

などから構成される。
文生成部の操作は、文生成デバッガのトップ・ウインドウを通じて、対話的に行なわれ、文生成デバッガが、各ツールの呼び出しを行なっている。

2.1 文生成部の起動と終了

文生成デバッガは、SIMPOS デバッガから起動される。(2.1.1起動方法参照)
起動直後に、初期処理を行なう。 初期処理では、以下の処理が行なわれる。

- トップ・ウインドウの生成・表示
  図 2.1のようなウインドウが生成・ 表示される。 以後、このウインドウのことをトップ・ウインドウと言い、このウインドウ上でメニューを選択することにより、 文生成部の操作が行なわれる。

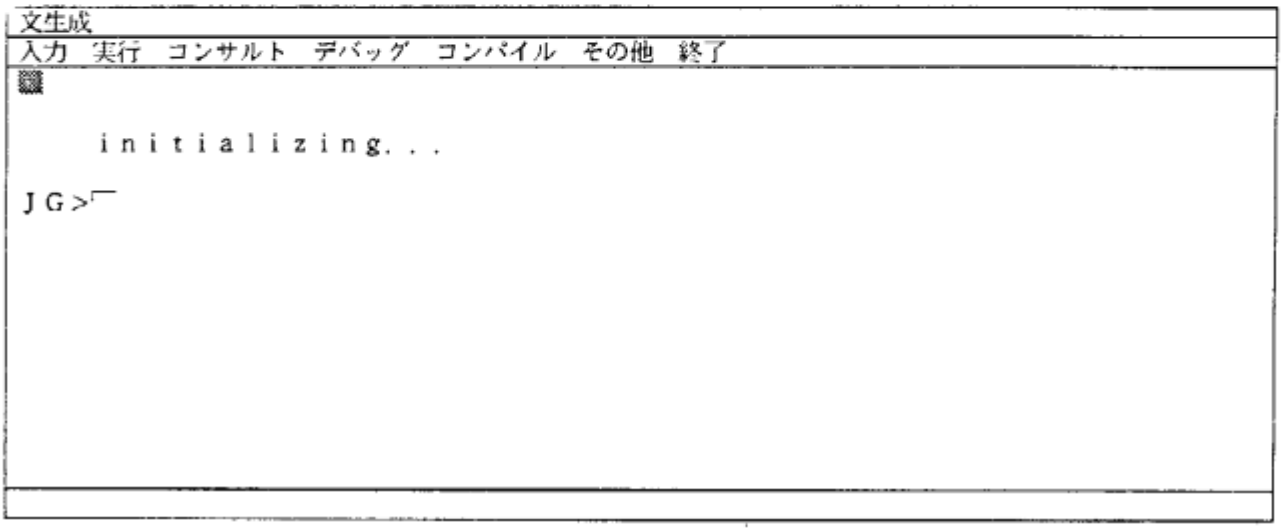


図 2.1: トップ・ウインドウ

- デフォルト・パラメータの処理
  デフォルト・パラメータに記述された指定にしたがって、環境設定などを行なう。 とくに、生成エンジンのコンサルト[1]の実行に対しては、次のようなメッセージがトップ・ウインドウ上に表示される (図 2.2)。

2.1.1 起動方法

SIMPOS デバッガで、

```
:jg(#jg).
```

と入力することにより、文生成デバッガが起動される。
ただし、この時 SIMPOS デバッガのデフォルト・パッケージは、“user” あること。

2.1.2 デフォルト・パラメータ

デフォルト・パラメータは、 ユーザが文生成デバッガ起動時の環境設定を行なえるようにするためのものである。

1. デフォルト・パラメータ・ファイル
   デフォルト・パラメータが記述されているファイルのことをデフォルト・パラメータ・ファイルといい、

   ```
   >sys>user>jg_v10>defparam.dat
   ```

   に、固定されている。

[1] 2.2.2参照

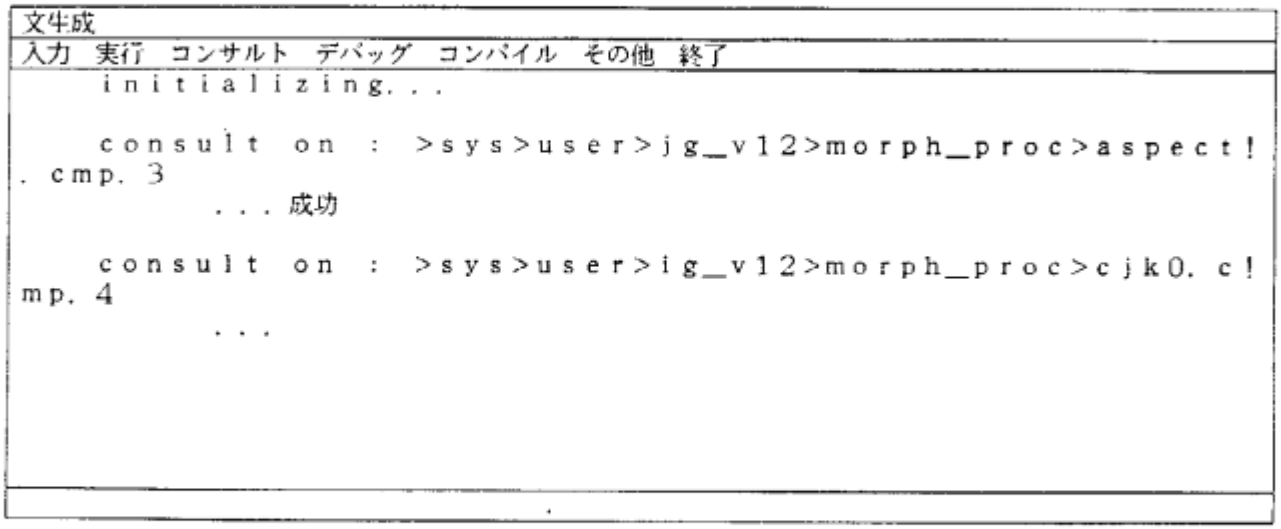


図 2.2: デフォルト・パラメータ処理

2. デフォルト・パラメータの機能

- ディレクトリの設定
  データ・ディレクトリ、生成エンジンが参照する規則 (マクロ展開規則、文生成用辞書辞書、形態素生成用テーブル類) を格納するディレクトリの設定を行なう。
- 起動時のコンサルト
  文生成デバッガ起動時に、生成エンジン規則類のコンサルトを行なう。
- 中間表現プリティ・プリントの深さ指定
  中間表現のプリティ・プリント表示の深さ指定を行なう。

3. デフォルト・パラメータの記述形式
   デフォルト・パラメータ・ファイルに書かれる内容は、1 個のリストであり、リストの各構成要素として、以下で定義される個々のデフォルト・パラメータが記述される。

   (a) ディレクトリの設定の場合
   デフォルト・パラメータは、

   default_directory(識別子, ディレクトリ・パス名)

   の形である。識別子とデータ種別、生成エンジンとの対応は、表 2.1で与えられる。ディレクトリ・パス

表 2.1: デフォルト・パラメータの識別子

ディレクトリの種別	識別子
中間表現	chuukan
表層構造	kihon
マクロ展開	macro_expand
形態素生成	string_synthe
辞書	dictionary

名は、ストリングである。

(b) コンサルトの場合
デフォルト・パラメータの形式は、

consult_on(識別子, ファイル名)

である。 識別子は、表 2.1で与えられた通りであるが、コンサルトの場合には、生成エンジン (マクロ展開 / 形態素生成 / 辞書) のみが許される。
ファイル名は、ストリングである。

(c) 中間表現プリティ・プリントの深さ指定の場合
デフォルト・パラメータの形式は、

display_depth(chuukan,3)

である。 深さは、自然数値である。 中間表現の深さがこの制限値を越えた部分は、 ドット (…) により省略表示される。

(d) 記述上の注意
ディレクトリの設定は、コンサルトの前に記述されている必要がある。 (図 2.3 デフォルド・パラメータの記述例参照)

4. デフォルト・パラメータの記述例

```
[default_directory(macro_expand,">sys>user>jg_v12>macro_expand"),
 default_directory(dictionary,">sys>user>jg_v12>dictionary"),
 consult_on(dictinary,"kdic3.cmp"),
 default_directory(dictionary,">sys>user>jg_v12>dictionary"),
 consult_on(string_synthe,"kat3.cmp"),
 display_depth(chuukan,3)]
```

図 2.3: デフォルト・パラメータの記述例

2.1.3 終了方法

文生成デバッガのトップ・ウインドウのメニューで

"終了"

を選択することにより、文生成デバッガが終了し、トップ・ウインドウは消去される。

2.2 文生成デバッガの機能概要

文生成デバッガは、次のような機能を持つ。

- モードの設定
  中間表現、表層構造、表層文の入出力モードなどの設定を行う。(2.2.1 モード設定参照)
- 対象ファイル・規則の切り替え
  辞書、マクロ展開部、形態素生成部の規則のコンサルトを行う。(2.2.2 対象ファイル・規則の切り替え参照)
- デバッグ機能
  データの入出力・編集、生成実行過程の追跡 (トレース) などを行う。(2.2.3 デバッグ機能参照)
- 中間表現チェック
  中間表現の、構文チェックを行う。(2.2.4 中間表現チェック参照)

2.2.1 モード設定

文生成部で対象として扱われるデータは、次の 3 種類である。

- 中間表現
- 表層構造
- 表層文

文生成デバッガは、以上の 3 種類のデータに対して、入力、表示の機能を提供し、更に、多様な入出力対象の選択を可能にしている。 ユーザは、モード設定によって、これらの入出力対象などを指定できる。
3 種類のデータ種別の各々に対して、設定できる項目と内容を次に述べる。

1. 中間表現

   (a) 入力対象の設定
   中間表現をファイルまたは、ストリームから入力できる。 ファイル入力の場合、ディレクトリは、デフォルト・パラメータにより設定される。 デフォルト・パラメータにより明示的に指定されていなければ、既定値として、

   ```
   >sys>user>jg_v10>macro_expr
   ```

   が設定される。

   (b) 中間表現チェッカの設定

   - 構文チェック・フラグの設定
     中間表現データ入力時に、構文チェックを行う / 行わない、の指定をする。
   - エラー表示の選択
     構文チェックを行った時、エラー表示をプリティ・プリントつきで行うか、単にエラー種別の表示を行うかという指定ができる。

   (c) 出力対象[2]の設定
   ウインドウ、ファイル、ストリーム、ヴォイドの中から出力先を選択できる。

   - ウインドウは、文生成デバッガのトップ・ウインドウへの出力である。
   - 出力先がファイルの場合、ファイル名は、1eで述べる出力ファイルによって設定される。
   - ヴォイドが選択された場合、どこにも出力されない。

   (d) 中間表現プリティ・プリントの設定
   プリティ・プリントの深さをキーボードより自然数値を入力することにより指定できる。

   (e) 出力ファイルの設定
   出力対象がファイルの場合に使用されるファイル名を、キーボードから入力する。 パス名に相当するディレクトリがない場合には、入力対象の設定 (1a) で述べたデフォルト・ディレクトリが使用される。 ワールド名を使うことも出来る。

2. 表層構造
   表層構造でモード設定できる項目は次の通り。

   (a) 入力対象の設定

   (b) 出力対象の設定

   (c) 出力ファイルの設定

   これらの内容は、中間表現の場合 (第 1項 中間表現参照) と同様である。

3. 表層文
   表層文でモード設定できる項目は、次の通り。

[2]中間表現の出力対象とは、データ入力時のデータ表示先のことである。

(a) 出力対象の設定

(b) 出力ファイルの設定

これらの内容は、中間表現の場合 (第 1項 中間表現参照) と同様である。

2.2.2 対象ファイル・規則の切り替え

文生成デバッガでは、文生成機構の対象ファイル・規則の切り替えをファイルのコンサルト・オン／オフをもちいて、行っている。 ここで、コンサルト・オンとは、ファイルのコンサルトを行うことであり、コンサルト・オフとは、コンサルトの解除のことである。 生成エンジンは、

- マクロ展開部
- 形態素生成部
- 辞書部

の3種類に分類できる。 これらに属する規則の切り分けは、各々に対応するSIMPOSのディレクトリを 設定し、ファイルの管理を行なうことにより実現される。 ディレクトリの設定は、デフォルト・パラメータによって行なわれる。 ただし、デフォルト・パラメータによって明示的に指定されない場合には、 既定値として、次のように設定される。

- マクロ展開部 >sys>user>jg_v10>macro_expand
- 形態素生成部 >sys>user>jg_v10>morph_proc
- 辞書部 >sys>user>jg_v10>dictionary

2.2.3 デバッグ機能

文生成デバッガは、ユーザ定義データである中間表現データ、表層構造データ、マクロ展開規則、形態素生成規則の検証・解析の道具として、デバッグ機能を有する。 デバッグ機能の具体的な内容は、

- データの編集
- データの入出力
- データの構文チェック
- 実行過程の追跡

である。

ここで、データとは、中間表現データと表層構造データである。
一般に中間表現データは、ユーザ定義のデータであり、構文チェック機能は、生成実行前のチェックの道具として有効である。 表層構造データは、中間表現データにマクロ展開規則を適用することにより得られるものである。 表層構造データに対する構文チェックの機能はない。

データの編集

中間表現データと表層構造データの編集の機能は、以下の通りである。

- 中間表現データの編集
  - pmacs の機能
  - データ・バッファ [3] 内のデータのロード、セーブ
  - 中間表現チェッカ
  - ファイルからの直接入力
  - ファイルへの出力
  - プリティ・プリント
- 表層構造データ
  - pmacs の機能
  - データ・バッファ 内のデータのロード、セーブ
  - グラフィック表示

データの入出力

文生成デバッガの外部から入力されたデータは、データ・バッファ に保存され、更に出力部に送られる。 以下に、データの入力、保存、出力に関して述べる。

1. データの入力
   文生成デバッガ外からのデータの入力方法には、
   - ストリームからの入力
   - ファイルからの入力

   があり、その選択は、モード設定によって行なわれる。 ファイルからの入力を行なう場合、メニュー形式でファイル名を選択できる。 また、ディレクトリを用いてデータ種別の切り分けを行なっている。例えば、中間表現データと表層構造データのディレクトリの既定値は、各々次のように定められている。
   - 中間表現 >sys>user>jg_v10>macro_expr
   - 表層構造 >sys>user>jg_v10>primitive

[3] 入力もしくは生成された、 中間表現データや表層構造データを 一時的に保存するために、 各々のデータ種別に対応する データ・バッファがある。

これらのディレクトリは、デフォルト・パラメータを予め設定することにより変更できる。ただし、ディレクトリの設定は、文生成デバッガ起動時に行なわれ、それ以後は変更出来ない。

2. データの保存
人力もしくは生成された、中間表現データや表層構造データを一時的に保存するために、各々のデータ種別に対応するデータ・バッファ がある。 いったんデータ・バッファに保存されたデータを編集したり、生成実行を行なうことが出来る。 データ・バッファの内容が書き変わるのは、次に示す場合である。

- 中間表現データ・バッファの内容が書き変わる場合
  - データ入力
  - データ編集
    ただし、編集後にデータがデータ・バッファに送られなければ、書き変わらない。
- 表層構造データ・バッファの内容が書き変わる場合
  - データ入力
  - データ編集
    ただし、編集後にデータがデータ・バッファに送られなければ、書き変わらない。
  - 表層構造生成
    中間表現データから表層構造データを生成した場合。

3. データの出力
データの出力対象は、次の 4 種類である。

- ウインドウ
  トップ・ウインドウに出力される。
- ファイル
  指定されたファイルへの出力。
- ストリーム
  ストリームへの出力。
- ヴォイド
  どこにも出力されない。

これらの出力対象は、モード設定によって設定される。ただし、既定値は window である。 データの出力方法は、データ種別によって若干異なる。

- 中間表現データのプリティ・プリント
  中間表現データのプリティ・プリントが使われるのは、以下の場合である。
  - データ入力
    データ入力時に入力データのプリティ・プリントが行なわれる。
  - 編集
    中間表現データ・バッファから、編集用バッファにデータをロードした時、プリティ・プリントが行なわれる。 また、中間表現データの編集時に、編集されたデータのプリティ・プリントを行うコマンドが編集用メニューに用意されている。
- 表層構造データのプリティ・プリント
  表層構造データのプリティ・プリントが行なわれるのは、以下の場合である。
  - データ入力
    データ入力時に、入力データのプリティ・プリント表示が行なわれる。
  - 表層構造データ生成
    表層構造データ生成時に、生成結果のプリティ・プリント表示が行なわれる。
  - 編集
    表層構造データ・バッファから編集用バッファにデータをロードした時、プリティ・プリント表示が行なわれる。
- 表層構造データのグラフィック表示
  表層構造が木構造の形でグラフィック表示される。

実行過程の追跡

マクロ展開部または形態素生成部のファイルに対して、デバッグ・オンの指定をすることにより、その生成の実行過程を追跡(トレース)することができる。 実行過程の追跡では、CIL デバッガが使われている。
以下では、生成実行、デバッグ・オン / オフの切り替え、コンパイルについて述べる。

1. 生成実行
   実行には、4 種類ある。

   - 中間表現⟹表層構造
     中間表現データから表層構造データを生成する。
   - 中間表現⟹表層文
     中間表現データから表層文を生成する。
   - 表層構造⟹表層文
     表層構造データから表層文を生成する。
   - 連続処理
     上記 3 種類の生成実行を複数個のデータに対して行う。 また、この 3 種類の生成実行の種別を連続処理の種別という。

   いずれの場合でも、 生成エンジンのファイルの中に、 デバッグ・オンの状態のものがあれば、 CIL デバッグ・ウインドウが表示され、 実行のトレースが行える。

2. デバッグ・オン / オフ
   マクロ展開部、形態素生成部のプログラムのファイルをメニューで指定して、デバッグ・オン / オフを行う。ただし、 指定できるファイルは、既にコンサルト・オンされているファイルで、ファイル拡張子が "cil" のもののみである。 また、既にデバッグ・オンの状態のファイルは、メニューでは反転表示されている。

3. コンパイル
   マクロ展開部、形態素生成部または辞書のファイルを指定して、コンパイルを行う。ただし、指定できるファイルは、ファイル拡張子が "cil" のもののみである。

2.2.4 中間表現チェッカ

1. 概要

(a) 目的

中間表現チェッカは、生成用中間表現の記述形式、記述内容を定義に照らし合わせて、その内容が妥当なものかどうかをチェックするものである。
ユーザが独自に構築したマクロ表現による中間表現データを、生成エンジンに渡す前に、記述のチェックを行なうことが出来る。
中間表現チェッカには、中間表現全体をチェックする全体チェックと、中間表現の一部分をチェックする部分チェックとがある。
部分チェックは、中間表現作成支援（マクロエディタ）から使用される。
チェック内容と、マクロ表現の構造の定義は、マクロ定義表として書かれている。

(b) チェック方法

生成用ツールとしては、今のところ全体チェックのみサポートしているため、部分チェックについては説明を省略する。
チェックは、マクロ定義表及び、それに関する制約を用いてチェックを行なう。
まず、定義に関する制約によるチェックを行なう。そのチェックでエラーが検出されると、直ちにチェックを中止し、そのエラー内容をユーザに知らせる。
制約がリストで与えられている場合は、制約リストの要素順にチェックを行なう。チェックの際には、リスト要素の関数に対して、チェックすべきマクロ表現データ、定義表現及び、エラーコード値が、返る引数を付加して関数を再構成して、該当する関数により、チェックを行なう。
次に、定義表現にしたがってマクロ表現の全ての属性構造をチェックする。但し、定義表現が、変数又は、 [A,B,...]*etc の場合には、定義によるチェックは行なわない。

(c) マクロ定義表

マクロ定義表は、マクロ表現の構造の定義と、属性名に対する制約を定義したもので構成される。
現在のマクロ定義表の記述仕様を以下に示す。

```
jgmcr_table(<マクロ名>, % 属性名又は、便宜上のテンプレート名
            <定義>,
            <定義に関する制約>,
            <テンプレート作成フラグ>).
<定義> ::= <定義表現>
<定義表現> ::= <リスト表現> |
                <＊リスト表現> |
                <etc 表現> |
                <t 表現> |
                <部分項表現> |
                <関数表現> |
                アトム |
                変数
<リスト表現> ::= [<定義表現>,...]
                   % リスト中の何れかの表現が可能なことを示す。
<＊リスト表現> ::= *[<定義表現>,...] | *[<... 表現>]
<... 表現> ::= <定義表現>...
               % リストを値として取ることを示す。
               % リストの各要素は、対応するリスト要素の表現を
               % 規定する。リスト要素の定義が <... 表現> の
               % 時には、リストの全ての要素がその定義表現によ
               % って規定されることを示す。
<etc 表現> ::= [<定義表現>,...]*etc
               % リスト中に表現例が列挙される。
```

```
<t表現> ::= t(<マクロ名>)
                % マクロ名の定義表現を参照することを示す。
<部分項表現> ::= {<属性名> / <属性値>,...}
<属性名> ::= <t表現> | アトム
<属性値> ::= <定義表現>
<関数表現> ::= set(<定義表現>) | アトム(<定義表現>)
<定義に関する制約> ::= <制約> | [<制約>,...]
<制約> ::= <定義表現>を規定する任意の個数
<テンプレート作成フラグ> ::= yes | no
```

% マクロエディタで使用。

% マクロ定義表の内容をテンプレート登録するかどうかのフラグ。

2.3 文生成デバッガの操作方法

文生成デバッガの各機能は、トップ・ウインドウ上のメニューから各機能に対応した操作コマンドが選択されることにより実行される。
文生成デバッガの基本的な機能は次の通りである。

- 入力
  中間表現、表層構造の入力を行なう。(2.3.1 入力参照)
- 実行
  生成実行を行なう。(2.3.2 実行参照)
- コンサルト
  生成エンジンのコンサルト・オン / オフを行なう。(2.3.3 コンサルト参照)
- デバッグ・オン / オフ
  生成エンジンのデバッグ・オン / オフを行なう。(2.3.4 デバッグ参照)
- コンパイル
  生成エンジンのコンパイルを行なう。(2.3.5 コンパイル参照)
- その他
  - 編集
    中間表現、表層構造の編集を行なう。(3 編集参照)
  - モード設定
    各種のモードの設定を行なう。(2 モード設定参照)
  - 連続処理
    生成実行を複数個のデータに対して行う。(1 連続処理参照)
- 終了
  文生成デバッガを終了する。(2.3.7 終了参照)

