TM-0755

LTB機能説明資料

田中裕一, 久保幸弘, 福本文代,
福島秀顕, 萩原　葦, 天沼敏幸(三菱電機),
妙泉正隆(JIPDEC)

July, 1989

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

資料 2

LTB 機能説明資料

1989 年 7 月 5 日

ICOT 第二研究室

目次

I はじめに 3

II CIL 4

1 CILの概要 4
1.1 言語機能 4
1.2 プログラミング環境 4

2 CIL言語機能 5
2.1 部分項 5
2.1.1 部分項の概念 5
2.1.2 部分項と拡張単一化 6
2.1.3 部分項用シンタックス・シュガー 6
2.2 遅延実行制御 7
2.2.1 遅延実行制御機能 7
2.2.2 遅延実行制御用シンタックス・シュガー 7
2.2.3 制約記述 8
2.3 マクロ機能 8
2.3.1 ESP マクロ機能 8
2.3.2 ユーザ・シンタックス・シュガー機能 8

3 CILプログラミング環境 8
3.1 コマンド・インタープリタ 8
3.2 ユニット機能とファイル操作機能 8
3.3 シンタックス・チェッカ付のテキスト・エディタ 9
3.4 ソース・プログラムを即座に実行できるインタプリタ 9
3.5 スクリーン・イメージで行うデバッグ支援環境 9
3.6 プログラムの実行を高速化する最適化コンパイラ 10
3.7 部分項、デーモン付変数を調べるインスペクタ 10
3.8 ツールボックス機能 10

III 形態素解析システム LAX 11

4 概要 11

5 LAXの特徴 11
5.1 特徴 11
5.2 形態素辞書記述形式 11
5.2.1 形態素文法と辞書 11
5.2.2 記述形式 12
5.2.3 状態遷移表と接続のセマンティクス 12

6 ツールの操作法 13
6.1 形態素辞書開発環境 13
6.2 LAX トップ・ウインドウ 14
6.3 辞書エディタ 14
6.4 インスペクタ 14
6.5 プリティ・プリンタ 15

7 おわりに 15

IV 構文解析システム SAX 20

8 SAX の概要 20
8.1 SAX とは 20
8.2 解析アルゴリズム 20
8.3 システム構成 20

9 文法の記述形式 21
9.1 拡張 21
9.2 制限 23

10 文法開発支援ツール 23

V 文生成部 28

11 概要 28

12 文生成部の機能 28
12.1 文生成機構 28
12.1.1 中間表現 (基本表現、標準マクロ表現) 28
12.1.2 表層構造 29
12.2 文生成用ツール 29
12.2.1 編集 29
12.2.2 入出力先の指定 29
12.2.3 実行制御 29
12.2.4 その他 29

13 文生成機構の構成 30
13.1 マクロ展開部 30
13.2 形態素生成部 30
13.3 文生成用辞書 30

VI マスタ辞書 36

14 概要 36

15 構成と内容 36

16 アクセスメソッド 40

VII LTB−shell 42

17 LTB−shell とは 42

18 LTB−shell の特徴 42

19 LTB−shell の利用方法 43
19.1 コマンドの起動方法 43
19.2 コマンドの入出力の指定方法 43
19.2.1 リダイレクション 44
19.2.2 パイプ 44
19.3 ジョブの管理 44
19.4 その他の機能 44
19.5 LTB−shell の操作環境 44
19.6 SIMPOS シェル からの LTB の利用法 45
19.6.1 ツールの修正 45
19.6.2 login.com の修正 45

第 I 部
はじめに

ICOT 第二研究室では、自然言語処理の研究開発を行ってまいりましたが、このたび、研究開発の過程で整備してきた開発環境を汎用日本語処理系 (Language Tool Box 略称 LTB)1.1 版としてまとめ、広くユーザにリリースすることになりました。

本資料は、LTB を構成するツールの概要を取りまとめたものです[1]。

1. **CIL**
   CIL は、自然言語処理にあらわれる複雑な構造を持つオブジェクトとオブジェクトの間の制約関係をより柔軟に記述するために Prolog を拡張した言語です。本資料は CIL の言語機能, プログラミング環境について述べています。