2.3.1 入力

1. トップ・ウインドウで

"入力"

を選択すると、データ種別を項目とするサブ・メニューが表示される (図 2.4)。 以下では、中間表現の場合に関して述べるが、表層構造の場合も同様である。

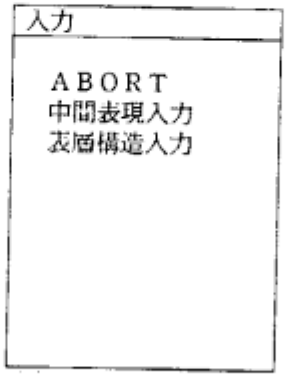


図 2.4: データ種別

2. サブ・メニューで

"中間表現入力"

を選択すると、入力モードがファイルの場合、ファイル名を項目とするメニューが表示される (図 2.5)。

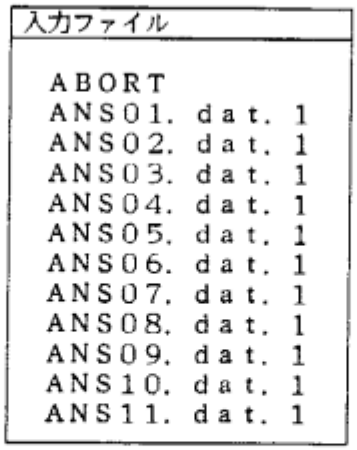


図 2.5: ファイル名メニュー

3. ファイル名メニューの中から、入力したいデータのファイルを選択すると、データが入力され、入力されたデータがトップ・ ウインドウにプリティ・プリント表示される (図 2.6)。

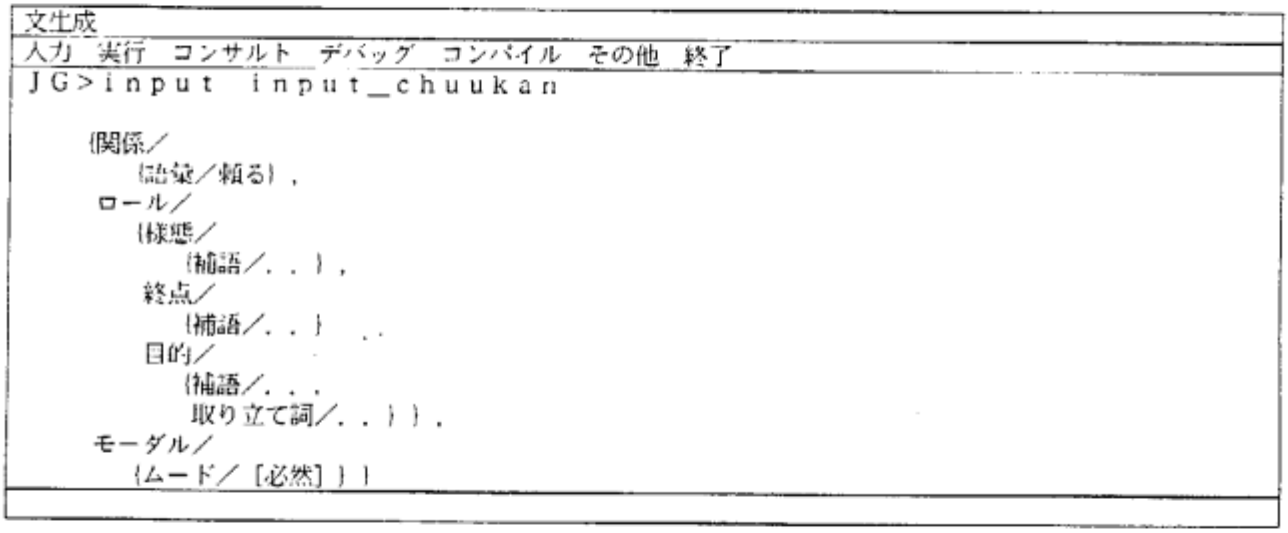


図 2.6: 中間表現プリティ・プリント

2.3.2 実行

データ・バッファにあるデータから、生成実行を行なう。 生成エンジンの中に、デバッグ・オンの状態のものが有れば、CIL デバッグ・ウインドウが表示され、実行のトレースが行える (図 2.7)。 また、全てのファイルがデバッ

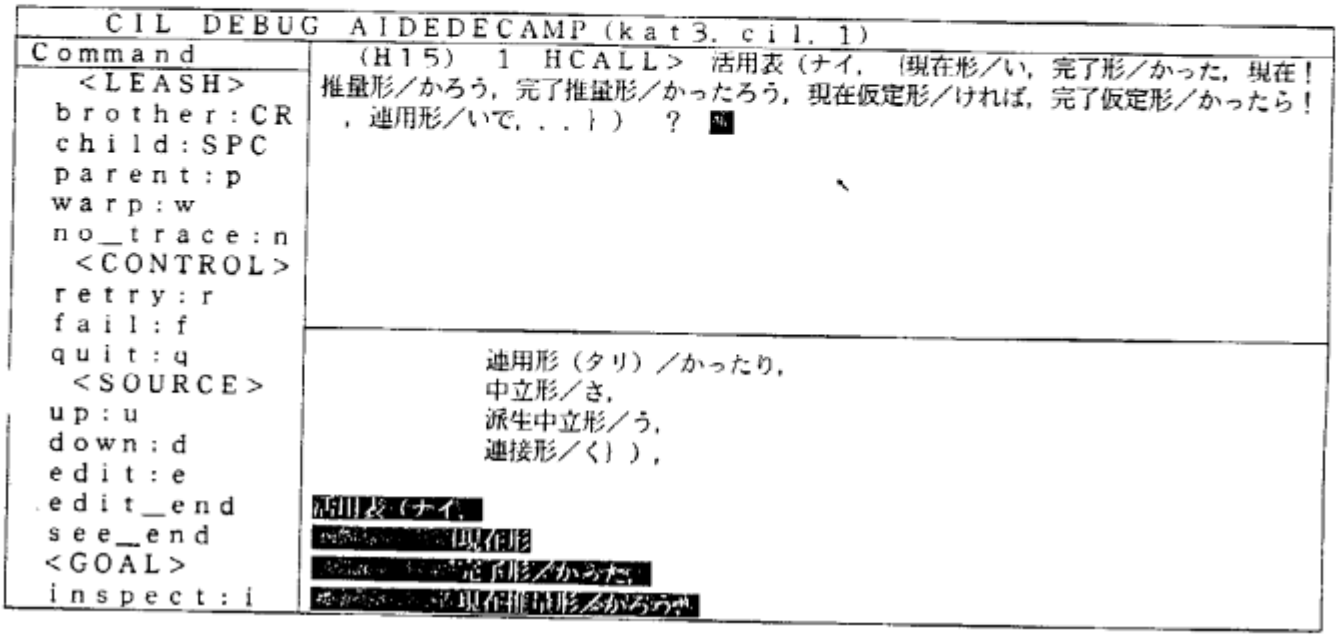


図 2.7: CIL デバッガ

グ・オフの状態であれば、単に生成結果が表示されるだけである。

1. トップ・ウインドウのメニューで、

"実行"

を選択すると、実行種別を項目とするサブ・メニューが表示される (図 2.8)。

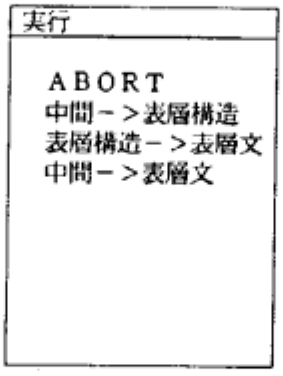


図 2.8: 実行種別

2. 以下に、サブ・メニューの各項目について説明を述べる。

- 中間表現⟹表層文
  中間表現データ・バッファにある中間表現データのマクロ展開を行い、更に、生成された表層構造データ

の形態素生成処理を行う。 形態素生成処理の結果得られる表層文は、表層文データのモード設定で指定された出力対象に出力される。 ただし、生成された表層構造データは、表層構造データ・バッファにはセットされない。

- 中間表現⟹表層構造
  中間表現データ・バッファにある中間表現データのマクロ展開を行う。 生成された表層構造は、表層構造データのモード設定で指定された出力対象に出力され、 更に、表層構造データ・バッファにセットされる。
- 表層構造⟹表層文
  表層構造データ・バッファにある表層構造データの形態素生成処理を行う。 生成された表層文は、表層文データのモード設定で指定された出力対象に出力される。

2.3.3 コンサルト

1. トップ・ウインドウのメニューで

"コンサルト"

を選択すると、生成エンジンの種別を項目とするサブ・メニューが表示される (図 2.9)。

図 2.9: 生成エンジンの種別

2. サブ・メニューの項目から、コンサルト・オン / オフの切り替えを行う種別 (マクロ展開 / 形態素生成 / 辞書) を選択すると、ファイル名を項目とするメニューが表示される。
既にコンサルト・オンされているファイルは、反転表示されている (図 2.10)。

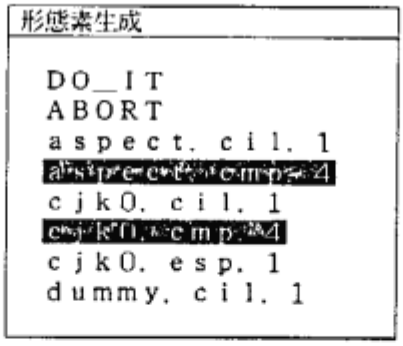


図 2.10: ファイル名メニュー

3. ファイル名を選択し、最後に

"DO_IT"

を選ぶ。反転表示された項目の選択は、コンサルト・オフの指定を意味する。

以上で、コンサルト・オン / オフが行われ、トップ・ウインドウに コンサルト・オン / オフのメッセージがファイル名付きで表示される (図 2.11)。

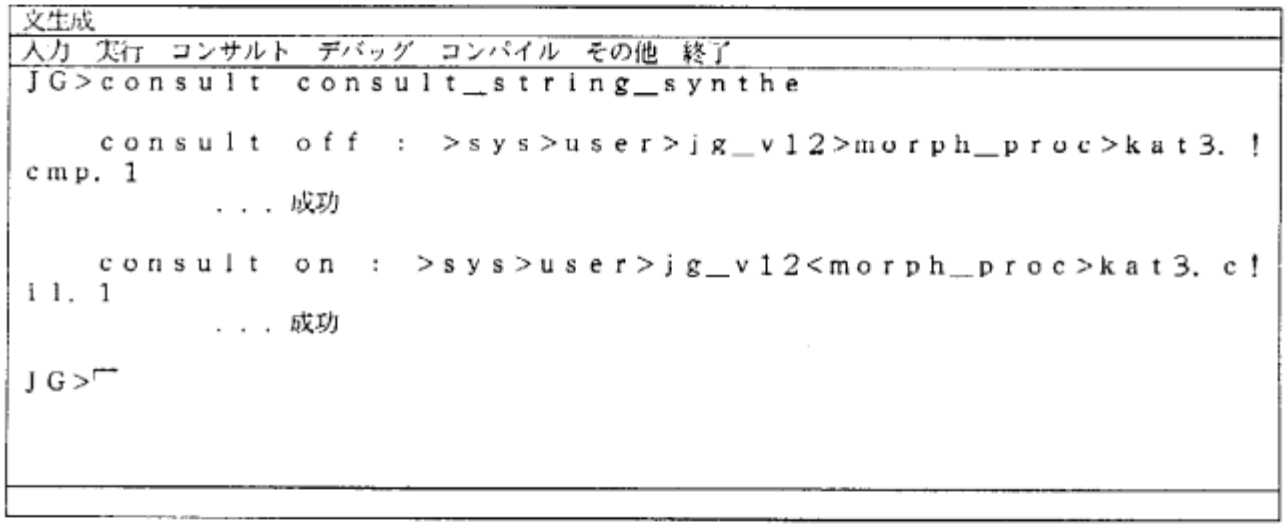


図 2.11: コンサルト・オン / オフのメッセージ

2.3.4 デバッグ

1. トップ・ウインドウのメニューで

"デバッグ"

を選択すると、生成エンジンの種別を項目とするサブ・メニューが表示される (図 2.12)。

図 2.12: 生成エンジンの種別

2. サブ・メニューの項目から、デバッグ・オン / オフの切り替えを行う種別 (マクロ展開 / 形態素生成処理) を選択すると、ファイル名を項目とするメニューが表示される。
既にデバッグ・オンされているファイルは、反転表示されている (図 2.13)。

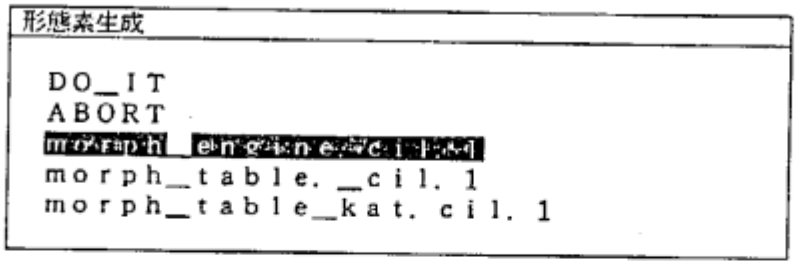


図 2.13: ファイル名メニュー

3. ファイル名を選択し、最後に

"DO_IT"

を選ぶ。 反転表示された項目の選択は、デバッグ・オフの指定を意味する。

以上の操作の後、デバッグ・オンされた各ファイルに対応してデバッグ・ウインドウが生成表示される (図 2.14)。

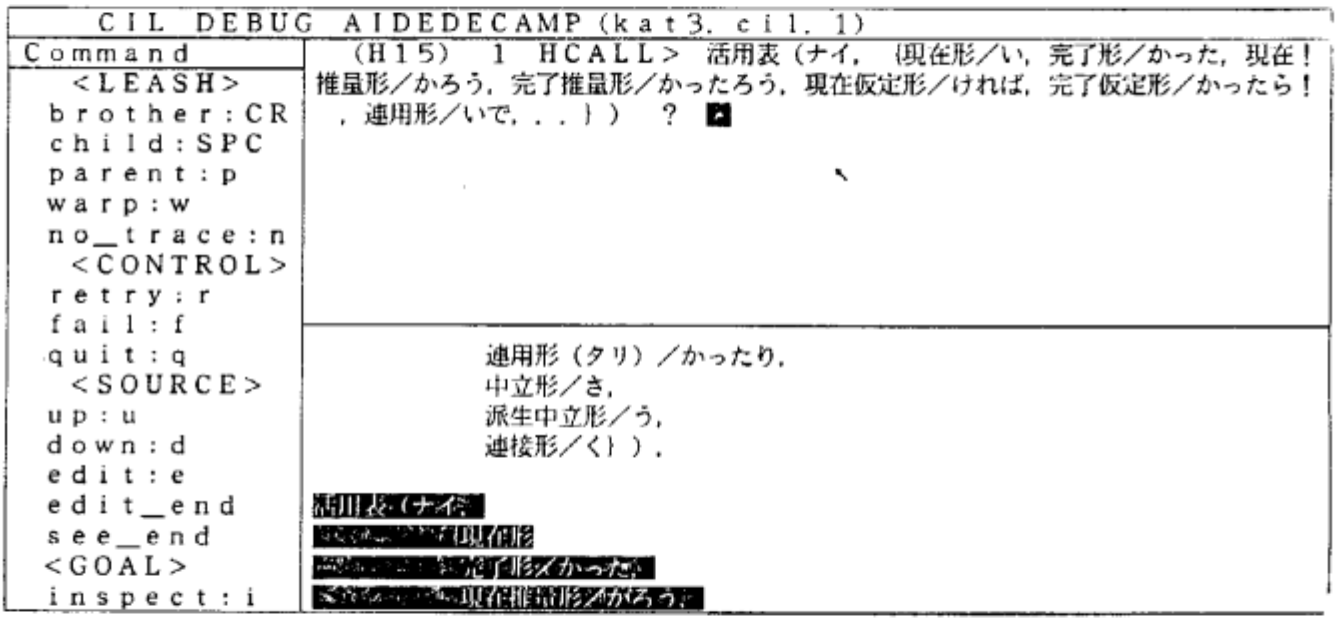


図 2.14: CIL デバッガ

2.3.5 コンパイル

1. トップ・ウインドウのメニューで

"コンパイル"

を選択すると、生成エンジンの種別を項目とするサブ・メニューが表示される (図 2.15)。

図 2.15: 生成エンジンの種別

2. サブ・メニューの項目から、コンパイルを行う種別 (マクロ展開 / 形態素生成処理 / 辞書) を選択すると、ファイル名を項目とするメニューが表示される (図 2.16)。

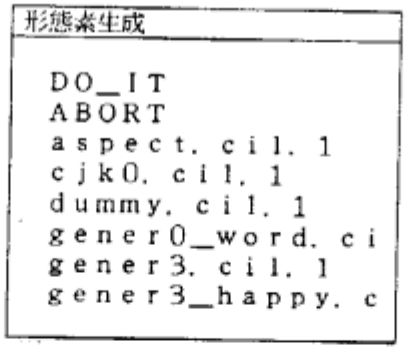


図 2.16: ファイル名メニュー

3. ファイル名を選択し、最後に

"DO_IT"

を選ぶ。

以上で、コンパイルが行われ、トップ・ウインドウに、コンパイルのメッセージがファイル名付きで表示される。ただし、対象ファイルがコンサルト・オンの状態の場合には、コンパイルの前にコンサルト・オフを行い、コンパイルの後にコンサルト・オンを行う (図 2.17)。

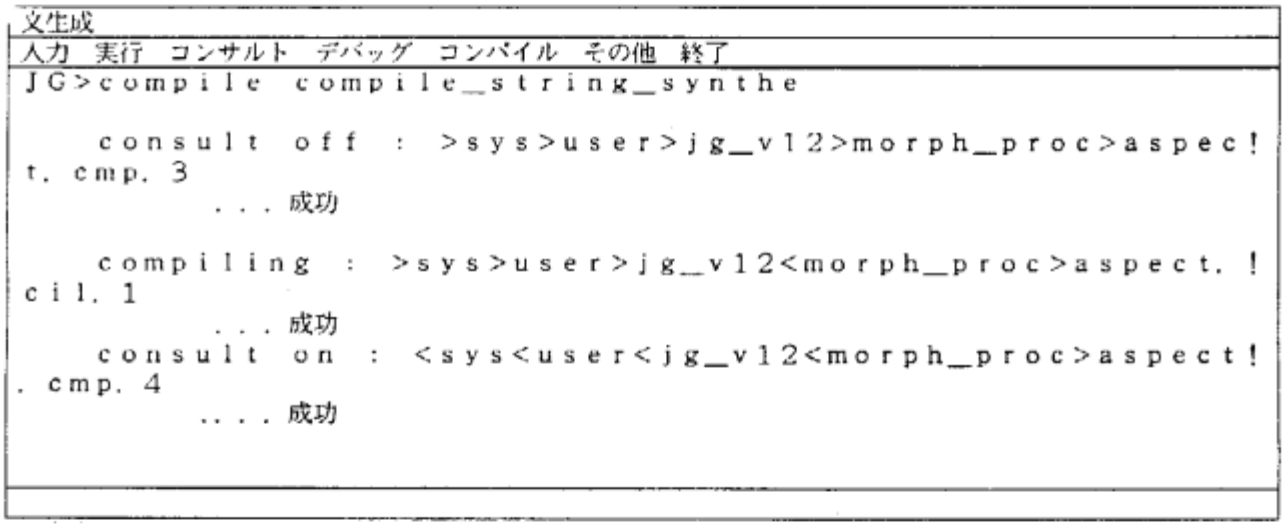


図 2.17: コンパイルのメッセージ

2.3.6 その他

“ その他 ” では、連続処理、モード設定、編集が行なえる。これらは、サブ・メニューでそれらに対応した項目を選択することにより、操作を開始できる (図 2.18)。 以下では、サブ・メニューを選択した後の操作の説明を行な

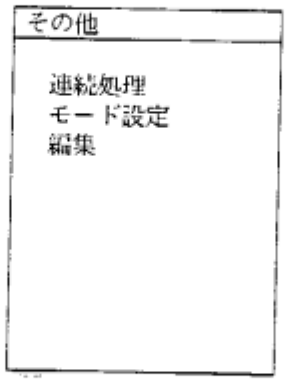


図 2.18: “ その他 ” のサブ・メニュー

う。

1. 連続処理

「連続処理」とは、複数個のデータを入力し、それを元に順次生成実行を行う処理のことである。

(a) サブ・メニューで、

“ 連続処理 ”

を選択すると、図 2.19のような連続処理の種別メニューが表示される。 ()

図 2.19: 連続処理の種別メニュー

(b) 連続処理の種別の選択後、ファイル名を項目とする入力データ・メニューが表示される (図 2.20)。

(c) 入力データ・メニューから複数個のファイルを選択し、

“DO_IT”

を行うと、順次生成実行が行われる。

2. モード設定の操作方法

(a) サブ・メニューで

“ モード設定 ”

を選ぶと、データ種別を項目とするメニューが表示される (図 2.21)。

(b) データ種別を選択すると、そのデータに固有のモード設定メニューが表示される。 モード設定の各項目に関しては、機能概要のモード設定を参照のこと (図 2.22, 2.23, 2.24)。

3. 編集

文生成デバッガでは、中間表現と表層構造の編集を行うことができる。

(a) データ編集開始まで

ここでは、中間表現と表層構造の各々に対して固有のウインドウを生成表示するまでの操作を述べる。

i. サブ・メニューで、

"編集"

を選択すると、中間表現、表層構造を項目とするデータ種別・メニューが表示される (図 2.25)。

ii. データ種別メニューで中間表現または表層構造を選択すると、編集用ウインドウ (図 2.26) が表示される。

(b) 中間表現データの編集

中間表現データの編集中は、編集用ウインドウ上のメニューで "end" が選択されるまで、このウインドウ上の操作 (編集用メニューの選択と、 pmacs の操作) のみが有効である (図 2.26)。

以下にメニューの各項目の機能を述べる。

i. from_buf

中間表現データバッファから中間表現データを持ってくる。中間表現データは、編集用ウインドウにプリティ・プリント表示される (図 2.27)。

ii. to_buf

テキスト・バッファ上の中間表現データを中間表現データ・バッファに書き込む。この操作を行わないと、中間表現データ・バッファの内容は書き変わらない。

iii. input

この項目を選択すると、入力データ選択用のファイル名メニューが表示される。このメニューでファイル名を選択すると、データが入力されて、編集用ウインドウ上にプリティ・プリント表示される。

iv. pp

テキスト・バッファ上の中間表現データのプリティ・プリント表示を行う。

v. check

テキスト・バッファ上の中間表現データの中間表現チェックを行う。中間表現にエラーがある場合、トップ・ウインドウにメッセージが表示される。 エラー表示には 2 通りあり、次に述べる "mode" によって設定される (図 2.28, 図 2.29)。

vi. mode

中間表現チェックのエラー表示のモードを設定する。 モード選択メニュー (図 2.30) で、 "yes" を選択すると、プリティ・プリントつきのエラー表示が行なわれ、 no を選択すると、単にエラー・メッセージだけが表示される。

vii. write

テキスト・バッファ上の中間表現データをファイルに書き込む。 この項目を選択すると、編集用ウインドウの下部に、エコー・エリア・ウインドウが表示される (図 2.31)。 ここに、ファイル名を書けば、そのファイル名で中間表現データがセーブされる。

viii. end

中間表現の編集を終了して、トップ・ウインドウに戻る。 この項目を選択すると、編集用ウインドウは消去される。

(c) 表層構造データの編集

表層構造データの編集中は、編集用ウインドウ (図 2.32) 上のメニューで、 "end" が選択されるまで、このウインドウ上の操作 (編集用メニューの選択と、 pmacs の操作) のみが有効である。

以下にメニューの各項目の機能を述べる。

i. from_buf
表層構造データ・バッファから表層構造データを持ってくる。 表層構造データは編集用ウインドウにプリティ・プリント表示される (図 2.33)。

ii. to_buf
テキスト・バッファ上の表層構造データを表層構造データ・バッファに書き込む。この操作を行わないと、表層構造データ・バッファの内容は書き変わらない。

iii. graphic
グラフィック表示用ウインドウが生成され、テキスト・バッファ上の表層構造データが二分木の形でグラフィック表示される (図 2.34)。
以後、グラフィック表示用ウインドウのメニューで、"EXIT" が選択されるまで、グラフィック表示用ウインドウ上のメニュー操作のみが有効となる。 以下に、グラフィック表示用ウインドウのメニューの各項目に関して述べる。

- up
  グラフィック表示された二分木の上方向へのスクロールを行う。
- down
  グラフィック表示された二分木の下方向へのスクロールを行う。
- left
  グラフィック表示された二分木の左方向へのスクロールを行う。
- right
  グラフィック表示された二分木の右方向へのスクロールを行う。
- EXIT
  グラフィック表示の終了。 グラフィック表示用ウインドウは消去され、表層構造編集ウインドウに戻る。

iv. end
表層構造の編集を終了して、トップ・ウインドウに戻る。 この項目を選択すると、編集用ウインドウは消去される。

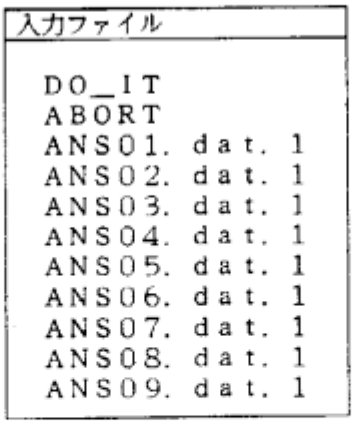


図 2.20: ファイル名メニュー

図 2.21: データ種別

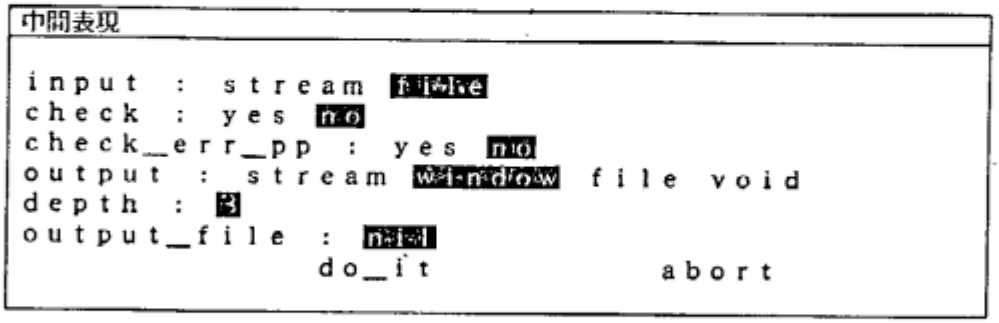


図 2.22: 中間表現データ・モード設定メニュー

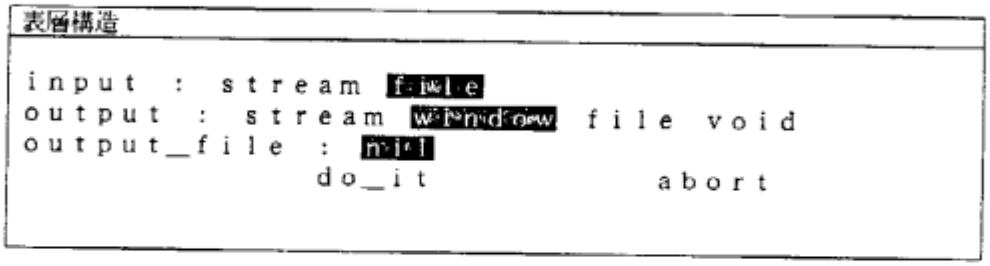


図 2.23: 表層構造データ・モード設定メニュー

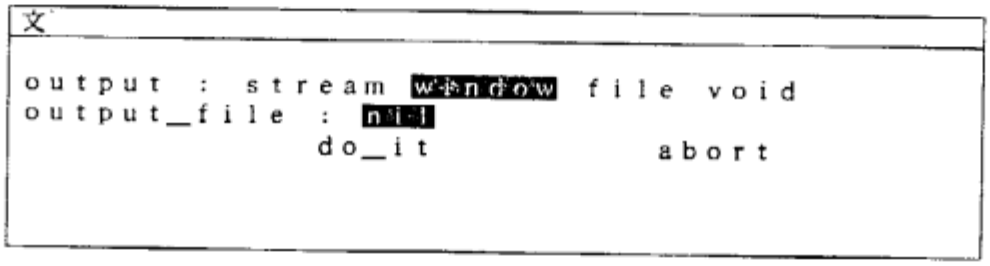


図 2.24: 表層文データ・モード設定メニュー

図 2.25: データ種別メニュー

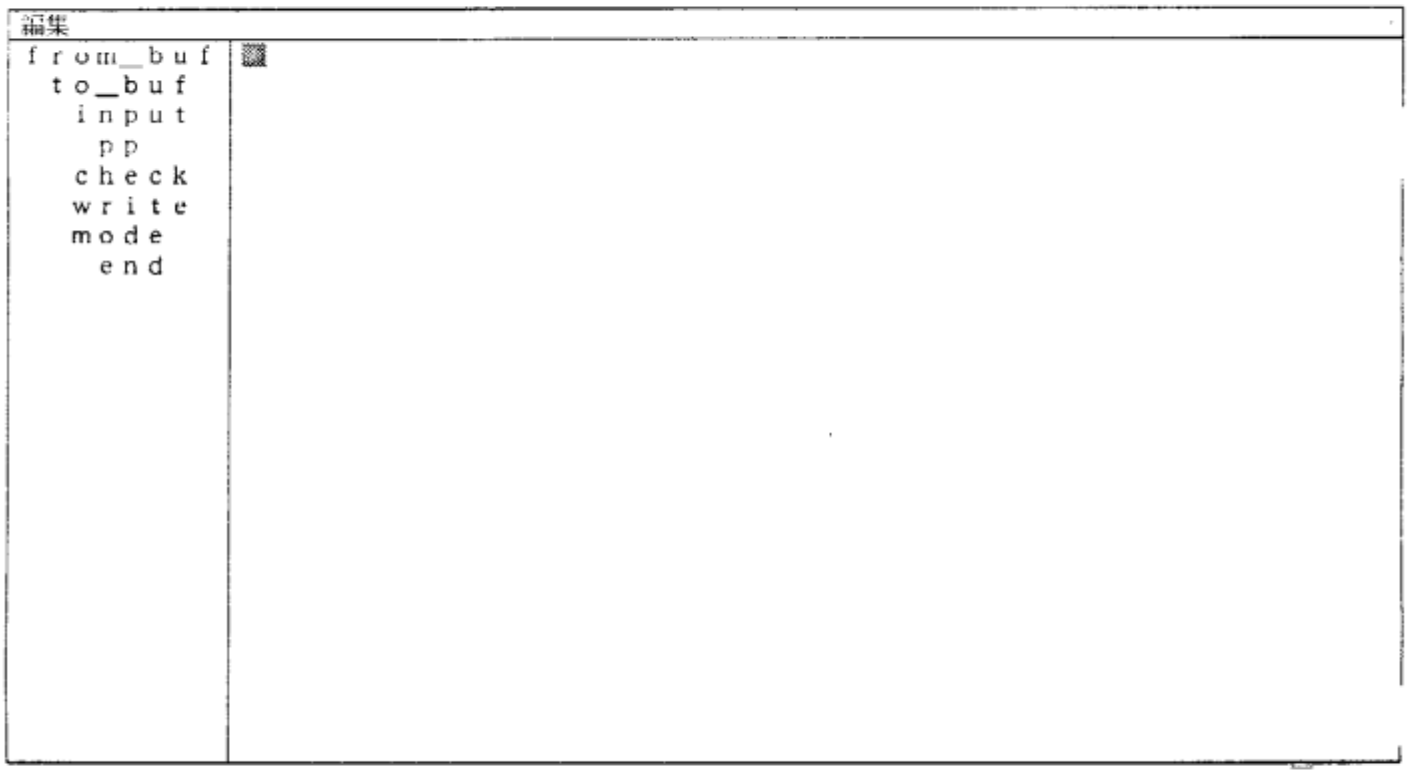


図 2.26: 中間表現編集ウインドウ

```
編集
from_buf    {関係／
  to_buf      {語彙／頼る}，
    input     ロール／
      pp        {様態／
    check           {補語／
    write               {語彙／どうしても}}，
    mode          終点／
      end           {補語／
                      {被修飾／
                        {語彙／恵み}，
                      連体修飾／
                        {語彙／自然}}}，
                目的／
                  {補語／
                    {関係／
                      {語彙／生きる}，
                    ロール／
                      {行為者／
                        {補語／
                          {語彙／人間}}，
                       場所／
                        {補語／
                          {被修飾／
                            {語彙／上}，
                          連体修飾／
```

図 2.27: 中間表現プリティ・プリント

```
文生成
入力  実行  コンサルト  デバッグ  コンパイル  その他  終了
      目的／
        {補語／
          {関係／
            {語彙／生きる}，
          ロール／
            {行為者／
              {補語／
                {
ERROR  :  辞書に登録されていません。
%%%%%  HERE  %%%%%
                  語彙／猿}}，
              場所／
                {補語／
                  {被修飾／
```

図 2.28: プリティ・プリントつきのエラー表示

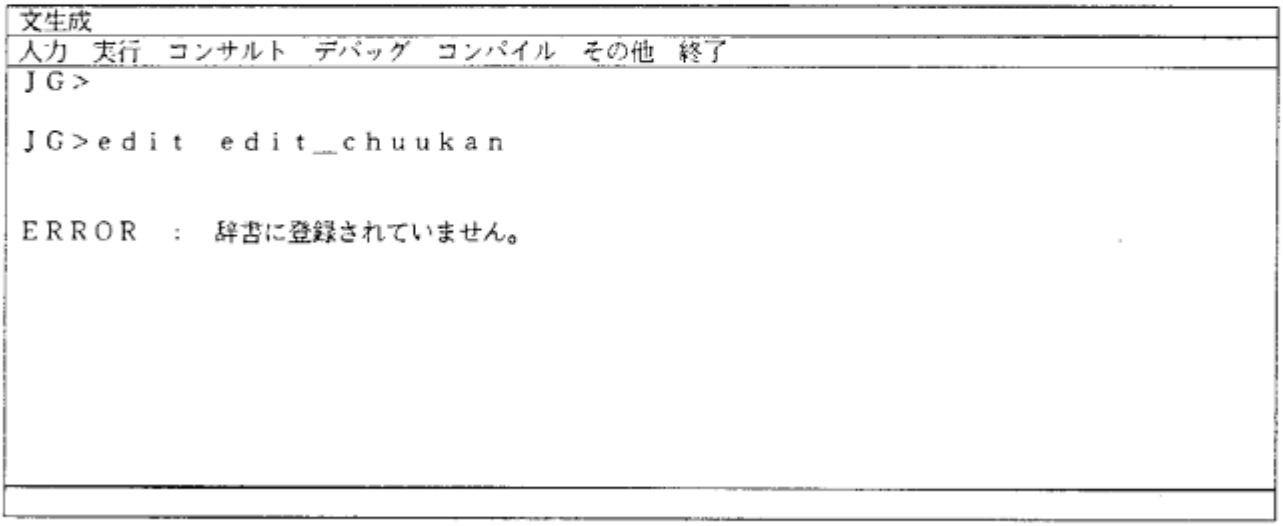


図 2.29: 中間表現チェックのエラー表示

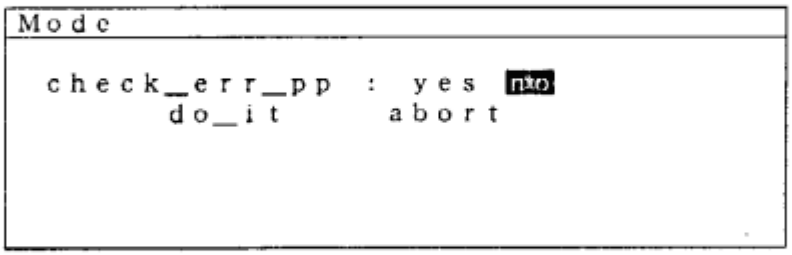


図 2.30: モード選択メニュー

編集

from_buf
to_buf
input
pp
check
write
mode
end

```
                    {語彙／恵み} ,
                連体修飾／
                    {語彙／自然} } } ,
            目的／
                {補語／
                    {関係子／
                        {語彙／生きる} ,
                    ロール／
                        {行為者／
                            {補語／
                                {語彙／ｓａｒｕ} } ,
                        場所／
                            {補語／
                                {被修飾／
                                    {語彙／上} ,
                                連体修飾／
                                    {被修飾／
                                        {語彙／地球} ,
                                    連体修飾／
                                        {語彙／この} } } } } ,
                    アスペクト／［つづける，ていく］} ,
                取り立て詞／は} } ,
```

(>ｓｙｓ>ｕｓｅｒ>ｊｇ_ｖ１２>ｍａｃｒｏ_ｅｘｐｒ)

図 2.31: ファイルへの書き込み

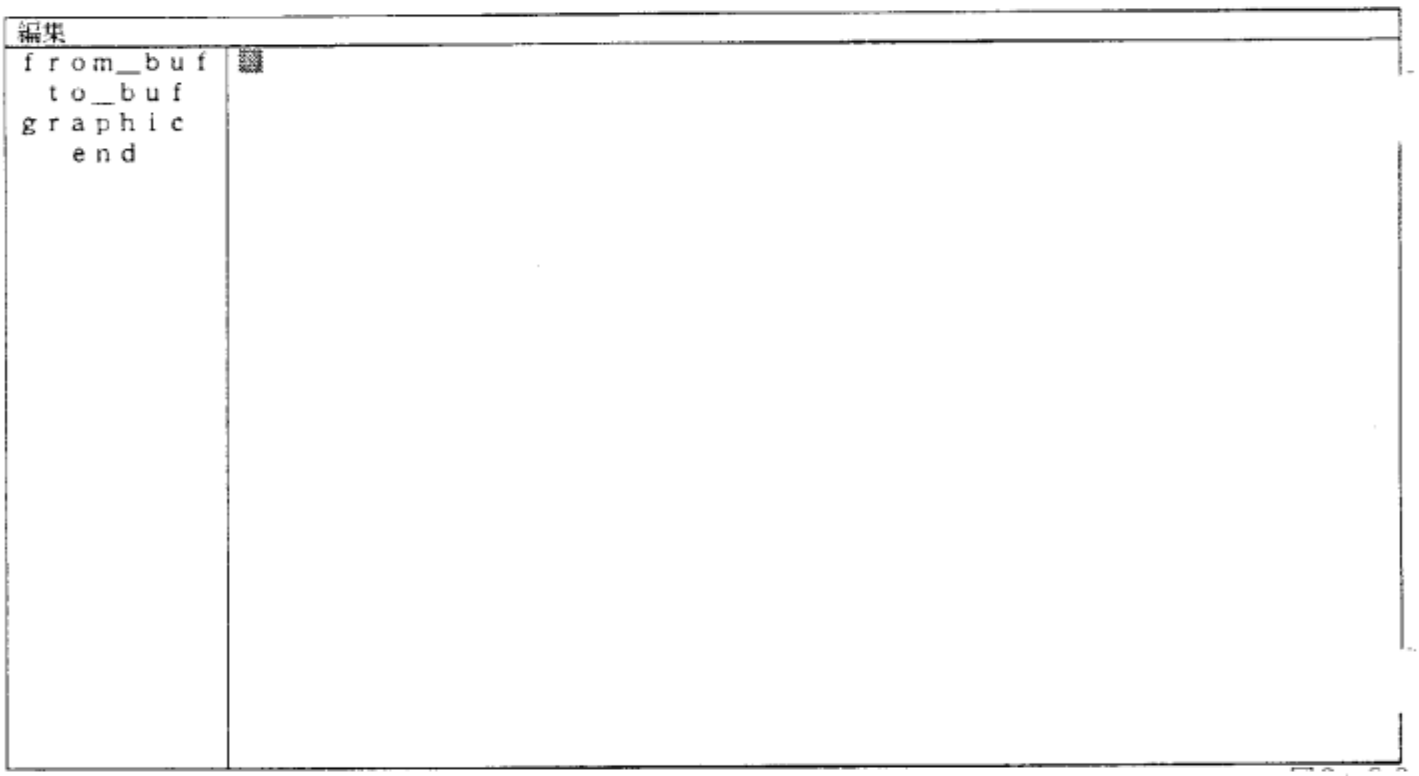


図 2.32: 表層構造編集ウインドウ

編集

from_buf
to_buf
graphic
end

{h／｛[symbol，。]｝，
c／{h／｛[助動詞，{語彙／なければならない，語幹／なければならな，活用／ナイ，接続／[！[ダ系，連用形]，[A，連接形]]，読み／なければならない}]｝，
c／{h／｛h／｛h／｛h／{}動詞，{語彙／頼る，派生／[たより，便り]，受動／{対象／！[：：：（対象，が），：：：（行為者，に）]｝，語幹／頼，活用／ウ系強変化ラ行，読み／たよる！，格／[：：：（行為者，が），：：：（対象，に）]，アスペクト分類／[継続]｝]｝｝，
c／{h／｛[格助詞，{語彙／に，読み／に}]｝，
c／{h／｛[名詞，{語彙／恵み，読み／めぐみ，類別／[普通！名詞]，意味分類／A}]｝]｝，
c／{h／｛[ind，連帯（の）]｝，
c／{h／｛[準用詞，{語彙／だ，語幹／´ ´，活！用／ダ系，読み／だ，格／[：：：（記述対象，は），：：：（内容，´ ´）]｝]｝｝，
c／｛{名詞，{number／A，語彙／自！然，読み／しぜん，類別／普通，意味／B，gender／C，sosei／D，person／3}！]｝｝｝｝｝｝，
c／｛[副詞，{語彙／どうしても，読み／どうしても，類別／A}]｝｝，
c／{h／｛[取り立て詞，{語彙／は，接続／[[ダ系，連用形]，[A，中立形]！]，読み／は}]｝，
c／{h／{h／｛[格助詞，{語彙／に，読み／に}]｝，
c／｛[名詞，{語彙／ため，読み／ため，類別／連用名（格助詞！)}]]｝｝，
c／{h／{h／{h／｛[助動詞，{語彙／いく，語幹／い，活用／ウ！系強変化イク，接続／連用形，読み／いく，アスペクト分類／継続}]｝，
c／{h／｛[助辞，て]｝，

図 2.33: 表層構造のプリティ・プリント

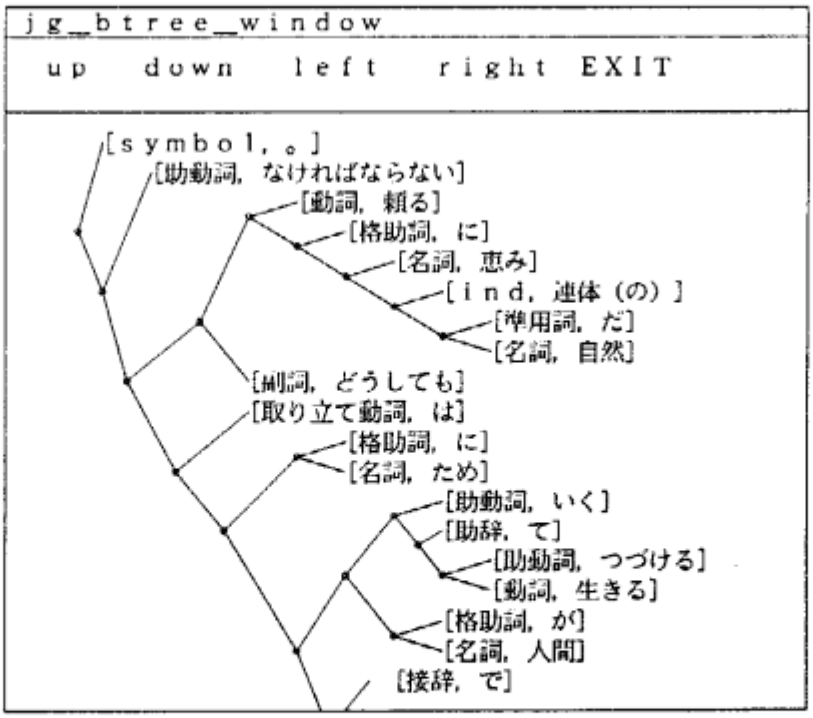


図 2.34: 表層構造のグラフィック表示

2.3.7 終了

トップ・ウインドウのメニューで、

“終了”