2. **形態素解析システム LAX**
   LAX は、ユーザが与える形態素辞書に従って形態素解析を行うシステムです。本資料は、LAX の特徴, 形態素辞書の記述形式, その開発環境について述べています。

3. **構文解析システム SAX**
   SAX は、ユーザが与える文法に従って構文解析を行うシステムです。本資料は、文法の記述形式, その開発環境について述べています。

4. **文生成**
   文生成は、ユーザの与える文の統語情報から表層表現を得るシステムです。本資料は、統語情報の記述形式, その開発環境について述べています。

5. **マスタ辞書**
   マスタ辞書には、LAX,SAX, 文生成が使用する単語の形態的情報と構文的情報が記載されています。本資料は、マスタ辞書の構成, アクセス方法について述べています。

6. **LTB シェル**
   LTB シェルは、LTB ツール群や、ユーザによって作成されるツール群を効率よく組み合わせて使うためのユーザ・インタフェースを提供するものです。本資料は、LTB シェルの実現構想について述べています。

本資料の目的は、LTB を構成するツールを簡単に紹介し、機能、使い方を説明することによって、多くの人に LTB を使ってもらうきっかけを作ることです。したがって、LTB のすべての機能を説明しているわけではありません。本資料を一通り読んだ後、LTB を使ってみようと思う人は、「LTB 操作説明書」を参照して下さい。「LTB 操作説明書」には、LTB を構成するツールの詳しい機能、操作方法が書かれています。

[1] LTB シェルについては、LTB2 版以降で実現する予定の構想を中心に述べています。

第 II 部
CIL

1 CIL の概要

1.1 言語機能

CIL (Complex Indeterminates based Language) は、自然言語処理にあらわれる複雑な構造を持つオブジェクトとオブジェクト間の制約関係をより柔軟に記述するために Prolog を拡張した言語です。本言語によって、GPSG, LFG 等の構文論および、状況意味論 (situation semantics) を用いた自然言語処理システムを記述することを目的としています。

CIL は論理型言語 Prolog をベースとして、新たに二つの基本機能を導入しました。

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

言語仕様の面では、これらの基本機能を容易に記述するためのシンタックス・シュガー（標準マクロ）と組込述語を提供しています。

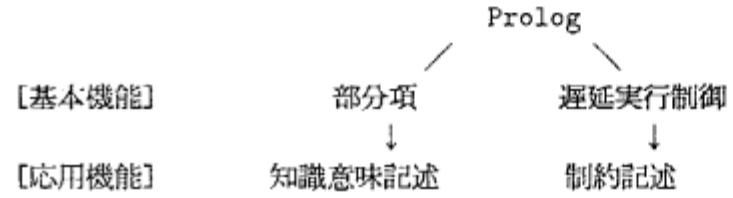


図 1: CIL の言語機能

1.2 プログラミング環境

CIL プログラミング環境は PSI・SIMPOS 上での、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ます。

CIL システムはコマンド・インタプリタを介して行う対話型のシステムであり、その操作仕様は、マルチウインドウ機能とマウスを使用した操作仕様となっています。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た操作性を実現しています。

CIL プログラミング環境のユーザー・インターフェイスとしては、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, インスペクタが提供されています。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ず、コマンド・インタプリタを介して利用します。