を選ぶと、文生成デバッガは消去され、SIMPOS デバッガに戻る。

2.4 メニュー項目一覧

文生成ツールのメニューの項目の一覧を記述する。記述の順序は、メニュー内の順序に対応する。

2.4.1 文生成デバッガ・トップ・ウインドウ

1. 入力

(a) ABORT

機能 トップ・ウインドウに戻る。

(b) 中間表現入力

機能 中間表現データの入力を行なう。

関連参照ページ 21

(c) 表層構造入力

機能 表層構造データの入力を行なう。

関連参照ページ 21

2. 実行

(a) ABORT

機能 トップ・ウインドウに戻る。

(b) 中間⟹表層構造

機能 マクロ展開を行なって中間表現から表層構造を生成する。

関連参照ページ 24

(c) 表層構造⟹表層文

機能 形態素生成を行なって表層構造から表層文を生成する。

関連参照ページ 24

(d) 中間⟹表層文

機能 マクロ展開と形態素生成を行なって中間表現から表層文を生成する。

関連参照ページ 23

3. コンサルト

(a) ABORT

機能 トップ・ウインドウに戻る。

(b) マクロ展開

機能 マクロ展開規則のコンサルトを行なう。

関連参照ページ 25

(c) 辞書

機能 辞書ファイルのコンサルトを行なう。

関連参照ページ 25

(d) 形態素生成

機能 形態素生成規則のコンサルトを行なう。

関連参照ページ 25

4. デバッグ

(a) ABORT

機能 トップ・ウインドウに戻る。

(b) マクロ展開

機能 マクロ展開規則のデバッグ・オン / オフを行なう。

関連参照ページ 27

(c) 形態素生成

機能 形態素生成規則のデバッグ・オン / オフを行なう。

関連参照ページ 27

5. コンパイル

(a) ABORT

機能 トップ・ウインドウに戻る。

(b) 辞書

機能 辞書ファイルのコンパイルを行なう。

関連参照ページ 29

(c) マクロ展開

機能 マクロ展開規則のコンパイルを行なう。

関連参照ページ 29

(d) 形態素生成

機能 形態素生成規則のコンパイルを行なう。

関連参照ページ 29

6. その他

(a) 連続処理

i. ABORT

機能 トップ・ウインドウに戻る。

ii. 中間⟹表層文

機能 複数個の中間表現データに対して、 マクロ展開と形態素生成を行なって、順次、表層文を生成する。

関連参照ページ 31

iii. 中間⟹表層構造

機能 複数個の中間表現データに対して、 マクロ展開を行なって、 順次、表層構造を生成する。

関連参照ページ 31

iv. 表層構造⟹表層文

機能 複数個の表層構造データに対して、形態素生成を行なって、順次、表層文を生成する。

関連参照ページ 31

(b) モード設定

i. ABORT

機能 トップ・ウインドウに戻る。

ii. 中間表現

機能 中間表現データ操作部のモード設定を行なう。

関連参照ページ 31

iii. 表層構造

機能 表層構造データ操作部のモード設定を行なう。

関連参照ページ 31

iv. 表層文

機能 表層文データ操作部のモード設定を行なう。

関連参照ページ 31

(c) 編集

i. ABORT

機能 トップ・ウインドウに戻る。

ii. 中間表現

機能 中間表現の編集を行なう。

関連参照ページ 32

iii. 表層構造

機能 表層構造の編集を行なう。

関連参照ページ 32

7. 終了

機能 文生成デバッガを終了する。

関連参照ページ 42

2.4.2 中間表現編集ウインドウ

1. from_buf

機能 中間表現データ・バッファから中間表現データをロードする。

関連参照ページ 32

2. to_buf

機能 中間表現データ・バッファに中間表現データをセーブする。

関連参照ページ 32

3. input

機能 中間表現データを入力する。

関連参照ページ 32

4. pp

機能 中間表現データのプリティ・プリントを行なう。

関連参照ページ 32

5. check

機能 中間表現チェックを行なう。

関連参照ページ 32

6. write

機能 中間表現データをファイルへ書き込む。

関連参照ページ 32

7. mode

機能 中間表現チェックのモード設定を行なう。

関連参照ページ 32

8. end

機能 中間表現の編集を終了する。

関連参照ページ 32

2.4.3 表層構造編集ウインドウ

1. from_buf

   機能 表層構造データ・バッファから表層構造データをロードする。

   関連参照ページ 33

2. to_buf

   機能 表層構造データ・バッファに表層構造データをセーブする。

   関連参照ページ 33

3. graphic

   機能 表層構造の木構造のグラフィック表示を行なう。

   関連参照ページ 33

4. end

   機能 表層構造の編集を終了する。

   関連参照ページ 33

2.4.4 グラフィック表示ウインドウ

1. up

   機能 上方向へのスクロールを行なう。

   関連参照ページ 33

2. down

   機能 下方向へのスクロールを行なう。

   関連参照ページ 33

3. left

   機能 左方向へのスクロールを行なう。

   関連参照ページ 33

4. right

   機能 右方向へのスクロールを行なう。

   関連参照ページ 33

5. EXIT

   機能 グラフィック表示を終了する。

   関連参照ページ 33

2.5 出力メッセージ

1. `compiling : <ファイル名> ...` 成功

   メッセージの意味 ファイル名で指示されたファイルのコンパイルの成功。

   ユーザの対応 なし。

   関連参照ページ 29

2. `compiling : <ファイル名> ...` 失敗

   メッセージの意味 ファイル名で指示されたファイルのコンパイルの失敗。

   ユーザの対応 なし。

   関連参照ページ 29

3. `consult off : <ファイル名> ...` 成功

   メッセージの意味 ファイル名で指示されたファイルのコンサルト・オフの成功。

   ユーザの対応 なし。

   関連参照ページ 25

4. `consult off : <ファイル名> ...` 失敗

   メッセージの意味 ファイル名で指示されたファイルのコンサルト・オフの失敗。

   ユーザの対応 なし。

   関連参照ページ 25

5. `consult on : <ファイル名> ...` 成功

   メッセージの意味 ファイル名で指示されたファイルのコンサルト・オンの成功。

   ユーザの対応 なし。

   関連参照ページ 25

6. `consult on : <ファイル名> ...` 失敗

   メッセージの意味 ファイル名で指示されたファイルのコンサルト・オンの失敗。

   ユーザの対応 なし。

   関連参照ページ 25

7. `initializing ...`

   メッセージの意味 初期化の実行中。

   ユーザの対応 なし。

   関連参照ページ 8

8. マクロ展開 : 失敗

   メッセージの意味 マクロ展開の実行中、失敗 (fail) した。

   ユーザの対応 なし。

   関連参照ページ 23

9. 形態素生成 : 失敗

   メッセージの意味 形態素生成の実行中、失敗 (fail) した。

   ユーザの対応 なし。

   関連参照ページ 23

10. 構文チェック成功

メッセージの意味 中間表現チェック中に、エラーが発見されなかった。

ユーザの対応 なし。

関連参照ページ 18

第3章

文生成のための規則

文生成機構のうち文生成用規則類はユーザに開放されていて、再調整したり、別の規則と入れ替えたりすることができる。また、生成用辞書についても必要なエントリを加えたり、種々の情報を再調整することができる。ここでは、マクロ展開規則、生成用辞書、形態素生成用テーブル類の作成手順について説明する。

3.1 マクロ展開規則の作成手順

マクロ展開部では文生成機構の入力である生成用中間表現を入力とし、形態素生成に必要な語彙データの集まりである表層構造 を出力とする。入力の生成用中間表現を基本表現に展開し、同時に表層構造を生成して、表層構造を形態素生成部へ渡す。

入力である生成用中間表現は CIL の部分項で書かれていて、生成すべき日本語表層文の構文情報がすべて含まれている。

生成用中間表現の内で、表層構造の生成に用いる構文情報を二分木の形で明示したものを、基本表現と呼び、より簡潔なマクロを用いて書いたものをマクロ表記された中間表現と呼ぶ。現在、標準的なマクロが定義されている。

基本表現は次の形式に従って書かれる。

```
<文>::=
        {head/<文>,comp/<文>}                    |
        {cont/<文>,symbol/<記号>}                |
        {lex/<表層表現>}
<記号>::=
        {type/<TYPE>,right/<SYMBOL>,left/<SYMBOL>}
<TYPE>::=
        right |left |right_left
```

句読点や引用符などの記号の処理を行うために、cont 及び symbol という属性を用意した。cont 属性の値として与えられた文に symbol 属性 の値として与えられた記号が付加される。

図 3.1、に表層文「人間は自然の恵みに頼らなければならない。」に対応する基本表現を示す。

```
{head/{head/{head/{lex/ なければならない},
             comp/{lex/ 頼る}},
       comp/{head/{lex/ に},
             comp/{head/{lex/ 恵み},
                   comp/{head/{lex/ の},
                         comp/{lex/ 自然}}}}},
comp/{symbol/{type/left,left/"、 "},
       cont/{head/{lex/ は},
             comp/{lex/ 人間}}}}
```

図 3.1: 基本表現による記述例

基本表現は、生成すべき文の構文情報を非常に厳密に記述する表現である。

次に標準マクロ定義について説明する。標準マクロ定義では、文の構成をおおまかに次のように考える。

文＝ロール群＋関係＋認め方＋時制＋アスペクト＋モダリティ

例として、「人間は自然に頼らなければならない」を考えてみる。この文は「頼る」という関係とそのロール「人間は」「自然に」と必然のムードを表す助動詞「なければならない」から成っている。ロールとは、関係に対して「行為者」「対象」などの役割を果たすものである。ロールと関係を合わせて文核と呼ぶ。文核に付加する成分には、「ている」などアスペクトを表すもの、「なければならない」などムードを表すもの、疑問の「か」など態度を表すもの、過去を表す「た」などがある。これらの構成要素を関係属性、ロール属性、時制属性のように属性名とその値で明示する。

次に、名詞句を考える。標準マクロ定義では名詞句は、

名詞句＝連体修飾句＋被修飾句｜語

であるとする。連体修飾関係はなくてもよく、その場合は語が名詞句を構成する。文と同様に連体修飾属性、被修飾属性でそれぞれの構成要素を表記する。

さらに、文、節、句、を構成する最小のものとして語がある。これは、語彙という属性名で明示され、

$$\text{語} = \text{数} + \text{語彙} + \text{派生}$$

であるとする。数は「3人の学生」のような表現を表すのに用いる。派生は「豊か」という名詞を「豊かにする」と派生させる場合の「にする」にあたる。

以上のことから、標準マクロ表記された中間表現は次の3つのタイプに分類される。

type1	{関係 /.., ロール /....}	関係とロールなどを持つ。
type2	{被修飾 /.., 連体修飾 /....}	被修飾と連体修飾などを持つ。
type3	{語彙 /....}	語彙などを持つ。

大まかに言って、type1は文に、type2は名詞節に、type3は語に相当する。
主にtype1の中間表現が持つ属性のうち主なものを示す。

1. 関係属性
   関係属性の値は、通常type1またはtype3の中間表現である。例えば「自然に頼る」の場合、関係の値は{語彙 / 頼る}である。これ以外に特殊な関係として並立、分配、特殊がある。

2. 関係子属性
   関係子属性は分配関係、並立関係、特殊関係で用いる。属性値として「と」「たり」「名詞連接」などをとる。「名詞連接」を除いてロール成分に付加される。

3. ロール属性
   ロール属性の値は幾つかのロールの集合であり、「行為者」「対象」などのロール名を属性名とし、そのロールのロール成分を属性値とする。ロール成分 は補語属性、数量化属性、取り立て詞属性、記号属性、表層属性からなる。補語属性にはそのロール成分の補語の部分を記述し、その他の属性には補語に付加する表層表現を記述する。表層属性がある時は、その値を接辞として補語属性の表層表現に付加する。補語属性の値は文であるから、 type1、type2、type3いずれでもよい。例えば、「自然にだけ頼る」の対象ロール「自然にだけ」は次のように書くことができる。

   {対象 /{補語 /{語彙 / 自然}, 取り立て詞 / だけ}}

4. 数量化属性
   数量化属性は、ロールの中でしか使われない。「学生がみな」や「学生が3人」のようにロール成分の後ろに付加して数量を限定する働きをする語を記述する。数量化属性の値は「みな」など表層表現をそのまま与えるか、「3人」のような数を表す部分項を与える。「3人」の表記の仕方は数属性の値と同じものである。例えば、行為者ロール「学生が3人」は次のように書くことができる。

   {行為者 /{補語 /{語彙 / 学生}}, 数量化 /{数 /3, 助数詞 / 人}}

5. 取り立て詞属性
   後ろに付加する取り立て詞を与える。「だけ」「も」など表層表現を直接与える。

6. 記号属性
   付加する記号名を「読点」、「句点」のように直接与える。また、表層表現をそのまま与えてもよい。

7. モーダル属性
   モーダル属性の値はモダリティに関する接辞をマクロ表記した部分項であり、ムード属性、態度属性、認め方属性、時制属性、丁寧属性からなる。認め方と時制はそれぞれモダリティに関する認め方および時制である。ムードおよび態度は話し手が自分の態度を相手に示そうとする部分である。このうち客観的な記述に関する態度(対事的様相)をムード属性で表し、話相手に対する態度(対人的様相)を態度属性で表す。例えば「なければ

なりませんか」という表現を考えると、これは「なければならない」という必然のムードを丁寧表現した助動詞と「か」という疑問の態度を表す終助詞からなる。これを次のように表記する。

{ムード/[必然], 態度/[疑問], 丁寧/yes}

ムード、態度の値は「なければならない」「か」のように直接表層表現を与えることもできる。「てもよいのだろうか」のような複合モダリティはリストを用いて記述することができる。ただし、与えられたリストの順に生成される。

8. 時制属性
「現在」、または「過去」を値とする。具体的には助辞の「た」を最後に付加するかを決める。文核に対する時制とムードや態度に対する時制とを区別することに注意しなければならない。「のだ」というムードを表す助動詞は過去を表すために助辞「た」をとることができる。したがって、「頼るのだった」と「頼ったのだ」は区別されなければならない。

9. 認め方属性
「肯定」、または「否定」を値とする。具体的には助動詞の「ない」または「ぬ」を最後に付加するかを決める。時制属性と同様に文核に対する認め方とモダリティに対する認め方を区別しなければならない。

10. 語順属性
語順を指定したい時に用いる。語順属性の値はロール名のリストであり、そのリストの順に表層文が生成される。語順属性による語順の指定のない時はマクロ展開規則の中の語順決定規則により語順が決定される。語順は構造化されたマクロ表記によっても指定できる。例えば「人間が自然に頼る」を次のように記述した場合、「人間が」と「自然に」の語順は陽に決定されている。

```
{関係/{関係/{語彙/頼る},
        ロール/{対象/{補語/{語彙/自然}}}},
  ロール/{行為者/{補語/{語彙/人間}}}}
```

11. アスペクト属性
「ている」「かける」などのアスペクト辞を指定する。「〜しかけている」のような複合アスペクトはリストを用いて記述することができる。ただし、与えられたリストの順に生成される。

12. ヴォイス属性
「受動」、「使役」、「自発」、「可能」、「もらう」、「くれる」、「あげる」を値とする。それぞれ態変換を行う。間接受動主体、使役、授受など態変化に伴ってロールを必要とする場合は、受動({語彙/私}) のように記述する。

13. 派生属性
派生辞を指定する。派生辞には「する」「なる」「である」「だ」などがある。例えば「豊か」という名詞を「豊かにする」と派生させることができる。さらに、「名詞化」を値とすれば、文核全体を名詞へ派生させることができる。

次に、主に type2 が持つ属性について説明する。

1. 連体修飾属性
連体修飾句を記述するのに用いる。連体修飾属性の値は文であり、type1、type2、type3 のいずれでもよい。

2. 被修飾属性
連体修飾句のかかり先を示す。被修飾の値は名詞節であり、type2、type3 が与えられる。

3. 数属性
名詞節の数を指定する。例えば「3 人の学生」や「学生たち」を表現するのに用いる。数属性の値は単、複、あるいは 2、3 などの数字である。複数が指定された時は、辞書を参照して「たち」「々」などを付加する。数字が与えられた場合は、辞書を参照して助数詞を取り出し、「3 人の」という形にして名詞節の前に付加する。助数詞は助数詞属性を用いて指定することができる。例えば「一回の実行」は次のように記述できる。

{語彙 / 実行, 数 / 一, 助数詞 / 回}

この他に、取り立て詞属性、記号属性、派生属性、表層属性を持つことができる。

type3 は語を表現するのに用いる。語彙属性を用いて、辞書の見出しを指定する。他に、数属性、記号属性、派生属性、表層属性を持つことができる。

以上が標準マクロ定義の概要である。

マクロ展開部ではマクロ展開を行うと同時に表層構造を生成しなければならない。表層構造とは、形態素生成に必要な語彙データのリストである。語彙データとは品詞と辞書項目の組を要素とするリストである。「自然」の語彙データを示す。

[名詞,{読み / しぜん, 語彙 / 自然, 類別 /[普通名詞, 連用名 (単)]}]

辞書に存在しない語、例えば助辞の「の」の場合は、[助辞, の] のようになる。この、語彙データの集まりが表層構造である。表層構造もまた、二分木の形をしている。「自然を頼る」に対応する表層構造を示す。

```
[[[名詞,{読み / シゼン, 語彙 / 自然, 類別 /[普通名詞, 連用名 (単)]}],
[格助詞,{読み / ニ, 語彙 / に}]],
[動詞,{読み / タヨル, 語彙 / 頼る, 語幹 / 頼, 活用 / ウ系強変化ラ行,
        格 /[行為者:: が, 対象:: に], 受動 /{対象 /[対象:: が, 行為者:: に]},
        アスペクト分類 /[継続], 派生 /[たより, 頼り]}]]
```

マクロ展開部への入力がマクロ表記されている場合はマクロ展開規則を用いてマクロ展開を行う。そのために、マクロ展開エンジンからマクロ展開規則を参照する時のインタフェースを取るための述語の述語名が定められている。標準マクロ定義に対応する述語が定義されているが、ユーザはこれらの述語を再定義、変更、追加することによってマクロ展開規則の調整を行うことができる。

マクロ展開エンジンでは、マクロの展開を行うと同時に表層構造の生成 を行う。表層構造を生成するための述語が幾つか用意されていて、ユーザはマクロ展開規則の中からこれらの述語を利用することができる。同様に、辞書引きを行うための述語が用意されていて、マクロ展開規則の中から利用することができる。

さらに、マクロ展開エンジンではマクロ展開規則の記述を容易にするために 3 つの属性、'*pos'(part of speach) と '*dterm'(dictionary term) と '*case' を用意している。'*pos' 属性の値はその属性を持つ中間表現の主辞の品詞を表し、'*dterm' 属性の値はその中間表現の主辞の辞書項目を表す。'*case' 属性の値はその中間表現の主辞の格情報を表す。名詞句など中間表現の主辞が格情報を持たない場合は、'*case' 属性を与えない。'*pos' と '*dterm' の 2 つの属性は辞書引きを行う際に自動的に付加される。また、'*case' 属性は格情報を参照するための述語 jg_get_case/2 によって付加される。マクロ展開エンジンの処理内容を図 3.2に示す。

マクロ展開規則の実行は、jg_macro_expand/2 という述語による。以下で、標準マクロ展開規則を例にマクロ展開規則の作成手順を説明する。

例としてあげる述語の定義は正確なものではないので、より詳しくはソースを直接参照されたい。また、比較的ユーザによる調整を行い易い述語については付録にまとめてある。

jg_macro_expand(X1,X2)

マクロ表記された生成用中間表現が入力された場合、マクロ展開エンジンは jg_macro_expand/2 を実行する。2 つの引数のうち入力 X1 がマクロ表記された中間表現、出力 X2 が表層構造を含んだ基本表現である。標準マクロ展開規則の jg_macro_expand/2 を示す。

```
jg_macro_expand(X1,X19):-
    jg_goi(X1,X2),              語彙属性の展開
    jg_kankei(X2,X3),           関係属性の展開
    jg_hishuushoku(X3,X4),      被修飾属性の展開
    jg_hasei(X4,X5),            派生属性の展開
    jg_rentai(X5,X6),           連体修飾属性の展開
    jg_voice(X6,X7),            ヴォイス属性の展開
```

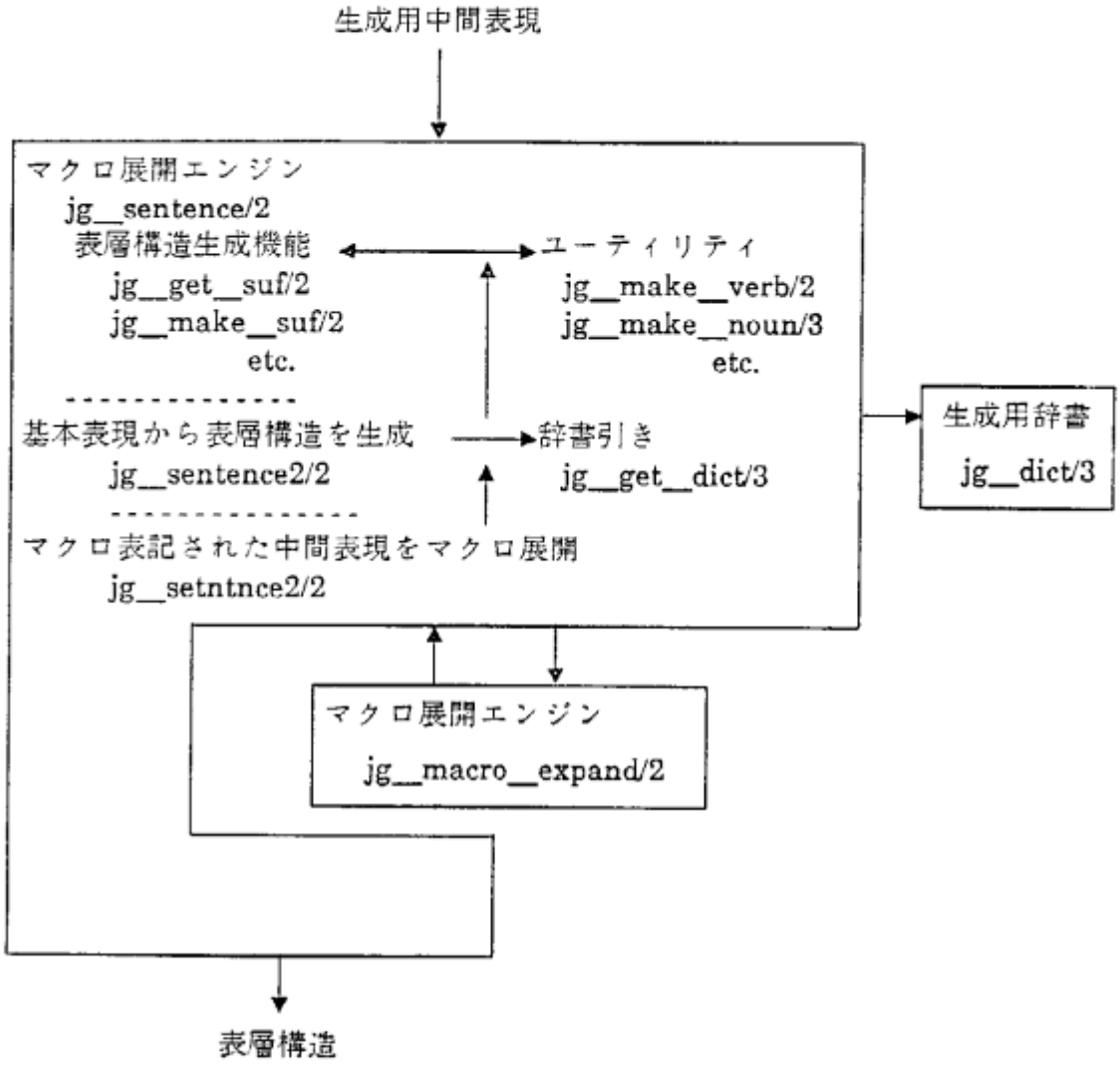


図 3.2: マクロ展開エンジンの処理内容

jg_aspect(X7,X8),	アスペクト属性の展開
jg_roles(X8,X9),	ロール属性の展開
jg_gojun(X9,X10),	語順の決定
jg_mitomekata(X10,X11),	認め方属性の展開
jg_toritate(X11,X12),	取り立て詞属性の展開
jg_meishika(X12,X13),	名詞化
jg_jisei(X13,X14),	時制属性の展開
jg_modal(X14,X15),	モーダル属性の展開
jg_polite(X15,X16),	丁寧属性の展開
jg_kigou(X16,X17),	記号属性の展開
jg_setsuzokushi(X17,X18),	接続詞属性の展開
jg_hyousou(X18,X19).	表層属性の展開

X1 がマクロ表記された中間表現、X19 が表層構造を含んだ基本表現である。jg_macro_expand/2 は各属性を展開するモジュールを順に実行して、最終的に基本表現を得る。入力となるマクロ表記された中間表現は先に述べた 3 つの type が考えられるが、jg_macro_expand/2 の定義は一つしかなく、どのような type の中間表現が入力されて

も展開できる。

ユーザによるマクロ展開規則の調整としてヴォイス属性の処理を変えたければ、ヴォイス属性の処理はjg_voice/2で行うので、jg_voice/2の定義を変更すればよい。同様に、語順規則を変える場合はjg_gojun/2を、丁寧化の方式を変える場合はjg_polite/2をそれぞれ変更すればよい。また、あらたに属性を設けて展開してもよく、ある属性を削除してもよい。まったく別の方式で展開することも可能で、その場合はjg_macro_expand/2の定義を書き換えることになるが、以後ユーザの調整に必要と思われる述語を示していくので、その述語について定義の変更、追加を行うとよい。

以下では、jg_macro_expand/2から実行する18個の述語のそれぞれについて作成の手順を詳しく説明する。

jg_goi(X1,X2)

jg_goi/2は語彙属性を展開する。語彙属性はtype3の中間表現が持つ属性である。type3の中間表現は数属性などを伴って語を構成する。語彙属性の展開とは、辞書引きを行って、基本表現 {lex/...} の形にすることである。マクロ表記された中間表現の辞書引きは、jg_macro_dict/3を用いて行う。

jg_goi/2への入力、出力の例を示す。

入力 {語彙 / 自然}

出力 {lex/自然,
'*dterm'/{読み / しぜん, 語彙 / 自然, 類別 /[普通名詞, 連用名(単)]},
suf/[名詞,{読み / しぜん, 語彙 / 自然, 類別 /[普通名詞, 連用名(単)]}]}

一般に辞書引きはマクロ展開エンジン中の jg_get_dict/3を用いて行えばよく、jg_get_dict/3は入力がマクロ表記されている場合はjg_macro_dict/3を実行するように定義されている。しかし、ここではX1は必ずマクロ表記されているので直接、jg_macro_dict/3を実行する。

さらに、語彙属性に付随して数属性が存在する可能性があるため、jg_goi/2の中で数属性の展開を行う。type3の中間表現に固有の属性は語彙属性の展開時に行う。

以上のことから、jg_goi/2の定義は次のようになる。

```
jg_goi(X1,X2):-                          X1は語彙属性を含むマクロ表記された中間表現
      getRole(X1,語彙,LEX),!,
      jg_macro_dict(X1,H,Dt),            辞書引きを行う
      jg_kazu(X1,X2).                    数属性の展開を行う
```

jg_kazu/2の複数形の処理を行う規則は次のように定義すればよい。

```
jg_kazu(X1,X2):-                         X1は数属性を含むマクロ表記された中間表現
      getRole(X1,数,複),!,
      getRole(X1,suf,[H,Dt]),
      getRole(Dt,複数形,[N|_]),          辞書項目から複数形に関する情報を取り出す
      jg_kazu2(N,Dt,Dt2),                複数形を生成する
      masked_merge(X1,{suf/_,'*dterm'/_},
                   X2#{suf/[H,Dt2],'*dterm'/Dt2}).
```

数属性のように語彙に関する情報を別の属性として与える場合はjg_goi/2から展開規則を実行する。

jg_kankei(X1,X2)

jg_kankei は関係属性の展開を行う。関係属性の値は、中間表現、「並立」、「分配」、「特殊」のいずれかである。関係属性の値が中間表現の場合は展開した結果をX2のheadの値とすればよい。「並立」、「分配」、「特殊」関係の場合は必ず関係子属性を持っていて、関係子属性の値が対応するロールに付加される。「並立」と「分配」は2個以上のロール成分を持っているのでロール属性の展開の時にそれぞれのロール成分に関係子属性の値を付加する。特殊の場合は、

{head/ 関係子,comp/ ロールの値}

のように決めることができるので関係子の値を X2 の head の値とすればよい。
jg_kankei/2 への入力、出力の例を示す。

入力 {関係 /{語彙 / 頼る},...}

出力 {head/{lex/ 頼る},
'*dterm'/{読み / タヨル, 語彙 / 頼る, 語幹 / 頼, 活用 / ウ系強変化ラ行,
格 /[行為者:: が, 対象:: に], 受動 /{対象 /[対象:: が, 行為者:: に]},
アスペクト分類 /[継続], 派生 /[たより, 頼り]},
'*pos'/ 動詞,...}

文生成用辞書では名詞の中に形容名詞、サ変名詞、連用名詞などが含まれていて、名詞を関係として用いる場合がある。標準マクロ定義では関係属性の値は名詞としては用いないことになっているので、jg_kankei/2 の中で関係属性の値の動詞化を行う。
jg_kankei/2 の定義のうち関係属性の値が中間表現の場合の展開規則を示す。

```
jg_kankei(X1,X2#{head/REL3}):-
    getRole(X1,関係,REL1),
    jg_sentence2(REL1,REL2),        関係属性を展開する
    jg_doushika(REL2,REL3),         動詞化を行う
    masked_merge(X1,{head/_,suf/_,'*pos'/_,'*dterm'/_,'*last'/_},
                 X2#{head/REL3,
                     '*pos'/REL3!'*pos',
                     '*dterm'/REL3!'*dterm',
                     '*last'/REL3!'*last',
                     '*nano'/REL3!'*nano'}).
```

動詞化を行うには jg_make_verb/2 を用いればよいが、実行の前に、動詞化してもよいかどうか確認する必要がある。品詞属性による動詞以外の品詞の指定、派生属性による名詞化の指定がある時には動詞化をしない。

jg_hishuushoku(X1,X2)

jg_hishuushoku/2 は被修飾属性の展開を行う。被修飾属性は連体修飾属性と共に現れて名詞句を構成する。名詞句を基本表現に展開したものは次のような形式をしている。

{head/ 被修飾句,comp/ 連体修飾句}

被修飾句と連体修飾句のどちらを先に展開しても構わないが、先に被修飾句を展開して、連体修飾句属性の展開の後で名詞句全体の表層構造を生成する。jg_hishuushoku/2 では被修飾属性の値を取り出して、jg_sentence2/2 を実行して結果を X2 の head へ返すだけでよい。
jg_hishuushoku/2 への入力、出力の例を示す。

入力 {被修飾 /S1, 連体修飾 /MODIFIER,...}

出力 {head/S2, 連体修飾 /MODIFIER,...}

S2 は S1 をマクロ展開した基本表現である。

jg_hishuushoku/2 の定義は次のようになる。

```
jg_hishuushoku(X1,X2):-                 X1 は被修飾属性を含むマクロ表記された中間表現
    getRole(X1,被修飾,S1),!,
    jg_sentence2(S1,S2),                被修飾属性を展開する
    delete('*pos',X1,X2#{head/S2,'*pos'/S2!'*pos'}).
```

jg_hasei(X1,X2)

jg_hasei/2 は派生属性を展開する。派生の種類には「する」「なる」「ようだ」を付けて動詞にするものと、「の」「こと」をつけて名詞句にするものがある。また、属性値として「名詞化」という値を与えることによって「ビルを建設する」から「ビルの建設」へのような名詞化を行うこともできる。いずれの場合でも、X1 の head を派生させる。ただし、

{lex/ 豊か, 派生 / する}

のように X1 が head を持たない場合は X1 自身を派生させる。
jg_hasei/2 への入力、出力の例を示す。

入力 {head/{lex/ 豊か,suf/SUF1}, 派生 / する, ...}

出力 {head/{head/{lex/ する,suf/SUF2},comp/{lex/ 豊か,suf/SUF1},suf/[SUF1,SUF2]}, ...}

SUF1 は「豊か」の表層構造、SUF2 は「する」の表層構造である。

jg_hasei/2 による派生化処理は、派生辞を head とし、派生化の対象を comp とすればよい。jg_hasei/2 は次のようになる。

```
jg_hasei(X1,X2#{head/HEAD2}):-                  X1 は派生属性を含むマクロ表記された中間表現
        getRole(X1, 派生,G),
        getRole(X1,head,HEAD),                  head がある時
        jg_hasei2(HEAD#{派生 /G},HEAD2),        X1 の派生化属性の値に応じて HEAD を派生化
        masked_merge(X1,{head/_,'*pos'/_,'*case'/_,'*last'/_}
                        X2#{head/HEAD2,
                            '*pos'/HEAD2!'*pos',
                            '*case'/HEAD2!'*case',
                            '*last'/HEAD2!'*last'}).
jg_hasei(X1,X2):-                               X1 は派生属性を含むマクロ表記された中間表現
        getRole(X1, 派生,G),
        jg_hasei2(X1,X2).                       X1 を派生化
```

派生化は jg_hasei2/2 によって行う。「する」「なる」に対しては助動詞の「だ」の連用形を用いて「にする」「になる」を派生辞とする。「よう」に対しては派生化対象が名詞の場合に助動詞の「だ」の連体形を用いて「のようだ」を派生辞にするなどの処理を行っている。派生属性の値が「名詞化」のときは jg_make_noun/3 を用いて派生名詞化する。

派生化の際の細かい処理内容を変更したり、追加したりする場合には、jg_hasei2/2 を変更、追加する。

例として「する」を付加して「〜にする」という動詞に派生させる場合の処理を示す。派生対象は名詞でなければならない。これをチェックするためには '*pos' 属性を調べる。また「に」は助動詞の「だ」の連用形を用いる。「する」が後接しているため形態素生成部で「に」と活用する。動詞に派生させることから格情報を付加しなければならないが、そのために「する」の持っている格情報を使う。以上のことから次のような規則が書ける。

```
jg_hasei2(X1,X2):-
        getRole(X1, 派生, する),                    派生属性の値が「する」であれば
        getRole(X1,'*pos', 名詞),                   派生対象が名詞かどうか調べ
        jg_get_suf([助動詞, する],[助動詞,SURU]),    「する」の表層構造を生成する。
        jg_get_suf([助動詞, だ],DA),                「だ」の表層構造を生成する。
        jg_get_case({lex/ する},Case),               「する」の格情報を取り出す。
        masked_merge(X1,{head/_,comp/_,suf/_,'*dterm'/_,'*pos'/_},
                        X2#{head/{lex/ にする,suf/[DA,[助動詞,SURU]]},
                            comp/X1,
                            suf/[[X1!suf,DA],[助動詞,SURU]],
                            '*case'/Case,
```

```
'*pos'/ 動詞,
'*dterm'/SURU}.
```

jg_rentai(X1,X2)

jg_rentai/2 は連体修飾属性の展開を行う。標準マクロ定義では、X1 は被修飾属性が展開された中間表現である連体修飾属性の値は中間表現であるから、jg_rentai/2 では連体修飾属性の値を取り出し、jg_sentence2/2 を用いて展開して、得られた表層構造を X1 の表層構造と合成して、X2 の表層構造にする。

jg_rentai/2 への入力、出力の例を示す。

入力 `{head/MODIFEE, 連体修飾 /S1,...}`

出力 `{head/MODIFEE,comp/S2,suf/[S2!suf,MODIFEE!suf],...}`

S2 は S1 をマクロ展開した基本表現である。

jg_rentai/2 の定義は次のようになる。

```
jg_rentai(X1,X2):-                    X1 は連体修飾属性を含むマクロ表記された中間表現
    getRole(X1,連体修飾,R1),
    jg_sentence2(R1,R2),
    getRole(X1,suf,SUF1),
    getRole(R2,suf,SUF2),
    masked_merge(X1,{head/_,comp/_,suf/_},
                 X2#{head/X1,comp/R2,suf/[SUF2,SUF1]}).
```

標準マクロ定義では連体修飾属性の値は、幾つかの連体修飾句を表現するために連体修飾句のリストが許されている。これに対応した jg_rentai/2 の定義が追加されている。さらに連体修飾句が名詞である時に「の」を間に入れたり、動詞化された形容名詞の時、連体形の「の」に活用させる指示を挿入する処理を行っている。

jg_voice(X1,X2)

jg_voice/2 はヴォイス属性の展開を行う。X1 はヴォイス属性の値に応じて態変化する文であり、関係属性、ロール属性を持っている。ロール属性は省略されていることがある。態変化による助詞変換を伴うから、先に関係属性、ロール属性の展開を行わなければならない。また、語順の決定をしてしまうとロール成分が構造化されてしまうため助詞変換がしにくいので、語順決定の前で行う。

ここでは、X1 に関係属性、ロール属性があり、関係属性の展開の後でヴォイス属性の展開を行う場合の手順を示す。

1. X1 の関係属性の値は展開されていて X1 の head の値にバインドされている。 head の値とロール属性の値を用いて基本表現を生成する。
2. 1で得られた基本表現に対して態変化を行う。ヴォイス属性の属性値が「可能」の場合は、1で得られた基本表現の head を可能動詞化する。生成用辞書を参照して、可能動詞化ができれば語幹に - eru を付加して可能動詞にする。可能動詞化ができない場合は得られた基本表現の head に「れる」を付加して X2 の head とする。ヴォイス属性の値が「受動」の場合は、基本表現の head に「れる」を付加し、「使役」の場合は基本表現の head に「せる」を付加する。
3. 態変化に伴う助詞変換を行う。1で得られた基本表現の comp の値は助詞変換前のロール群である。助詞変換規則は受動の場合は生成用辞書の記述に従い、その他の態変化は助詞変換規則を用いて行う。

jg_voice/2 への入力、出力の例を示す。

入力 `{head/HEAD, ヴォイス / 受動, ロール /ROLES1,...}`

出力 `{head/{head/{lex/れる,suf/SUF},comp/HEAD,suf/[HEAD!suf,SUF]},comp/ROLES2,...}`

ROLES2 は ROLES1 をマクロ展開した基本表現である。

以上のことからjg_voice/2は次のように定義すれば良い。

```
「れる」を付加せずに可能動詞を生成する場合
jg_voice(X1,X2):-                                  X1はヴォイス属性を含むマクロ表記された中間表現
      getRole(X1, ヴォイス, 可能),
      jg_roles(X1,BTREE),                          X1のロールを展開する
      jg_get_voice(可能,HEAD,X1!head!'*dterm'), BTREE!headを可能動詞化してHEADとする
      change_case(可能,BTREE,COMP,CASE),   BTREE!compを助詞変換してCOMPとする
      masked_merge(X1,{head/_,'*case'/_},
                   X2#{head/HEAD,comp/COMP,'*case'/CASE}).
「れる」「せる」などを付加する場合
jg_voice(X1,X2):-
      getRole(X1, ヴォイス,V),
      jg_roles(X1,BTREE),                          X1のロールを展開する
      jg_get_voice(V,HEAD),                        Vの値に応じてなどの接辞を決めHEADとする
      change_case(V,BTREE,COMP,CASE),
      masked_merge(X1,{head/_,'*case'/_},
                   X2#{head/{head/HEAD,comp/BTREE!head,suf/[BTREE!head!suf,HEAD!suf]}
                       comp/COMP,
                       suf/[COMP!suf,[BTREE!comp!suf,HEAD!suf]]}).
```

助詞変換はget_change_case_list/4で得られる助詞変換リストに基づいて行われる。助詞変換規則を調整する場合は、get_change_case_list/4の定義を追加、変更する。使役に対する助詞変換リストを求めるget_change_case_list/4を示す。

```
get_change_case_list(使役,BTREE,CASE):-
      get_case_with_j([に, で, から],BTREE!comp,RN),
      get_case_with_j([が],COMP, 原因),
      jg_merge_case([RN::: が, 原因::: を],BTREE!'*case',CASE).
```

BTREEは変換前のロール群をcomp属性の値として持つ基本表現である。CASEに助詞変換リストが返される。助詞変換リストCASEは生成用辞書の格リストと同じ形式をしている。上記の助詞変換規則は、「に」「で」「から」のどれかを接辞として持っているロールRNをBTREEのcomp属性の値から取り出し、さらに接辞「が」を持つ原因ロールがあれば、RNを「が」に原因ロールを「を」に変換することを表している。

jg_aspect(X1,X2)

jg_aspect/2はアスペクト属性を展開する。アスペクト属性の展開は、X1のロールとは無関係である。アスペクト辞を決めてX1の headの値に付加する。ヴォイス属性と同様に関係属性が展開されていることが必要であるが、語順の決定よりは前に行う。

アスペクト辞は、「ていく」「ている」のように与えられるがアスペクト辞の表層構造の生成は次に示すようなテーブルを用いて行う。

```
jg_get_aspect(ていく,{lex/ていく,suf/[[助辞, て],[助動詞,Dt]]}):-
            jg_dict(助動詞, いく,Dt).
```

上記のjg_get_aspect/2はアスペクト属性の値として与えられた「ていく」の表層構造を定義している。助辞の「て」と助動詞の「いく」から表層構造を生成する。アスペクト辞を追加、変更する場合はjg_get_aspect/2を追加、変更する。助動詞は生成用辞書に登録されていなければならないため辞書の記述にも注意する必要がある。

jg_aspect/2への入力、出力の例を示す。

入力 `{head/REL, アスペクト/ていく,...}`

出力 `{head/{head/{lex/ていく,suf/SUF},comp/REL,suf/[REL!suf,SUF]},...}`

jg_roles(X1,X2)

jg_roles/2 はロール属性の展開を行う。入力 X1 の head の値は関係属性を展開した基本表現である。ロール属性の値を展開して comp の値とする。

jg_roles/2 への入力、出力の例を示す。

入力 {head/HEAD, ロール /{行為者 /AGT1, 対象 /OBJ1},...}

出力 {head/HEAD,comp/{行為者 /AGT2, 対象 /OBJ2},...}

AGT2、OBJ2 はそれぞれ AGT1、OBJ1 をマクロ展開した基本表現である。

標準マクロのロール属性の値を例にどのような処理を行わなければならないかを示す。
ロール属性の値は一般には次のような形式をしている。

{ロール名 1/ ロール成分 1, ロール名 2/ ロール成分 2,...}

これらを、ロールごとに展開する。標準マクロ定義ではロール成分は補語属性、派生属性、取り立て詞属性、数量化属性、記号属性 からなる。したがってロール成分の展開を行う述語は次のようになる。

```
jg_role(ROLE1,ROLE6):-                  ROLE1 はロール成分
      jg_hogo(ROLE1,ROLE2),             補語属性の展開 (ロールの内容)
      jg_hasei(ROLE2,ROLE3),            派生属性の展開 (〜ことなど)
      jg_toritate(ROLE3,ROLE4),         取り立て詞属性の展開 (は、だけなど)
      jg_suuryouka(ROLE4,ROLE5),        数量化属性の展開 (みな、すべてなど)
      jg_kigou(ROLE5,ROLE6).            記号属性の展開 (読点など)
```

jg_role/2 を用いてロール成分を順に展開する。jg_roles/2 は次のようになる。

```
jg_roles(X1,X2):-                       X1 は文をマクロ表記した中間表現
      getRole(X1, ロール,ROLES1),       ロール属性の値を取り出す
      jg_get_case(X1,CASE),             X1 の格情報を得る
      jg_roles2(ROLES1#{'*case'/CASE,'*pdt'/X1!'*dterm'},ROLES2),
      masked_merge(X1,{comp/_},
                   X2#{comp/ROLES2,'*case'/CASE}.
```