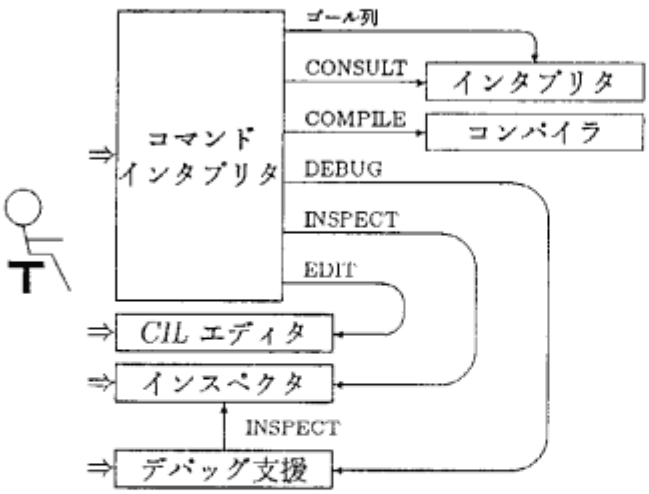


図 2: CIL のプログラミング環境

2 CIL 言語機能

2.1 部分項

2.1.1 部分項の概念

部分項は、Minsky のフレーム (Frame), GPSG のカテゴリ, LFG の機能構造 (functional structure), 状況理論における割り当て (assignment) や事態 (state of affair) などの概念や計算機言語やデータベース、計算機工学で広く見られるデータ構造を、抽象化し、一階述語論理の項の拡張として実現したものです。

「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」という面を強調したものです。部分項で表現されたデータ構造は、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ません。そのために、部分項は本質的に拡張可能なデータ構造であるという性質を持っています。

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、具体的には、次の形式で記述します。

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

a_i はラベルを表し、b_i は値を表します。そして、a_i/b_i の出現順序には意味がありません。a_i には通常の項が、b_i には部分項も含めた項が記述できます。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ります。

[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 } }

2.1.2 部分項と拡張単一化

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 を用いて行われます。部分項機能をプログラミングの際に充分に利用するための組込述語や記法を提供しています。

また、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ます。CIL の拡張された単一化の実質的意味は併合 (merge) であり、Prolog の単一化の仕様を拡張することで実現されています。

1. 部分項が関与しない単一化（通常項同士の単一化）
   Prolog の単一化と同じです。

```
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します。未定義変数は相手の部分項と同じになります。

```
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します。

```
{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が行われます。

```
{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が、新たに、他方の部分項に追加されます。

```
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}
```

2.1.3 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，以下の諸記法を提供しています。

1. 拡張単一化
   部分項を含む項の単一化は， CIL 組込述語の unify_cil/2 に展開されます。
   ［例］ X = Y

2. 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 と書きます。 P!L は、組込述語 role(L,P,V) に展開され、 V の値を持ちます。
［例］
・ {a/1, b/X}!b = 3. は、 X = 3 となります。
・ X = {a/1, b/2}, X!c = 3. は、 X = {a/1, b/2, c/3} となります。

3. 条件付項
「条件 C を満たす X」を X:C と書きます。 条件 C として述語が記述できます。
［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」という意味です。

4. 同格記法
X#Y は、 X:(X=Y) の略記です。 X は Y と同格の項であることを示します。
［例］
X# {a/1, b/X!a} = Y. は、 X = Y = {a/1, b/1} となります。

2.2 遅延実行制御

2.2.1 遅延実行制御機能

CIL の遅延実行制御のオリジナルは、 Prolog-II(Colmerauer) のフリーズ制御です。 CIL の遅延実行制御は、 PSI の bind_hook を用いて実現されています。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」ことを指定する述語です。

例えば、遅延実行制御を用いると、「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。」 と言うようなプログラムを次のように記述することができます。

..... 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 に対してデーモンとして述語 test を登録する」という意味です。 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 にわたります。

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ます。上の場合では、 test(X?) は「X は test を満足する」という宣言とみなすことができます。

2.2.2 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，以下の諸記法を提供しています。

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? と書きます。 print(X?) は bind_hook(X, print(X)) を意味する記述です。
［例］
print(X?), X = ok. は、 ok が表示されます。
print(X?). だけでは、 X は未定義のままで何も表示されません。

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

3. ?? 記法
X?? は、T:(T?=X) の略記です。
［例］
X = a(Y??) は freeze(T, (T=Y)), X=a(T) と同じです。これを実行すると Y の状態にかかわらず X は具体化します。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 は具体化しません。

2.2.3 制約記述

構文解析や意味解析などの制約を記述するために、遅延実行制御の応用として、組込述語 constr を提供しています。組込述語 constr の第一引数に制約が記述できます。制約には、基本制約として、論理式、等式、不等式、計算式、代入が記述でき、それらの基本制約を or や and などの接続子で結合した制約式が記述できます。

［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ます。

2.3 マクロ機能

2.3.1 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ます。これにより、CIL プログラムから ESP プログラムを利用することができます。