ロールの展開時に '*pdt'(Parent's dictionary term) という属性を用いる。'*pdt' 属性には X1 の '*dterm' 属性の値を入れる。'*pdt' 属性はロール成分を順に展開する時に参照される。

jg_roles2/2 は次のようになる。

```
jg_roles2(ROLES1,ROLES2):-
      getRole(ROLES1,R_NAME,ROLE),      ロール名とその値を取り出す
      jg_macro_dict(ROLE,H,Dt),         ロール成分の辞書引きを行う
      jg_role(ROLE#{'*case'/ROLES1!'*case',
                    '*pdt'/ROLES1!'*pdt',
                    '*cname'/R_NAME},ROLE2),
      delete(R_NAME,ROLES1,ROLES),
      jg_roles2(ROLES,ROLES2).
```

jg_roles/2 はロール属性の値全体に関する処理を行い、jg_roles2/2 はロールごとの処理を行う。jg_role/2 で補語属性、派生属性などを展開するが、そのうち補語属性の展開を行う jg_hogo/2 の処理は次のようになる。

```
jg_hogo(ROLE1,ROLE2):-
    getRole(ROLE1, 補語,COMP1#{'*pdt'/ROLE1!'*pdt',
                               '*cname'/X1!'*cname',
                               '*dterm'/X1!'*dterm',
                               '*pos'/X1!'*pos',
                               '*setsuji'/SETSUJI}),
    jg_get_setsuji(ROLE1,SETSUJI),          接辞を求める
    jg_sentence2(COMP1,COMP2),              補語属性の値を展開する
    jg_make_suf(COMP2!suf,SETSUJI,SUF),     補語の表層構造と接辞の表層構造を合成する
    masked_merge(X1,{head/_,comp/_,suf/_,'*pos'/_,'*last'/_},
                 X2#{head/SETSUJI,
                     comp/COMP2,
                     suf/SUF,
                     '*setsuji'/SETSUJI,
                     '*pos'/COMP2!'*pos',
                     '*last'/COMP2!'*last'}).
```

jg_get_setsuji/2 はまず生成用辞書の格リスト ('*case' の値) を調べ次に 'ロール' という述語を用いて接辞を決定する。例えば「目的」というロール名に対してロール成分が名詞ならば「のために」という接辞を付加する場合、次のようなロール述語を定義しておく。

```
ロール(目的, 名詞,_,[[接辞, の],[名詞, ため],[格助詞, に]]):-!.
```

ロール述語の第一引数はロール名、第二引数はロール成分の品詞、第三引数はロール成分の補語属性の値、第四引数は接辞である。接辞は[品詞, 表層表現]またはそのリストで表される。

接辞を調整する場合はロール述語を変更、追加する。また、生成用辞書の格リストを用いない場合は jg_get_setsuji/2 を修整して '*case' を参照しないようにする。

関係属性の値が「並立」、「分配」のときはロール属性の値は set(SET) のようにセットとして与えられる。このときは、関係子属性の値を取り出し接辞の代わりに用いる。次に SET からロール成分を取り出して jg_roles2/2 を実行する。jg_role_set/4 を参照のこと。

関係属性の値が特殊の時は標準マクロ定義ではロール名を省略している。この場合はロール名を「特殊」にして jg_roles2/2 を実行する。

jg_gojun(X1,X2)

jg_gojun/2 はロールの語順の決定を行う。基本表現では語順は構造として表現される。jg_roles/2 でロール属性を展開した結果、中間表現 X1 は次のような形式をしている。

{head/HEAD,comp/{ロール名 1/ROLE1, ロール名 2/ROLE2,..}}

comp の値を取り出し、ロールの語順を決めて head と comp による二分木構造 を生成すればよい。

jg_gojun/2 への入力、出力の例を示す。

入力 {head/REL,comp/{行為者 /AGT, 対象 /OBJ,..}}

```
出力 {head/{head/REL,
            comp/AGT,
            suf/[AGT!suf,REL!suf]},
      comp/OBJ,
      suf/[OBJ!suf,[AGT!suf,REL!suf]]}
```

標準マクロ定義では gojun(ROLE1,ROLE2) という述語でロールの順序を決定している。gojun/2 は ROLE1 が ROLE2 より前に置かれる時に成功する述語である。gojun/2 はまず取り立て詞の「は」があるかどうかなどを調べるが多くの場合はロール名によって語順を決定している。

ロール名による語順の決定は述語、格分類表/2によって与えられたロールごとの点数の比較による。格分類表/2の第二引数の値が大きいほどそのロールが前に置かれることを意味していて、この値を調整することによってロール名による語順の調整をすることができる。

語順規則を変更する場合は、ROLE1がROLE2よりも前に置かれる時の条件を変更、追加して、gojun/2の定義を変更する。

jg_mitomekata(X1,X2)

jg_mitomekata/2は認め方属性の展開を行う。認め方属性の値が「否定」の場合に助動詞の「ない」「ぬ」を付加する。標準マクロ定義では認め方属性の値が「否定」の場合は「ない」、「否定(ぬ)」の場合は「ぬ」と区別している。

認め方属性の展開処理は、「ない」「ぬ」をX2のheadとし、X1をcompとする。標準マクロ定義では否定する述語の活用がダ系である時、取り立て詞の「は」を挿入している。

jg_mitomekata/2への入力、出力の例を示す。

入力 {head/HEAD,comp/COMP,suf/SUF1,認め方/否定,...}

出力 {head/{lex/ない,suf/SUF2},comp/{head/HEAD,comp/COMP,...},suf/[SUF1,SUF2],...}

jg_toritate(X1,X2)

jg_toritate/2は取り立て詞属性の展開を行う。取り立て詞属性の値は「は」「も」のように与えられる。取り立て詞の表層構造を生成して、X2のheadとし、X1をcompとする。取り立て詞は、生成用辞書に登録されている必要がある。

jg_toritateshi/2への入力、出力の例を示す。

入力 {head/HEAD,comp/COMP,suf/SUF1,取り立て詞/は,...}

出力 {head/{lex/は,suf/SUF2},comp/{head/HEAD,comp/COMP,...},suf/[SUF1,SUF2],...}

jg_meishika(X1,X2)

jg_meishika/2は中間表現X1の名詞化を行うかどうか調べ、名詞化を行う場合にはjg_make_noun/3を用いて名詞化を行う。X1がロール成分であって、ロール成分の名詞化条件を満たさない場合、およびX1が関係属性の値として使われている場合は名詞化しない。

jg_meishika/2への入力、出力の例を示す。

入力 {head/{lex/頼る},comp/COMP,suf/SUF1,名詞化/形式,...}

出力 {head/{lex/こと,suf/SUF2},comp/{head/{lex/頼る},comp/COMP,...},suf/[SUF1,SUF2],...}

ロール成分の名詞化条件はjg_nom/4で定義される。jg_nom/4の第一引数はロール名、第二引数は接辞である。第三引数にはX1が動作名詞、形容名詞ならば派生が、そうでなければ形式が与えられる。第四引数には名詞化の方法として派生または形式が返される。jg_nom/4は次のように定義する。

jg_nom(原因,[接辞,によって],Nom,Nom):-!.

例に示したjg_nomはロール名が原因で、接辞が「によって」ならばロール成分が動作名詞、形容名詞の場合には派生名詞化し、そうでなければ形式名詞化することを表している。

jg_jisei(X1,X2)

jg_jisei/2は時制属性の展開を行う。時制属性の値は「現在」または「過去」である。「過去」の時に助辞の「た」をX2のheadとし、X1をcompとする。

jg_jisei/2への入力、出力の例を示す。

入力 {head/HEAD,comp/COMP,suf/SUF1,時制/過去,...}

出力 {head/{lex/た,suf/SUF2},comp/{head/HEAD,comp/COMP,...},suf/[SUF1,SUF2],...}

jg_modal(X1,X2)

jg_modal/2 はモーダル属性の展開を行う。モーダル属性の処理はムード、認め方、時制、態度に分かれ標準マクロ定義ではこの順序で展開を行う。 jg_modal/2 を示す。

```
jg_modal(X1,X5):-
        jg_mood(X1,X2),
        jg_modal_mitomekata(X2,X3),
        jg_modal_jisei(X3,X4),
        jg_taido(X4,X5).
```

ムード、態度はアスペクト属性の展開と同じ様に属性値から表層構造を生成して、 X1 に付加すればよい。
jg_mood/2 への入力、出力の例を示す。

入力 {head/HEAD,comp/COMP,suf/SUF1, ムード / ちがいない,...}

出力 {head/{lex/ ちがいない,suf/SUF2},comp/{head/HEAD,comp/COMP,...},suf/[SUF1,SUF2]}

ムードを表す接辞の表層構造を得るための述語 jg_get_mood/2 を示す。

```
jg_get_mood(ちがいない,{ lex/ ちがいない,suf/[助動詞,Dt]} ):-
                jg_dict(助動詞, ちがいない,Dt).
```

態度を表す接辞の種類を追加する場合は、必要な助動詞を生成用辞書に登録し、 jg_get_taido/2 の定義を追加する。ムード属性の展開も同様にして行う。ムードを表す接辞の種類を追加する場合は、必要な助動詞を生成用辞書に登録し、jg_get_mood/2 の定義を追加する。

モダリティに関する認め方属性と、時制属性の展開はそれぞれ jg_mitomekata/2,jg_jisei/2 を利用する。 jg_modal_jisei/2 を示す。

```
jg_modal_jisei(X1,X2):-
        getRole(X1! モーダル, 時制,T),
        jg_jisei(X1#{時制 /T},X2).
```

jg_polite(X1,X2)

jg_polite/2 は X1 を丁寧化して X2 に返す。 X1 が表現する文の文末の語彙データが X1 の '*last' 属性にバインドされている。

jg_polite/2 への入力、出力の例を示す。

入力 {head/{lex/ ちがいない,suf/SUF1},comp/COMP,suf/[COMP!suf,SUF1], モーダル /{丁寧 /yes},...}

出力 {head/{lex/ ちがいありません,suf/SUF2},comp/COMP,suf/[COMP!suf,SUF2],...}

丁寧化の方法を示す。

丁寧化の方法を調整するためには p_conv/3 を変更、追加する。 p_conv/3 の例を示す。

```
p_conv([終助詞,{語彙 / な}],COMP,{head/{lex/ てはいけません },
                                  comp/COMP,
                                  suf/SUF}):-
        jg_get_suf([取り立て詞, は],HA),
        jg_get_suf([助動詞, いけません],IKEMASEN),
        jg_make_suf([[助辞, て],HA,IKEMASEN],COMP!suf,SUF).
```

p_conv/3 の第一引数は丁寧化を行う文の文末の語彙データである。第二引数は丁寧化を行う文から第一引数を取り除いた残り、第三引数は丁寧化を行った文である。例にあげた p_conv は終助詞の「な」で終わっている文は「な」を「てはいけません」に変えるという規則を表している。

表 3.1: 丁寧化の処理内容

'*last' 属性の値	処理内容
ウ系の活用語、である	「ます」を付加する。
イ系、ダ系の活用語	「です」を付加する。
ん＋た	「ん＋です＋た」に変える
な	「てはいけません」に変える
命令形	「なさい」を付加する
くれる	「ください」に変える
だろう、であろう	「でしょう」に変える
ちがいない	「ちがいありません」に変える
かもしれない	「かもしれません」に変える
てもよい	「てもかまいません」に変える
といってよい	「といってかまいません」に変える
いけない	「いけません」に変える
なければならない	「なければなりません」に変える
ない (形容詞)	「ありません」に変える
イ系、ダ系の活用語＋ない (助動詞)	「はありません」に変える
ない (助動詞)	comp を丁寧化して、「ん」を付加する。
その他	comp を丁寧化する

jg_kigou(X1,X2)

jg_kigou/2 は記号属性の展開を行う。標準マクロ定義の記号属性の値は句点、読点、疑問などである。jg_get_symbol/2 を用いて記号の表層構造を求め、jg_make_symbol_suf/3 を用いて表層構造を合成すればよい。記号の種類を追加する場合は jg_get_symbol/2 を追加する。

jg_kigou/2 への入力、出力の例を示す。

入力 `{head/HEAD,comp/COMP,記号 / 句点,...}`

出力 `{cont/{head/HEAD,comp/COMP,...},symbol/{type/right,right/。}}`

jg_setsuzokushi(X1,X2)

jg_setsuzokushi/2 は接続詞属性の展開を行う。基本的には接続詞の表層構造を生成して X2 の comp とし、X1 を head とすれば良いが標準マクロ定義では接続詞属性の値としてリストを許している。リストの順に接続詞を付加する。

jg_setsuzokushi/2 への入力、出力の例を示す。

入力 `{head/HEAD,comp/COMP,suf/SUF1,接続詞 / しかし,...}`

出力 `{head/{head/HEAD,comp/COMP,...},comp/{lex/ しかし,suf/SUF2},suf/[SUF2,SUF1]}`

jg_hyousou(X1,X2)

jg_hyousou/2 は X1 に表層属性があるときに、表層属性の値を X1 の表層表現とする。表層属性の値から[string, 表層属性の値]という表層構造を生成する。

jg_hyousou/2 への入力、出力の例を示す。

入力 `{表層 / 自然に頼る}`

出力 `{lex/ 自然に頼る,suf/[string, 自然に頼る]}`

3.2 文生成用辞書作成手順

文生成用辞書は、中間表現から表層文生成の過程で必要な統語情報 (接辞に関する情報、活用情報) を提供する。

3.2.1 文生成用辞書の構成

文生成用辞書では各辞書エントリを述語 jg_dict(Hinshi,Entry_name, Dterm)で記述する。これらの引き数は以下の通りである。

- Hinshi
  そのエントリの単語の品詞名。
- Enrty_name
  エントリ名。
- Dterm
  そのエントリの単語の辞書項目。CIL の部分項である。

辞書項目の検索は、Entry_name を指定して jg_dict の unification を実行することで行われる。又、マスタ辞書に記述のある動詞、名詞、形容詞、副詞、連体詞の 5 品詞の関しては原則としてマスタ辞書の見出し語レコードと語義レコードから文生成用辞書のエントリを作成する。以下では各引き数の詳細、マスタ辞書との関連、辞書項目の記述例を述べる。

1. Hinshi そのエントリの単語の品詞名。現行の文生成用辞書の品詞分類は、自立語では, 動詞, 名詞, 形容詞, 副詞, 連体詞, 接続詞の 6 品詞, 付属語では, 助動詞, 取り立て詞, 格助詞, 接続助詞, 終助詞, 接尾語の 6 品詞となっている。マスタ辞書に記述のあるものは、「品詞」属性 (見出し語レコード) の値をそのまま記述する。

   Hinshi::= ' 動詞 '|' 名詞 '|' 形容詞 '|' 副詞 '|' 連体詞 '|' 助動詞 '|' 取り立て詞 '|' 接続助詞 '|
   ' 終助詞 '|' 接続詞 '|' 格助詞 '|' 接尾語 '

2. Entry_name 現行の文生成用辞書ではその単語の表層表現を用いている。但し、

   - 多語義語[1]の場合は、各語義に対してエントリを作成しエントリ名としてはその語の表層表現に post-fix を付加したもの (' 表層表現 ',' 表層表現 2',' 表層表現 3',...) を記載する。
   - 副詞については、“に”や“と”が後接しても同じ意味の副詞になる場合は、“に”や“と”が後接した形に対してもエントリを作成する。

   マスタ辞書に記述のあるものは、エントリ名として見出し語レコードの「表層表現」を記述する。そのとき、マスタ辞書の見出し語レコードに対して複数の語義レコードがある場合は多語義に相当し、副詞に関してはマスタ辞書の「に属性[2]」の値にしたがってエントリを作成する。マスタ辞書の見出し語テーブル、語義テーブルと生成用辞書のエントリ名との関係の例を表 3.2に示す。

3. Dterm 以下の3.2.2から3.2.9で各品詞ごとの辞書項目について述べる。尚、各品詞の辞書項目一覧表中の「必須」において、○の付いたものは必須属性であり値がない場合には、空値 ([],{},'') をとる。それ以外のものは、値がない場合には属性そのもの (スロット) を記述しない。又、「マスタとの対応」において、(見) はマスタ辞書の見出し語レコードの属性、(語) はマスタ辞書の語義レコードの属性であることを示す。

3.2.2 動詞の辞書項目

表 3.3に動詞の辞書項目一覧を掲げる。

1. id: 〈語彙識別番号〉 [アトム]
   マスタ辞書の語義指標を記述する。動詞の場合，以下の通り。
   動詞$X_1X_2X_3X_4X_5X_6C$　　但し，X_i: 数字,C: アルファベット (多語義に対応)

[1]現行の文生成用辞書では統語的な情報のみを取り扱っている。ここで言う多語義とは、例えば動詞などで異なる格パターンをとるものをさす。
[2]その副詞に“に”や“と”が後接して同じ意味を持つ副詞になるかどうかを示す情報 (に:“に”のみを後接、と:“と”のみを後接、にと:“に”と“と”を後接、単独: どちらも後接しない)

表 3.2: マスタ辞書と文生成用辞書のエントリ

見出し語レコード	語義レコード	生成用辞書エントリ名
表層表現 / する,...	識別番号 /001530A,...	する
	識別番号 /001530B,...	する 2
	識別番号 /001530C,...	する 3
表層表現 / いつも,...	にと / 単独,...	いつも
表層表現 / すぐ,...	にと / に,...	すぐ
		すぐに

表 3.3: 動詞の辞書項目

属性名	値の型、実現値	必須	マスタとの対応
id	アトム		語義指標 (見)
語彙	アトム	○	表層表現 (見)
読み	リスト (カタカナ表記)	○	読み (見)
語幹	アトム	○	表層表現 (見)
活用	アトム [表 3.4]	○	活用型 (見)、表層表現 (見)
格	〈格リスト〉	○	能動表層格 (語)
受動	〈受動格リスト〉		受動表層格 (語)
アスペクト分類	〈アスペクトリスト〉	○	相 (語)
可能動詞化	〈可能動詞〉	○	可能動詞化 (語)
派生	リスト (アトム)	○	派生名詞 (語)

2. 語彙: 〈語彙〉 [アトム]
   その語彙の表層表現。マスタ辞書の「表層表現」(見出し語レコード) から作成する。

3. 読み: 〈読み〉 [リスト (カタカナ表記)]
   その語彙の読みのカタカナ表記をリストの要素としたもの。マスタ辞書の「読み」(見出し語レコード) を記述する。

4. 語幹: 〈語幹〉 [アトム]
   その動詞の語幹。マスタ辞書の「表層表現」(見出し語レコード) から抽出する。語幹がない場合は ''(空アトム) が記載される。

5. 活用: 〈動詞の活用型〉 [アトム]
   その動詞の活用型。マスタ辞書の「活用型」「表層表現」(いずれも見出し語レコード) から作成する。マスタ辞書の「活用型」との対応は表 3.4の通りである。

6. 格: 〈格リスト〉 [リスト (演算子適用項)]
   その動詞のとる格の情報。マスタ辞書の「能動表層格」(語義レコード) から作成される。リストの要素として各格の深層格名と表層格名 (接辞) リストの対が演算子適用項 [3]で記述される。

   〈格リスト〉 ::=[格属性名::: 〈表層格リスト〉,...]

   〈表層格リスト〉 ::= 表層格名|[表層格名,...]

[3]演算子としては ':::' を用いるので，辞書の先頭で演算子宣言をしておく必要がある。

深層格名は、マスタ辞書上ではアルファベット三文字の略称で記述され、文生成用辞書上では漢字で記述される。マスタ辞書の深層格名との対応を表 3.5に示す。尚，マスタ辞書上では，原則として必須格について，表層文中の語順も考慮して記載されている。

7. 受動:〈受動格リスト〉[部分項]
その動詞が受動態で用いられるときの格の情報。マスタ辞書の「受動表層格」(語義レコード)から作成される。部分項の属性名としては，受動化されて主格になった深層格名を記述し，属性値としてそのときの〈格リスト〉(6参照)を記述する。

〈受動格リスト〉::={格属性名 / 〈格リスト〉,...}

この属性はその語に直接受動態が存在するとき(マスタ辞書中の「直接受動」='あり'のとき)のみ記述され，それ以外の場合は記述されない。

8. アスペクト分類:〈アスペクトリスト〉[リスト]
その語のアスペクト情報。マスタ辞書の「相」(語義レコード)を記述する。

〈アスペクトリスト〉::=[〈アスペクト〉,...]

〈アスペクト〉::='状態'|'準状態'|'瞬間'|'継続'

9. 可能動詞化:〈可能動詞化〉[アトム]
その動詞が可能動詞に派生できるかどうかを記述する。マスタ辞書中の「可能動詞化」(語義レコード)を記述する。その語自身が可能動詞の場合は'可能動詞'とする。

〈可能動詞化〉::='可'|'不可'|'可能動詞'

10. 派生:〈派生名詞語彙リスト〉[リスト]
その動詞の派生名詞をリストの要素として記述する。マスタ辞書の「派生名詞」(語義レコード)を記述する(派生名詞がない場合は[]が記載される)。

表 3.4: マスタ辞書と文生成用辞書の動詞の活用型

マスタ辞書	文生成用辞書の活用型	
強変化	ウ系強変化カ行	カ行五段活用
	ウ系強変化ガ行	ガ行五段活用
	ウ系強変化サ行	サ行五段活用
	ウ系強変化タ行	タ行五段活用
	ウ系強変化ナ行	ナ行五段活用
	ウ系強変化バ行	バ行五段活用
	ウ系強変化マ行	マ行五段活用
	ウ系強変化ラ行	ラ行五段活用
	ウ系強変化ワ行	ワ行五段活用
弱変化	ウ系弱変化	上一段活用、下一段活用
ある	ウ系アル	"ない"が後接しない
いる	ウ系イル	"れる"、"せる"が後接しない
なさる	ウ系ナサル	命令形が"～い"となる
いく	ウ系強変化イク	カ行五段で連用形が促音便
カ変	ウ系カ変	カ行変格活用
サ変	ウ系サ変	サ行変格活用
弱サ変	ウ系サ変ザ行	"～じる"/"～ずる"

表 3.5: マスタ辞書の深層格名と文生成用辞書の格属性名

agt	行為者	obj	対象	ods	記述対象	loc	対象空間
sou	始点	pex	存在場所	dir	方向	opp	対抗者
com	比較基準	exp	経験者	prd	生産物	mat	材料
via	通過点	goa	終点	pos	所有主	par	随伴者
cot	内容	imp	道具	cau	原因	pur	目的
man	様態	tim	時刻	tfr	始点時刻	tto	終点時刻
tlm	期限	pla	場所	pch	変化点	deg	程度
dur	継続期間	frq	頻度	cco	共同者		

3.2.3 名詞の辞書項目

表 3.6に名詞の辞書項目一覧を掲げる。表中の「必須」において、“動”, “形”はそれぞれ動作性名詞(「類別」=[動作名,...]), 形容動詞性名詞(「類別」= [形動名(-),...]のとき必須属性であることを示す。

表 3.6: 名詞の辞書項目

属性名	値の型、実現値	必須	マスタとの対応
id	アトム		語義指標(見)
語彙	アトム	○	表層表現(見)
読み	リスト(カタカナ表記)	○	読み(見)
類別	〈名詞の類別リスト〉	○	サ変動詞用法(語)、形容動詞用法(語)、副詞的用法(語)
にと	〈にと1〉		にと(語)
複数形	〈複数形リスト〉		可算性(語)、畳語用法(語)、接尾辞(語)
格	[動詞と同様]	動、形	サ変動詞用法(語)、形容動詞用法(語)
受動	[動詞と同様]		サ変動詞用法(語)
アスペクト分類	[動詞と同様]	動	サ変動詞用法(語)
助数詞	リスト(アトム)		助数詞(語)

1. id: 〈語彙識別番号〉 [アトム]
   マスタ辞書の語義指標を記述する。名詞の場合, 以下の通り。
   $K_1K_2X_1X_2X_3X_4X_5X_6C$　　但し, X_i: 数字,C: アルファベット(多語義に対応)
   K_1K_2: ‘名普’|‘名サ’|‘名形’(「類別」に対応)

2. 語彙: 〈語彙〉 [アトム]
   動詞と同様。

3. 読み: 〈読み〉 [リスト(カタカナ表記)]
   動詞と同様。

4. 類別: 〈名詞の類別リスト〉 [リスト(複合項, アトム)]
   その名詞の品詞性の情報をリストの要素として持つ。マスタ辞書の「サ変動詞用法」, 「形容動詞用法」, 「副詞的用法」(いずれも語義レコード)などを参照して作成される。即ち, マスタ辞書中に, 「サ変動詞用法」属性があれば‘動作名’, 「形容動詞用法」属性があれば‘形動名(〈連体法〉)’, いずれの属性もなければ‘普通名詞’を類別リストに記述する[4]。更に, マスタ辞書中に「副詞的用法」属性があれば, ‘連用名(〈連用法〉)’も

[4] 同一の表層表現でサ変動詞用法と形容動詞用法があるものは別語義とする。即ち、動作名, 形動名, 普通名詞に関してはこのうちのどれか1つだけが記載される。

記述する。複合項の引き数に付いては，〈連体法〉はマスタ辞書中の「連体法[5]」の値を，〈連用法〉はマスタ辞書中の「副詞的用法[6]」の値を記述する。

〈名詞の類別リスト〉::=[〈類別名〉,...]

〈類別名〉::= '動作名'|'形動名(〈連体法〉)'|'普通名詞'|'連用名(〈連用法〉)'

〈連体法〉::= なの|な|の|たる

〈連用法〉::= 単|節|両

5. にと:〈にと1〉[アトム]
その名詞が“する”や“なる”に前接するときの語尾を示す情報。マスタ辞書の「にと」属性(語義レコード)の値を記述するが，マスタ辞書に「にと」属性がない場合は記述しない。

〈にと1〉::='に'|'と'|'にと'

6. 複数形:〈複数形リスト〉[リスト(アトム,複合項)]
その名詞の複数形に関する情報。マスタ辞書の「可算性」属性(語義レコード)の値が'数','不明'の場合のみ記述され，「畳語用法」,「接尾辞」属性(いずれも語義レコード)の値から作成される。単複同型の場合は[]が記載される。

〈複数形リスト〉::=[]|[〈複数形〉,...]

〈複数形〉::='畳語'|'接尾辞(〈接尾辞〉)'

〈接尾辞〉::='ら'|'たち'|'達'|'供'

7. 格:〈格リスト〉[リスト(演算子適用項)]
その語が動作名詞または形容動詞性名詞の場合の格情報が記述される。マスタ辞書の「サ変動詞用法」(語義レコード)または「形容動詞用法」(語義レコード)の「能動表層格」属性の値を記述する。普通名詞の場合は記述されない。

8. 受動:〈受動格リスト〉[部分項]
その語が直接受動態の存在する動作名詞の場合に，サ変動詞としての受動格情報が記述される。マスタ辞書の「サ変動詞用法」(語義レコード)の「受動表層格」属性の値を記述する。直接受動態の存在しない動作名詞，形容動詞性名詞，普通名詞の場合は記述されない。

9. アスペクト分類:〈アスペクトリスト〉[リスト(アトム)]
その語が動作名詞の場合に，アスペクト情報が記述される。マスタ辞書の「サ変動詞用法」(語義レコード)の「相」属性の値を記述する。動作名詞以外の場合は記述されない。

10. 助数詞:〈助数詞リスト〉[リスト(アトム)]
その名詞の助数詞をリストの要素として持つ。マスタ辞書に「助数詞」属性(語義レコード)がある場合のみ記述され，「助数詞」属性の値が記述される。

3.2.4 形容詞の辞書項目

表3.7に形容詞の辞書項目一覧を掲げる。

1. id:〈語彙識別番号〉[アトム]
マスタ辞書の語義指標を記述する。形容詞の場合，以下の通り。

形容$X_1X_2X_3X_4X_5X_6C$　　但し，X_i:数字,C:アルファベット(多語義に対応)

2. 語彙:〈語彙〉[アトム]
動詞の場合と同様。

3. 読み:〈読み〉[リスト(カタカナ表記)]
動詞の場合と同様。

[5]連体修飾するときの語尾の情報(なの:“な”または“の”を後接、な:“な”を後接、たる:“たる”を後接)。
[6]副詞として働くときに連体修飾されるかどうかの情報(単独:単独で副詞となる、被修飾:必ず修飾されて節を構成する、両方:いずれでもよい)

表 3.7: 形容詞の辞書項目

属性名	値の型、実現値	必須	マスタとの対応
id	アトム		語義指標 (見)
語彙	アトム	○	表層表現 (語)
読み	リスト (カタカナ表記)	○	読み (語)
語幹	[動詞と同様]	○	表層表現 (語)
活用	アトム [表 3.8]	○	活用型 (語)、表層表現 (語)、読み (語)
格	[動詞と同様]	○	能動表層格 (語)

4. 語幹: 〈語幹〉 [アトム]
   動詞の場合と同様。

5. 活用: 〈形容詞の活用型〉 [アトム]
   その形容詞の活用型。マスタ辞書の「活用型」「表層表現」「読み [7]」 (いずれも見出し語レコード) から作成する。マスタ辞書の「活用型」との対応は表 3.8の通りである。

6. 格: 〈格リスト〉 [リスト (演算子適用項)]
   その形容詞のとる格の情報。動詞の場合と同様。

表 3.8: マスタ辞書と文生成辞書の形容詞の活用型

マスタ辞書	文生成用辞書の活用型	
普通	イ系ア	語幹末尾がア段
	イ系イ	語幹末尾がイ段
	イ系ウ	語幹末尾がウ段
	イ系オ	語幹末尾がオ段
ナ型	イ系アナ	イ系アで連体形が“～な”
	イ系イナ	イ系イで連体形が“～な”
よいない	イ系サ	中立形が“～さ”

3.2.5 副詞の辞書項目

表 3.9に副詞の辞書項目一覧を掲げる。表中の「必須」において、“動”はサ変動詞用法があるとき必須属性であることを示す。

1. id: 〈語彙識別番号〉 [アトム]
   マスタ辞書の語義指標を記述する。副詞の場合，以下の通り。

   副詞$X_1X_2X_3X_4X_5X_6C$　　但し，X_i: 数字,C: アルファベット (多語義に対応)

2. 語彙: 〈語彙〉 [アトム]
   動詞の場合と同様。但し，“に”や“と”が後接して新たに作成される場合は，後接した形を記載する。

3. 読み: 〈読み〉 [リスト (カタカナ表記)]
   動詞の場合と同様。但し，“に”や“と”が後接して新たに作成される場合は，後接した場合を記載する。

[7] 形容詞の活用型を決めるには，語幹の“読み”の情報が必要である。

表 3.9: 副詞の辞書項目

属性名	値の型、実現値	必須	マスタとの対応
id	アトム		語義指標 (見)
語彙	アトム	○	表層表現 (見)
読み	リスト (カタカナ表記)	○	読み (見)
類別		○	細分類 (語)
にと	〈にと 2〉		にと (語)
格	[動詞と同様]	動	サ変動詞用法 (語)
受動	[動詞と同様]		サ変動詞用法 (語)
アスペクト分類	[動詞と同様]	動	サ変動詞用法 (語)

4. 類別: 〈類別コード〉 [アトム]
   マスタ辞書の「細分類」属性 (語義レコード) の値を記述する。

5. にと: 〈にと 2〉 [アトム]
   その副詞に“に”や“と”が後接して同語義の副詞になるかを示す情報。マスタ辞書の「にと」属性の値を記述する。但し,“に”や“と”が後接して新たに作成される場合は,‘単独’と記載する。

   〈にと 2〉 ::= ‘に’|‘と’|‘にと’|‘単独’

6. 格: 〈格リスト〉 [リスト (演算子適用項)]
   その副詞にサ変動詞用法がある場合の格情報が記述される。マスタ辞書の「サ変動詞用法」(語義レコード) の「能動表層格」属性の値を記述する。サ変動詞用法がない場合は記述されない。

7. 受動: 〈受動格リスト〉 [部分項]
   その副詞にサ変動詞用法がありかつ直接受動態が存在する場合に,サ変動詞としての受動格情報が記述される。マスタ辞書の「サ変動詞用法」(語義レコード) の「受動表層格」属性の値を記述する。それ以外の場合は記述されない。

8. アスペクト分類: 〈アスペクトリスト〉 [リスト (アトム)]
   その副詞にサ変動詞用法がある場合に,アスペクト情報が記述される。マスタ辞書の「サ変動詞用法」(語義レコード) の「相」属性の値を記述する。サ変動詞用法がない場合は記述されない。

3.2.6 連体詞の辞書項目

表 3.10に連体詞の辞書項目一覧を掲げる。

表 3.10: 連体詞の辞書項目

属性名	値の型、実現値	必須	マスタとの対応
id	アトム		語義指標 (見)
語彙	アトム	○	表層表現 (見)
読み	リスト (カタカナ表記)	○	読み (見)

1. id: 〈語彙識別番号〉 [アトム]
   マスタ辞書の語義指標を記述する。連体詞の場合,以下の通り。

   連体$X_1X_2X_3X_4X_5X_6C$　　但し,X_i: 数字,C: アルファベット (多語義に対応)

2. 語彙: 〈語彙〉 [アトム]
   動詞の場合と同様。

3. 読み: 〈読み〉 [リスト (カタカナ表記)]
   動詞の場合と同様。

3.2.7 助動詞の辞書項目

表 3.11に助動詞の辞書項目一覧を掲げる。

表 3.11: 助動詞の辞書項目

属性名	値の型、実現値	必須
語彙	アトム	○
読み	リスト (カタカナ表記)	○
語幹	アトム	○
活用	〈助動詞の活用型〉	○
接続	〈接続情報〉 [下記]	○
格	[動詞と同様]	
受動	[動詞と同様]	
アスペクト分類	[動詞と同様]	

1. 語彙: 〈語彙〉 [アトム]
   動詞と同様。

2. 読み: 〈読み〉 [リスト (カタカナ表記)]
   動詞と同様。

3. 語幹: 〈語幹〉 [アトム]
   助動詞の語幹。

4. 活用: 〈助動詞の活用型〉 [アトム]
   その助動詞の活用型。値としては, 〈動詞の活用型〉等[8]をとる。

5. 接続: 〈接続情報〉 [アトム, リスト (リスト)]
   接続可能な活用語の活用形を示す。活用語の活用型により接続形が違う場合は, 活用型ごとにリストで記述する。

   〈接続情報〉 ::= 活用形|[[活用型, 活用形],...]

6. 格: 〈格リスト〉 [リスト]
   その助動詞が付加されることで生じる格の情報。記述形式は動詞の場合と同様。そのような格が存在しない場合は記述されない。

7. 受動: 〈受動格リスト〉 [部分項]
   その助動詞が付加された語全体が受動化されるときの格の情報。記述形式は動詞の場合と同様。受動態が存在しないときは記述されない。

8. アスペクト分類: 〈アスペクトリスト〉 [リスト]
   その助動詞が付加されることで変化したアスペクト情報。記述形式は動詞の場合と同様。アスペクト情報が変化しない場合は記述しない。

[8] 助動詞の活用型としては、動詞の活用型の他に ' ダ系 ',' デアル ',' ナイ ' 等がある。詳細は活用表 (付録 B) を参照のこと。

3.2.8 取り立て詞, 接続助詞, 終助詞の辞書項目

表 3.12に取り立て詞, 接続助詞, 終助詞の辞書項目一覧を掲げる。

表 3.12: 取り立て詞、接続助詞、終助詞の辞書項目

属性名	内容、意味	値の型、実現値	必須
語彙	表層表現	アトム	○
読み	読みのリスト	リスト(カタカナ表記)	○
接続	前接する活用語の活用形	[助動詞と同様]	○

1. 語彙: 〈語彙〉 [アトム]
   動詞と同様。
2. 読み: 〈読み〉 [リスト(カタカナ表記)]
   動詞と同様。
3. 接続: 〈接続情報〉 [リスト(リスト)]
   助動詞と同様。

3.2.9 接続詞, 格助詞, 接尾語の辞書項目

表 3.13に接続詞, 格助詞, 接尾語の辞書項目一覧を掲げる。

表 3.13: 接続詞、格助詞、接尾語の辞書項目

属性名	内容、意味	値の型、実現値	必須
語彙	表層表現	アトム	○
読み	読みのリスト	リスト(カタカナ表記)	○

1. 語彙: 〈語彙〉 [アトム]
   動詞と同様。
2. 読み: 〈読み〉 [リスト(カタカナ表記)]
   動詞と同様。

3.2.10 辞書エントリの記述例

“覆う”

```
jg_dict(動詞,覆う,{id/動詞000470A,
                    語彙/覆う,
                    語幹/覆,
                    読み/[オオウ],
                    活用/ウ系強変化ワ行,
                    格/[:::(道具,が),:::(対象,を)],
                    受動/{対象/[:::(対象,が),:::(道具,に)]},
                    アスペクト分類/[準状態],
                    可能動詞化/可,
                    派生/[覆い]}
```

“人”

```
jg_dict(名詞, 人,{id/ 名普 003380A,
                  語彙 / 人,
                  読み /[ヒト],
                  類別 /[普通名詞],
                  複数形 /[畳語, 接尾辞 (ら), 接尾辞 (達)],
                  助数詞 /[人]})
```

“朝”

```
jg_dict(名詞, 朝,{id/ 名普 000050A,
                  語彙 / 朝,
                  読み /[アサ],
 類別 /[普通名詞, 連用名 (両)],
 複数形 /[]})
```

“いろいろ”

```
jg_dict(名詞, いろいろ,{id/000030B,
                        語彙 / いろいろ,
                        読み /[イロイロ],
                        類別 /[形動名 (なの), 連用名 (単)],
                        にと / と,
                        複数形 /[],
                        格 /[:::(記述対象, が)]})
```

“運転”

```
jg_dict(名詞, 運転,{id/ 名サ 000010A,
                    語彙 / 運転,
                    読み /[ウンテン],
                    類別 /[動作名],
                    複数形 /□,
                    格 /[:::(行為者, が),:::(対象, を)],
                    受動 /{対象 /[:::(対象, が),:::(行為者,[に, によって])]},
                    アスペクト分類 /[継続]})
```

“痛い”

```
jg_dict(形容詞, 痛い,{id/ 形容 000070A,
                      語彙 / 痛い,
                      語幹 / 痛,
                      読み /[イタイ],
                      活用 / イ系ア,
                      格 /[:::(経験者, が),:::(対象, が)]})
```

“うまい具合”

```
jg_dict(副詞, うまい具合,{id/ 副詞 000110A,
                          語彙 / うまい具合,
                          読み /[ウマイグアイ],
                          にと / に,
                          類別 /1})
```

“うまい具合に”

```
jg_dict(副詞, うまい具合に,{id/ 副詞 000110A,
                              語彙 / うまい具合に,
                              読み /[ウマイグアイニ],
                              にと / 単独,
                              類別 /1})
```

“うっとり”

```
jg_dict(副詞, うっとり,{id/ 副詞 000100A,
                         語彙 / うっとり,
                         読み /[ウットリ],
                         類別 /1,
                         にと / と,
                         格 /[:::(経験者, が)],
                         アスペクト分類 /[準状態]})
```

“ある”

```
jg_dict(連体詞, ある,{id/ 連体 000020A,
                       語彙 / ある,
                       読み /[アル]})
```

“ない”

```
jg_dict(助動詞, ない,{語彙 / ない,
                      読み /[ナイ],
                      語幹 / な,
                      活用 / ナイ,
                      接続 /[[ダ系, 連用形],[ウ系サ変, 中立形],[_, 連接形]],
                      格 /[:::(対象, が)]})
```

“させる”

```
jg_dict(助動詞, させる,{語彙 / させる,
                        読み /[サセル],
                        語幹 / させ,
                        活用 / ウ系弱変化,
                        接続 / 連接形,
                        格 /[:::(行為者, が),:::(内容,'')],
                        受動 /{内容 /[:::(内容, が),:::(行為者, によって)]},
                        アスペクト分類 /[瞬間]})
```

“いる”

```
jg_dict(助動詞, いる,{語彙 / いる,
                      読み /[イル],
                      語幹 / い,
                      活用 / ウ系イル,
                      接続 / 連用形,
                      アスペクト分類 /[状態]})
```

“も”

```
jg_dict(取り立て詞, も,{語彙 / も,
                        読み /[モ],
                        接続 /[[ダ系, 中立系],[_, 現在形]]})
```

“しかし”

```
jg_dict(接続し，しかし,{語彙／しかし,
                        読み／[シカシ]})
```

3.3 形態素生成用テーブル類作成手順

形態素生成用テーブル類は、ファイル morph_kat_table.cil に活用表を格納し、ファイル morph_table.cil に、接続表やその他の形態素生成エンジンで用いるテーブル類を格納する。

1. 活用表
   活用表は、個々の活用型毎に

   活用表 (<活用型>,<活用データ>)

   という述語として記述する。以下に活用表作成時の注意点を列挙する。

   - 活用型は、生成用辞書の活用属性に与えられた値でなければならない。
   - 活用データは、活用形を属性名、その語尾を属性値とする CIL の部分項によって表す。
   - 活用語尾が存在しない活用形は、属性の記述を行なわない。
   - 語尾が空の場合は、値として「''」を与える。
   - 語幹の変更を伴う場合には、実際の語尾の前に「*」を付ける。さらに語幹変化表に変化規則を登録する。

2. 接続表
   接続表は、

   接続表 (<活用型>,<係り先の語の語彙データ>,<活用形>)

   という述語として記述する。以下に接続表作成時の注意点を列挙する。

   - 活用型は、生成用辞書の活用属性に与えられた値でなければならない。
   - 活用形は活用表データの属性として記述のあるものでなければならない。

3. 語幹変化表
   語幹変化表は、

   語幹変化表 (<語彙 1>,<語彙 2>)

   という述語として記述する。変更すべき語幹の実際に変更される部分を語彙 1 に置き換える表現を語彙 2 にそれぞれ漢字表現のストリングとして与える。

付録 A

活用表以外の形態素生成用テーブル類

A.1 接続表

```
接続表(_,[格助詞,Dterm],現在形) :- !.
接続表(_,[接辞,Dterm],現在形) :- !.
接続表(イ系イナ,[名詞,Dterm],連体形) :- !. % for 大きい
接続表(イ系アナ,[名詞,Dterm],連体形(な)) :- !. % for 小さい
接続表(ダ系,[名詞,Dterm],連体形) :- !.
接続表(_,[名詞,Dterm],現在形) :- !.
接続表(_,[助辞,れば],現在仮定形) :- !.
接続表(_,[助辞,たら],完了仮定形) :- !.
接続表(_,[助辞,う],現在推量形) :- !.
接続表(_,[助辞,たろう],完了推量形) :- !.
接続表(_,[助辞,まい],否定意志形) :- !.
接続表(_,[助辞,た],完了形) :- !.
接続表(_,[助辞,て],連用形) :- !.
接続表(_,[助辞,たり],連用形(たり)) :- !.
接続表(ナイ,[ind,and],連接形) :- !.
接続表(Ktype,[ind,and],連接形) :-
    jgmor_subname(Ktype,イ系),!.
接続表(ダ系,[ind,and],連用形) :- !.
接続表(ウ系イル,[ind,and],連用形) :- !.
接続表(_,[ind,and],中立形) :- !.
接続表(_,[ind,IND],IND) :- !.
接続表(ナイ,[symbol,、],連接形) :- !.
接続表(Ktype,[symbol,、],連接形) :-
    jgmor_subname(Ktype,イ系),!.
接続表(ウ系イル,[symbol,、],連用形) :- !.
接続表(ダ系,[symbol,、],連用形) :- !.
接続表(_,[symbol,、],中立形) :- !.
接続表(ナイ,[symbol,',  '],連接形) :- !.
接続表(Ktype,[symbol,',  '],連接形) :-
    jgmor_subname(Ktype,イ系),!.
接続表(ウ系イル,[symbol,',  '],連用形) :- !.
接続表(ダ系,[symbol,',  '],連用形) :- !.
接続表(_,[symbol,',  '],中立形) :- !.
接続表(K,[symbol,Sym],現在形) :- !.
接続表(Ktype,[Cat,Dterm],Stype) :-
    (Cat=接続助詞;Cat=助動詞;Cat=終助詞;Cat=取り立て詞;Cat=準用詞),
    getRole(Dterm,接続,St1),
```

```
    jgmor_select(Ktype,St1,Stype),!.
接続表(Ktype,[A,B],Stype) :-
    活用語(A),!,
    ((jgmor_subname(Ktype, イ系);Ktype=ダ系;Ktype=ナイ),!,Stype=連接形;
     jgmor_subname(Ktype, ウ系),Stype=連用形).
接続表(_,[string,HYOSO],現在形) :-!.
```

A.2 語幹変化表

```
語幹変化表(さ, そ).
語幹変化表(た, と).
語幹変化表(か, こ).
語幹変化表(な, の).
```

A.3 せるれる変換表

```
せるれる変換表(せ,Goi1,Goi2) :- jg_append_word(さ,Goi1,Goi2).
せるれる変換表(れ,Goi1,Goi2) :- jg_append_word(ら,Goi1,Goi2).
```

付録 B

活用表

活用表の全容を以下に示す。

活用表（ウ系強変化カ行，
　{現在形 / く，
　完了形 / いた，
　現在推量形 / こう，
　完了推量形 / いたろう，
　否定意志形 / くまい，
　命令形 / け，
　現在仮定形 / けば，
　完了仮定形 / いたら，
　連用形 / いて，
　連用形（たり）/ いたり，
　中立形 / き，
　連接形 / か}）．

活用表（ウ系強変化ガ行，
　{現在形 / ぐ，
　完了形 / いだ，
　現在推量形 / どう，
　完了推量形 / いだろう，
　否定意志形 / ぐまい，
　命令形 / げ，
　現在仮定形 / げば，
　完了仮定形 / いだら，
　連用形 / いて，
　連用形（たり）/ いだり，
　中立形 / ぎ，
　連接形 / が}）．

活用表（ウ系強変化サ行，
　{現在形 / す，
　完了形 / した，
　現在推量形 / そう，
　完了推量形 / したろう，
　否定意志形 / すまい，
　命令形 / せ，
　現在仮定形 / せば，
　完了仮定形 / したら，
　連用形 / して，

連用形（たり）/ したり，
中立形 / し，
連接形 / さ}）.