2.3.2 ユーザ・シンタックス・シュガー機能

ユーザー定義による言語機能拡張が行えます。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ます。

3 CIL プログラミング環境

3.1 コマンド・インタープリタ

コマンド・インタープリタはユーザー・インターフェイスのトップレベルを司り、処理系に対するコマンドの入力と実行やゴール列の入力と結果の表示などを行います。

3.2 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を提供しています。また、ユニット中のファイルを扱うための操作機能があります。

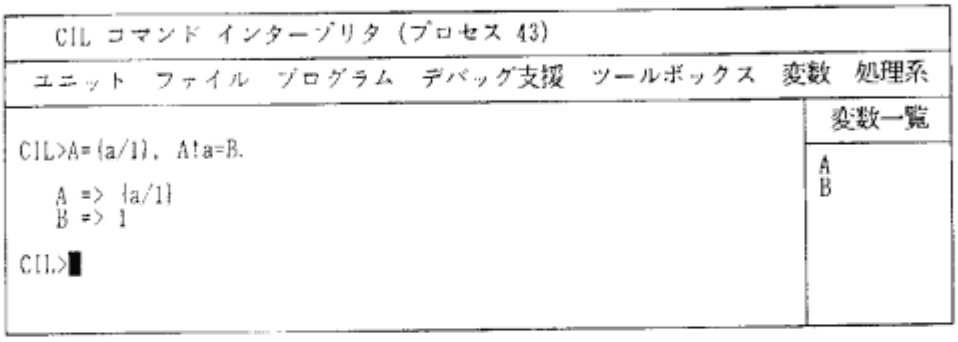


図 3: コマンド・インタープリタ

3.3 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, 未参照述語のチェックと編集を対話的に行うことができます。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ます。

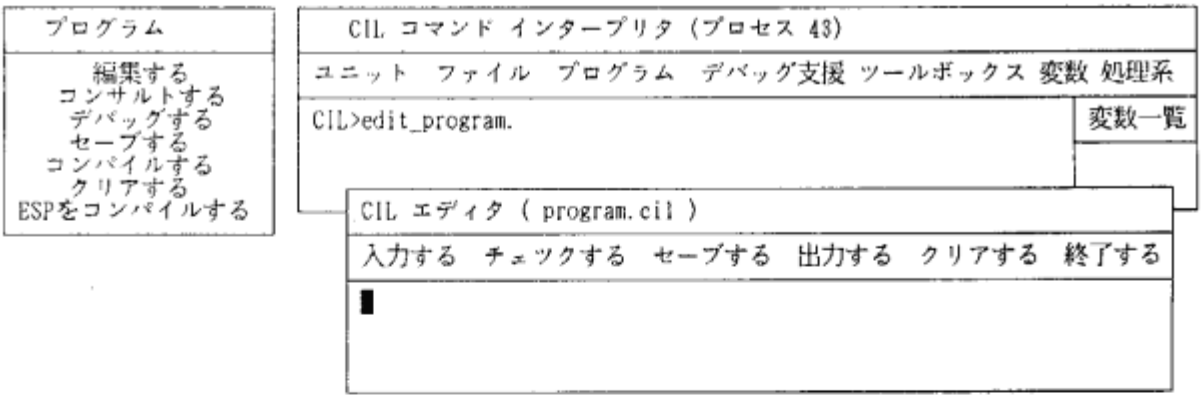


図 4: プログラム・エディタ

3.4 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ります。プログラムの実行はゴールを入力することによって行います。

3.5 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できます。また、トレース・モデルは遅延実行制御や条件付き項, 制約

をトレースするために、拡張トレース・モデルとなっています。それにともないリーシュ・コマンドもCIL独自の使い易いコマンドとなっています。

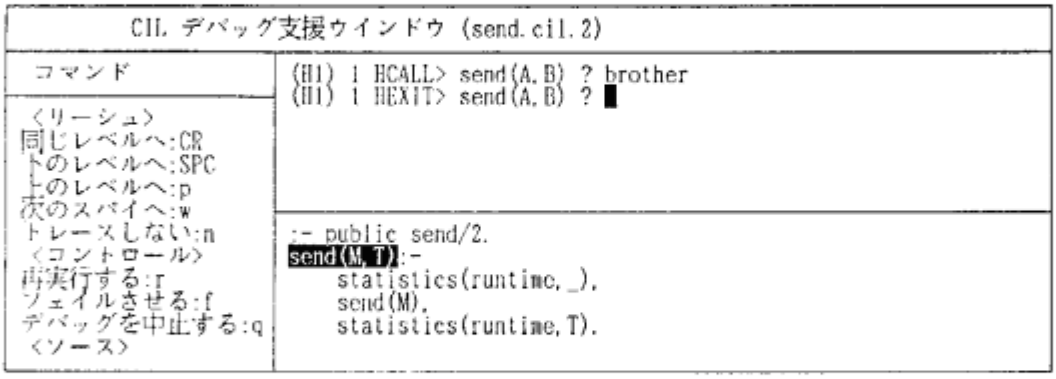


図 5: デバッグ支援ウインドウ

3.6 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ます。

3.7 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できます。

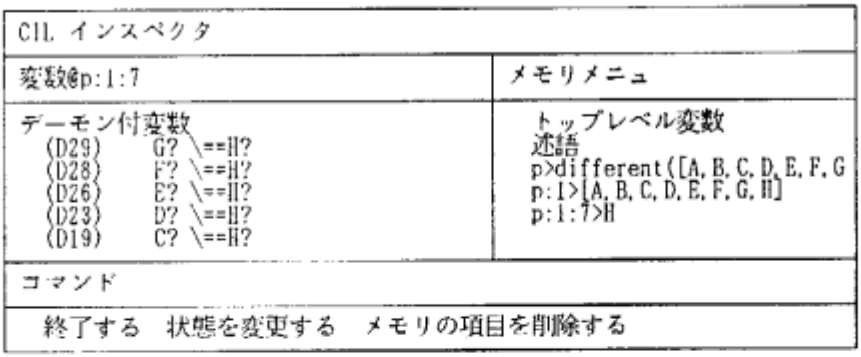


図 6: インスペクタ

3.8 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ります。ツールボックスの管理と利用の機能を提供しています。

第 III 部
形態素解析システム LAX

4 概要

形態素解析とは，分かち書きされていない文から語 (文節) の並びを認識する処理を言います．例えば，「ミカンを食べる。」という文が「ミカン・を」と「食・べ・る・。」という二つの語から構成されていることを判定します。ここで中黒 (・) で区切られているのが形態素と呼ばれる単位です．形態素解析の結果は語の間の関係を調べる構文解析などで使用されます．従って形態素解析処理部は日本語処理システムの入り口の処理モジュールと言えます．

形態素解析システム LAX[2](以下 LAX) は，このような形態素解析を行うプログラムの開発環境を提供しています．LAX では，LAX 辞書と呼ばれる形態素辞書をユーザが作成し，それを変換系を用いて形態素解析プログラムに変換します．LAX 辞書は，PSI の汎用エディタである pmacs エディタを用いて作成しても良いし，専用の辞書エディタを用いて作成することもできます．また，インスペクタと呼ばれるツールにより開発中の辞書を用いて随時形態素解析試験がおこなえ，効率の良い形態素辞書の開発が可能となっています．

LAX の生成する形態素解析プログラムは，50 文字程度からなる日本語文なら 0.1 秒程度で解析でき，また未登録形態素[3]が現れた場合も LAX はそれを認識することができます．

以下では、LAX の特徴とツールの簡単な操作法を説明します．なお，詳細はマニュアル [7, 8] を参照して下さい．

5 LAX の特徴

5.1 特徴

LAX では形態素解析プログラムは，後で説明する形態素辞書記述形式で書かれた LAX 辞書を変換して得られます [1]．そのプログラムはレイヤードストリーム方式 [2] と呼ばれるメカニズムにより解探索をおこない，決定的に，すなわち後戻りなしに全解を求めます．レイヤードストリーム方式は並列性の高いアルゴリズムであるばかりでなく，PSI のような逐次処理マシンでも十分実用的な時間で解析処理が可能です．