活用表（ウ系強変化タ行，
{現在形 / つ，
完了形 / った，
現在推量形 / とう，
完了推量形 / ったろう，
否定意志形 / つまい，
命令形 / て，
現在仮定形 / てば，
完了仮定形 / ったら，
連用形 / って，
連用形（たり）/ ったり，
中立形 / ち，
連接形 / た}）.

活用表（ウ系強変化ナ行，
{現在形 / ぬ，
完了形 / んだ，
現在推量形 / のう，
完了推量形 / んだろう，
否定意志形 / ぬまい，
命令形 / ね，
現在仮定形 / ねば，
完了仮定形 / んだら，
連用形 / んで，
連用形（たり）/ んだり，
中立形 / に，
連接形 / な}）.

活用表（ウ系強変化バ行，
{現在形 / ぶ，
完了形 / んだ，
現在推量形 / ぼう，
完了推量形 / んだろう，
否定意志形 / ぶまい，
命令形 / べ，
現在仮定形 / べば，
完了仮定形 / んだら，
連用形 / んで，
連用形（たり）/ んだり，
中立形 / び，
連接形 / ば}）.

活用表（ウ系強変化マ行，
{現在形 / む，
完了形 / んだ，
現在推量形 / もう，
完了推量形 / んだろう，
否定意志形 / むまい，
命令形 / め，

現在仮定形 / めば,
完了仮定形 / んだら,
連用形 / んで,
連用形（たり）/ んだり,
中立形 / み,
連接形 / ま}).

活用表（ウ系強変化ラ行,
{現在形 / る,
完了形 /った,
現在推量形 / ろう,
完了推量形 /ったろう,
否定意志形 / るまい,
命令形 / れ,
現在仮定形 / れば,
完了仮定形 /ったら,
連用形 /って,
連用形（たり）/ったり,
中立形 / り,
連接形 / ら}).

活用表（ウ系ナサル,
{現在形 / る,
完了形 /った,
現在推量形 / ろう,
完了推量形 /ったろう,
否定意志形 / るまい,
命令形 / い,
命令形（れ）/ れ,
現在仮定形 / れば,
完了仮定形 /ったら,
連用形 /って,
連用形（たり）/ったり,
中立形 / り,
連接形 / ら}).

活用表（ウ系強変化ワ行,
{現在形 / う,
完了形 /った,
現在推量形 / おう,
完了推量形 /ったろう,
否定意志形 / うまい,
命令形 / え,
現在仮定形 / えば,
完了仮定形 /ったら,
連用形 /って,
連用形（たり）/ったり,
中立形 / い,
連接形 / わ}).

活用表（ウ系弱変化,
{現在形 / る,
完了形 / た,

現在推量形 / よう，
完了推量形 / たろう，
否定意志形 / まい，
命令形 / ろ，
命令形（よ）/ よ，
現在仮定形 / れば，
完了仮定形 / たら，
連用形 / て，
連用形（たり）/ たり，
中立形 / ''，
連接形 / ''}).

活用表（ウ系イル，
{現在形 / る，
完了形 / た，
現在推量形 / よう，
完了推量形 / たろう，
否定意志形 / まい，
命令形 / ろ，
命令形（よ）/ よ，
現在仮定形 / れば，
完了仮定形 / たら，
連用形 / て，
連用形（たり）/ たり，
中立形 / ''，
連接形 / ''}).

活用表（ウ系アル，
{現在形 / る，
完了形 / った，
現在推量形 / ろう，
完了推量形 / ったろう，
否定意志形 / るまい，
命令形 / れ，
現在仮定形 / れば，
完了仮定形 / ったら，
連用形 / って，
連用形（たり）/ ったり，
中立形 / り}).

活用表（ウ系サ変，
{現在形 / する，
完了形 / した，
現在推量形 / しよう，
完了推量形 / したろう，
否定意志形 / するまい，
否定意志形（縮）/ すまい，
命令形 / しよ，
命令形（よ）/ せよ，
現在仮定形 / すれば，
完了仮定形 / したら，
連用形 / して，
連用形（たり）/ したり，

中立形 / し,
連接形 / さ}).

活用表(ウ系サ変ザ行,
{現在形 / ずる,
完了形 / じた,
現在推量形 / じよう,
完了推量形 / じたろう,
否定意志形 / ずるまい,
命令形 / ぜよ,
現在仮定形 / ずれば,
完了仮定形 / じたら,
連用形 / じて,
連用形(たり)/ じたり,
中立形 / じ,
連接形 / ざ}).

活用表(ウ系カ変,
{現在形 / くる,
完了形 / きた,
現在推量形 / こよう,
完了推量形 / きたろう,
否定意志形 / くるまい,
命令形 / こい,
現在仮定形 / くれば,
完了仮定形 / きたら,
連用形 / きて,
連用形(たり)/ きたり,
中立形 / き,
連接形 / こ}).

活用表(ウ系カ変来る,
{現在形 / 来る,
完了形 / 来た,
現在推量形 / 来よう,
完了推量形 / 来たろう,
否定意志形 / 来るまい,
命令形 / 来い,
現在仮定形 / 来れば,
完了仮定形 / 来たら,
連用形 / 来て,
連用形(たり)/ 来たり,
中立形 / 来,
連接形 / 来}).

活用表(ウ系マス,
{現在形 / す,
完了形 / した,
現在推量形 / しよう,
完了推量形 / したろう,
否定意志形 / すまい,
命令形 / せ,
現在仮定形 / すれば,

完了仮定形 / したら，
連用形 / して，
連用形（たり）/ したり}）．

活用表（ウ系強変化イク，
{現在形 / く，
完了形 / った，
現在推量形 / こう，
完了推量形 / ったろう，
否定意志形 / くまい，
命令形 / け，
現在仮定形 / けば，
完了仮定形 / ったら，
連用形 / って，
連用形（たり）/ ったり，
中立形 / き，
連接形 / か}）．

活用表（イ系ア，
{現在形 / い，
完了形 / かった，
現在推量形 / かろう，
完了推量形 / かったろう，
現在仮定形 / ければ，
完了仮定形 / かったら，
連体形 / き，
連用形 / くて，
連用形（たり）/ かったり，
中立形 / ''，
派生中立形 /（* う），
連接形 / く}）．

活用表（イ系ナイ，
{現在形 / い，
完了形 / かった，
現在推量形 / かろう，
完了推量形 / かったろう，
現在仮定形 / ければ，
完了仮定形 / かったら，
連体形 / き，
連用形 / くて，
連用形（たり）/ かったり，
中立形 / さ，
派生中立形 /（* う），
連接形 / く}）．

活用表（イ系アナ，
{現在形 / い，
完了形 / かった，
現在推量形 / かろう，
完了推量形 / かったろう，
現在仮定形 / ければ，
完了仮定形 / かったら，

連体形 / き,
連体形 (な)/ な,
連用形 / くて,
連用形 (たり)/ かったり,
中立形 /'',
派生中立形 /(* う),
連接形 / く}).

活用表 (イ系イ,
{現在形 / い,
完了形 / かった,
現在推量形 / かろう,
完了推量形 / かったろう,
現在仮定形 / ければ,
完了仮定形 / かったら,
連体形 / き,
連用形 / くて,
連用形 (たり)/ かったり,
中立形 /'',
派生中立形 / ゅう,
連接形 / く}).

活用表 (イ系イナ,
{現在形 / い,
完了形 / かった,
現在推量形 / かろう,
完了推量形 / かったろう,
現在仮定形 / ければ,
完了仮定形 / かったら,
連体形 / な,
連用形 / くて,
連用形 (たり)/ かったり,
中立形 /'',
派生中立形 / ゅう,
連接形 / く}).

活用表 (イ系ウ,
{現在形 / い,
完了形 / かった,
現在推量形 / かろう,
完了推量形 / かったろう,
現在仮定形 / ければ,
完了仮定形 / かったら,
連体形 / き,
連用形 / くて,
連用形 (たり)/ かったり,
中立形 /'',
派生中立形 / う,
連接形 / く}).

活用表 (イ系オ,
{現在形 / い,
完了形 / かった,

現在推量形 / かろう,
完了推量形 / かったろう,
現在仮定形 / ければ,
完了仮定形 / かったら,
連体形 / き,
連用形 / くて,
連用形(たり)/ かったり,
中立形 / '',
派生中立形 / う,
連接形 / く}).

活用表(イ系サ,
{現在形 / い,
完了形 / かった,
現在推量形 / かろう,
完了推量形 / かったろう,
現在仮定形 / ければ,
完了仮定形 / かったら,
連体形 / き,
連用形 / くて,
連用形(たり)/ かったり,
中立形 / さ,
派生中立形 / う,
連接形 / く}).

活用表(ナイ,
{現在形 / い,
完了形 / かった,
現在推量形 / かろう,
完了推量形 / かったろう,
現在仮定形 / ければ,
完了仮定形 / かったら,
連用形 / いで,
連用形(たり)/ かったり,
派生中立形 /(* う),
中立形 / '',
連接形 / く}).

活用表(ダ系,
{現在形 / だ,
完了形 / だった,
現在推量形 / だろう,
完了推量形 / だったろう,
現在仮定形 / ならば,
現在仮定形(なら)/ なら,
完了仮定形 / だったら,
連体形 / な,
連体形(の)/ の,
連用形 / で,
連用形(たり)/ だったり,
中立形 / '',
連接形 / に}).

活用表（デス，
　　{現在形 / です，
　　完了形 / でした，
　　現在推量形 / でしょう，
　　完了推量形 / だったでしょう，
　　連用形 / でして}）．

活用表（デアル，
　　{現在形 / である，
　　完了形 / であった，
　　現在推量形 / であろう，
　　完了推量形 / であったろう，
　　否定意志形 / であるまい，
　　現在仮定形 / であれば，
　　現在仮定形（あれ）/ であれ，
　　完了仮定形 / であったら，
　　連用形 / であって，
　　連用形（たり）/ であったり，
　　中立形 / であり}）．

活用表（ヌ，
　　{現在形 / ぬ，
　　現在仮定形 / ねば，
　　完了仮定形 / なかったら，
　　連体形 / ぬ，
　　連用形 / ないで，
　　連用形（たり）/ なかったり，
　　中立形 / ず}）．

活用表（ン，
　　{現在形 / ん}）．

付録 C

マクロ展開

C.1 マクロ展開で用いるの情報の属性名と属性値

属性名	属性の表す内容
'*pos'	品詞
'*dterm'	辞書項目
'*case'	辞書の格リスト
'*last'	末尾の語彙データ。丁寧化で用いる
'*nano'	連体修飾における「な」「の」の指示
'*pdt'	ロールの親 (関係) の辞書項目
'*setsuji'	ロールの接辞

C.2 ユーザによるマクロ展開規則の調整

ユーザによるマクロ展開規則の調整を行う場合、具体的にどの述語を定義すればよいかをまとめる。

C.2.1 標準マクロ定義に対する属性の追加

1. type1(関係、ロール属性を持ち文に対応する) の中間表現に対する属性の追加。
   新たに追加する属性を展開する述語を定義して、jg_macro_expand/2 から実行する。

2. type2(被修飾、連体修飾属性を持ち名詞句に対応する) の中間表現に対する属性の追加。
   新たに追加する属性を展開する述語を定義して、jg_rentai/2 から実行する。ただし、type1、type3 でも使用する属性は jg_macro_expand/2 から実行する。

3. type3(語彙属性を持ち語に対応する) の中間表現に対する属性の追加。
   新たに追加する属性を展開する述語を定義して、jg_goi/2 から実行する。ただし、type1、type2 でも使用する属性は jg_macro_expand/2 から実行する。

4. ある特定の属性と同時にしか使われないような属性を追加する。
   例えば、ヴォイス属性と同時にしか使用しない属性を追加する場合は、新たに追加する属性を展開する述語を定義して、jg_voice/2 から実行し、ヴォイス属性と同時に処理する。標準マクロ定義における数属性がこれに当たる。また、ロール成分を表す中間表現でのみ用いられる属性の展開は jg_role/2 から実行する。

C.2.2 語属性の展開の調整

1. 語彙属性の展開時の辞書引き
   jg_macro_dict/3 が生成用辞書とのインタフェースを取っている。 jg_macro_dict/3 の定義を変更し、新たな属性への対応、辞書引きが失敗した時の処理などを行うことができる。

2. 語彙に対する数の処理
   jg_kazu/2 の複数形、助数詞の生成方法を調整する。

C.2.3 関係属性の展開の調整

1. 動詞化の調整
   関係属性の値は jg_doushika/2 によって動詞化される。標準マクロ定義では述語性の名詞はすべて動詞化する。jg_doushika/2 の動詞化条件を変えることができる。

2. 関係属性が無い場合の処理
   標準マクロ定義では関係属性が無い場合は「である」を仮定して、「である」の格リストを用いる。jg_kankei/2 の関係属性が無い場合の処理を変更できる。

C.2.4 派生属性の展開の調整

1. 派生辞の付加方法の調整
   標準マクロ定義では、ほぼ派生属性の値ごとに派生辞の処理規則を記述している。例えば派生属性の値「よう」に対して、名詞句につく場合は「のようだ」を派生辞とし、それ以外は「ようだ」を派生辞とする。jg_hasei2/2 の派生辞の付加方法を調整することができる。

2. 派生辞の追加
   同様に、新たな派生辞を追加することができる。jg_hasei2/2 の最後の定義でデフォルトの処理を行っているので、その前に新たに定義した jg_hasei2/2 を挿入する。

C.2.5 連体修飾属性の展開の調整

1. 修飾句の活用の調整
   標準マクロ定義では連体修飾句が名詞の場合は「の」を形容動詞性名詞の場合は '*nano' 属性の指示にしたがって「な」または「の」を付加する。 jg_rentai/2 の定義を変更することによって、修飾句の活用の調整ができる。

2. 連体修飾句の語順の調整
   連体修飾句が set() で与えられた時、連体語順表 /2 を用いて修飾句の語順を決定する。連体語順表 /2 を変更することにより連体修飾句の語順を調整することができる。

C.2.6 ヴォイス属性の展開の調整

1. 可能動詞化の調整
   標準マクロ定義では可能態にする場合に可能動詞化を行うか、または「れる」を付加する。可能動詞化を行うかは辞書を参照して決める。 jg_get_voice/3 の定義を変更することにより、可能動詞化の処理を調整することができる。

2. 態変化による助詞変換の調整
   態変化による助詞変換は get_change_case_list/4 によって得られる助詞変換リストに基づいて行う。get_change_case_list/ の助詞変換リスト生成規則を変更することにより助詞変換の調整ができる。

C.2.7 ロール属性の展開の調整

1. ロールの接辞決定方法の調整
   ロールの接辞は jg_get_setsuji/2 によって決定する。jg_get_setsuji/2 はまず辞書を参照し次にロール /4 によって接辞を決定する。ロール /4 を変更することによって接辞の調整ができる。まったく辞書を参照しない場合は jg_get_setsuji/2 を変更する。

2. ロール成分における表層属性の調整
   標準マクロ定義では、ロール成分に表層属性がある場合はその値を接辞として用いる。jg_hogo/2 を変更することによってロール成分の表層属性の使用法を調整できる。

C.2.8 ロールの語順の調整

1. 語順決定方法の調整
   語順の決定方法は make_gojun_list/2 によって決められる。語順属性の値を用いるか、語順規則を用いるかを調整できる。

2. 語順規則の調整
   語順規則は gojun/2 を変更することによって調整できる。また、ロール名によるロール間の順位を調整する場合は、格分類表 /2 の第二引数の値を変える。

C.2.9 認め方属性の展開の調整

1. 否定に用いる助動詞の調整
   認め方属性の値が「否定」の時、助動詞の「ない」「ぬ」を付加する。否定の助動詞の変更は get_mitomekata/2 の変更によって行う。

C.2.10 名詞化の調整

1. 名詞化の条件の調整
   名詞化の条件は、jg_meishika/2 を変更することによって調整できる。jg_meishika/2 はロール成分の名詞化を行う際に jg_nom/4 を実行する。ロール成分の名詞化条件は jg_nom/4 を変更することによって調整できる。

C.2.11 モーダル属性の展開の調整

1. ムード属性の展開の調整
   ムード属性の値に応じて、jg_get_mood/2 によってムードの表層構造を生成する。jg_get_mood/2 を変更することによってムードの調整ができる。

2. 態度属性の展開の調整
   態度属性の値に応じて、jg_get_taido/2 によって態度の表層構造を生成する。jg_get_taido/2 を変更することによって態度の調整ができる。

C.2.12 丁寧属性の展開の調整

1. 丁寧化の方法の調整
   丁寧化の方法は p_conv/3 を変更することによって調整できる。

C.2.13 記号属性の展開の調整

1. 記号の種類の調整
   記号の種類は、jg_get_symbol/2 を変更することによって調整できる。

参考文献

[1] T. Ikeda et al. Sentence generation in LTB (*in Japanese*). In *5th Conference Proceedings of Japan Software Science and Technology*, 1988.

[2] T. Chikayama. ESP Reference Manual. Technical Report 044, ICOT, 1984.

[3] A. Colmerauer. Prolog-II: Reference Manual and Theoretical Model. Internal report, Group Intelligence Artificielle, Université d'Aix-Marseille II, 1982.

[4] ICOT. PSI/SIMPOS Editor Manual. 1988.

[5] T. Miyazaki and K. Taki. Multi-PSI ni okeru Flat GHC no Jitsugen Hôshiki (Installation of Flat GHC on Multi-PSI (*in Japanese*)). Technical Report 190, ICOT, 1986.

[6] K. Ueda. Guarded Horn Clauses. Technical Report 103, ICOT, 1985.

[7]

[8] J. Barwise. Recent Developments in Situation Semantics. In M. Nagao, editor, *Language and Artificial Intelligence*, pages 387–399, Amsterdam, 1987. North-Holland.

[9] J. Barwise and J. Perry. *Situations and Attitudes*. MIT Press, Cambridge, 1983.

[10] M. Dincbas, H. Simonis, and P.V. Hentenryck. Extending equation solving and constraint handling in logic programming. Internal Report IR-LP-2203, ECRC, 1987.

[11] T. Amanuma, T. Suzuki, T. Okunishi, and K. Mukai. CIL Programming kankyô (CIL Programming Environment (*in Japanese*)). In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*, pages 211–214, 1988.

[12] L. Cardelli. Typechecking Dependent Types and Subtypes. Technical report, DEC Systems Research Center, 1987.

[13] K. Mukai. Horn Clause Logic with Parameterized Types for Situation Semantics Programming. Technical Report 101, ICOT, 1985.

[14] K. Mukai. Unification over Complex Indterminates in Prolog. Technical Report 113, ICOT, 1985.

[15] K. Mukai. Anadic Tuples in Prolog. Technical Report 239, ICOT, 1987.

[16] K. Mukai. A System of Logic Programming for Linguistic Analysis based on Situation Semantics. In *Proceedings of the workshop on semantic issues in human and computer languages*. CSLI, 1987.

[17] K. Mukai. Partially Specified Term in Logic Programming for Linguistic Analysis. In *Proceedings of the International Conference on FGCS '88*, 1988.

[18] K. Mukai and H. Yasukawa. *Complex Indeterminates in Prolog and its Application to Discourse Models*, volume 3, pages 441–466. OHMUSHA, ltd. and Spring Verlag, 1985.

[19] J. Reynolds. Three approaches to type structures. *Lecture Note in Computer Science*, volume 185, Springer Verlag, 1985.

[20] K. Sakai and Y. Sato. Boolean Gröbner bases. Technical Memo 488, ICOT, 1988.

和文索引

英文索引