また形態素辞書は変換時に TRIE 構造化されており，逐次論理型言語 ESP のクローズインデキシング [6] が効果的に働き，高速に検索されます．一般に形態素解析では解に多くの曖昧さが生じますが，LAX では曖昧さが生じた時点で解を縮退するローカル・アンビギュイティ・パッキングと呼ばれる手法を採用して解析の効率を上げています．

また，解析時に未登録形態素が現れた場合も LAX はその語が未登録であることを示す情報を付加して出力します．

5.2 形態素辞書記述形式

5.2.1 形態素文法と辞書

形態素のカテゴリ体系やその分類基準，接続規則の体系などをまとめたものを形態素文法と呼びます．LAX 辞書は，その形態素文法に法り，図 7 の記述形式で記述される形態素の定義の集まり

[2]Lexical Analyzer for Syntax and Semantics

[3]LAX 辞書に登録されていない形態素

```
begin(形態素カテゴリ).
    表層(トークン) ::    状態遷移表f([接続素性f1,..])
                    && [ 状態遷移表b1([接続素性b11,..]),
                                :
                         状態遷移表bn([接続素性bn1,..]) ].
            :
end(形態素カテゴリ).
```

図 7: 形態素辞書記述形式

です. ところが形態素辞書の開発中には, しばしば文法体系自身も変更を受けるので, 形態素辞書の開発と形態素文法の開発はほとんど同一作業であると言えます.

LAX 辞書に記述する形態素文法は正規文法[4]のクラスに属しています. この記述形式は, 森岡の形態素文法 [5] に基づいた文法の記述がし易いように工夫されています [4].

5.2.2 記述形式

図 7のように, 各形態素は形態素カテゴリ毎にまとめて記述し, それぞれに接続規則を定義します. 接続規則は入力文字列から切り出すべき形態素の表層表現を見出しとして, 切り出した後に解析結果として返す文字列のトークンと 2 種類の接続素性とからなります. 2 種類の接続素性とは, 自分の形態素の識別子となる左方接続素性と, 後接しうる形態素の識別子をまとめた右方接続素性です. 接続素性には任意のアトムが使用できます. またトークンは, 表層表現と同じであれば省略することができます.

形態素解析の基本は, ある形態素の並びが語となるかどうかの判定処理で, そのメカニズムは, 語の開始素性を初期状態, 語の終了素性を最終状態とする非決定性有限状態オートマトンとみることができます. LAX 辞書は, このオートマトンの状態遷移関数を定義していると言えます. すなわち, 形態素の表層が入力記号を, 左方接続素性が入力状態を, 右方接続素性が遷移可能な状態集合を表しています.

5.2.3 状態遷移表と接続のセマンティクス

LAX では, 接続素性を記述するには必ずそれが属する状態遷移表名を指定する必要があります. これは, 性質の似通った素性を一つの状態遷移表にまとめて管理できるので, 効率的な形態素文法の開発に役立ちます.

状態遷移表は必要なだけユーザが宣言して使いますが, end という状態遷移表は, あらかじめシステムが用意していて特別な働きを持っています. すなわち, 状態遷移表 end 内の素性で接続された場合, 左方形態素が語の終わりを, 右方形態素が語の始まりを意味します. つまりそこで語と語が切れているとシステムは判断します.

形態素と形態素が接続される時の条件はつぎの通りです. 隣接する二つの形態素について, 右方形態素の左方接続素性中の状態遷移表が, 左方形態素の右方接続性中に存在し, かつそれぞれの遷移表内に共通する接続素性がただ一つ存在する場合に, それらの形態素は接続され, 語の構成要素となります.

[4] 3 型文法とも言う

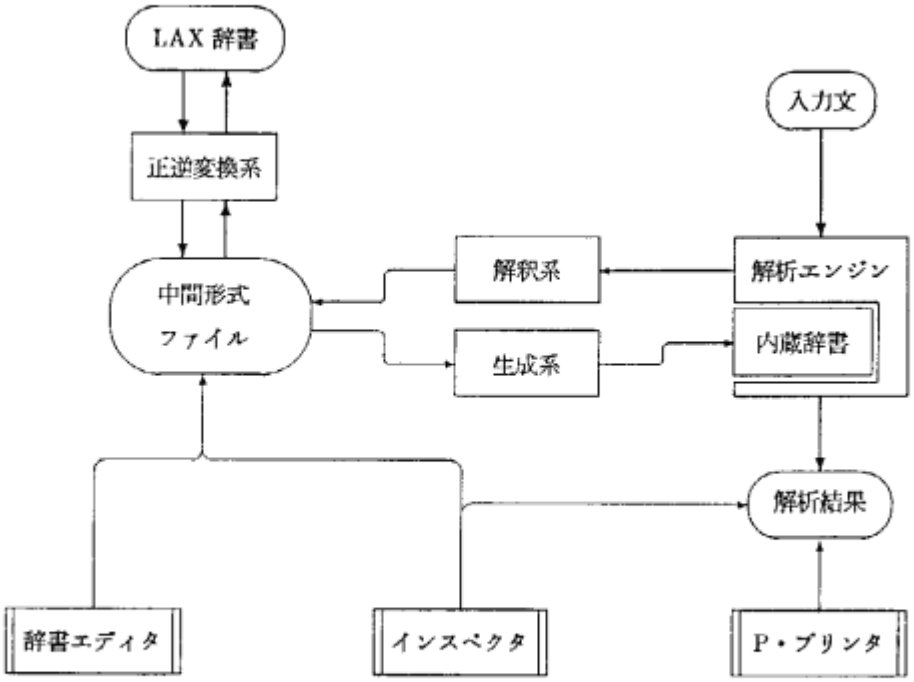


図 8: 形態素辞書開発環境

6 ツールの操作法

6.1 形態素辞書開発環境

図 8は LAX の辞書開発環境を示した図です. 図 7の辞書記述形式で書かれた LAX 辞書は, 変換系を通して中間形式ファイルに変換されます. LAX システムでは, この中間形式ファイルを編集することにより辞書開発をおこないます. LAX 辞書は, 逆変換系を用いて中間形式ファイルからいつでも変換可能です. この逆変換の機能は主に中間形式ファイルのバックアップの意味と現在の辞書の内容の確認のための二つの目的のためにあります.

開発の終了した中間辞書ファイルから生成系により, プログラム化された辞書 (内蔵辞書) が得られます. 形態素解析プログラムはこの内蔵辞書と, それをアクセスし実際に形態素解析を行う解析エンジンとから構成されます. 内蔵辞書は TRIE 構造化されており, ESP のクローズインデキシングが効果的に働き高速な辞書引きがなされます.

一方, 形態素辞書開発中の形態素解析では, 解釈系により中間形式ファイルを直接解釈して辞書引きがなされます. そのため辞書の修正結果が直ちに確かめられ, 効率の良い辞書開発が行えます. 解釈系を用いることによる形態素解析時間の増大は 10 倍前後です. それでも, 50 文字程度からなる文の解析は 1~2 秒なので, 実用上は問題ありません.

開発支援ツールとして, 辞書の検索, 更新, 追加, 削除等をおこなう辞書エディタと, 形態素解析結果を調べたり形態素の接続試験をおこなうインスペクタと, 形態素解析結果を曖昧さが分かり易い形で表示するプリティ・プリンタがあります [3]. 以下順を追って, これらのツールの説明をしていきます. 図 9と図 10に, 各ツールの画面のハードコピーを掲げてあります.

6.2 LAX トップ・ウインドウ

画面1がLAXを起動した直後に現れるトップ・ウインドウです．ここからは，後述する辞書エディタやインスペクタなどのツールを呼び出したり，LAX辞書を中間形式ファイルに変換したり，中間形式ファイルから内蔵辞書を生成したりといった作業をおこないます．画面1は，LAX辞書の変換をおこなおうとしているところです．一般のエディタを用いてLAX辞書を作成した場合はこのように中間形式ファイルへ変換する必要がありますが，辞書エディタを用いると自動的に中間形式ファイルが作られていきます．

このほかにトップ・ウインドウでは，インスペクタなどで形態素解析試験をおこなう時に使用する辞書を内蔵辞書にするか(コンパイルモード)，解釈系を用いて中間形式ファイルをアクセスするか(インタプリタモード)の選択がおこなえます．またLAXでは複数の辞書を平行して開発することが可能なので，どの辞書を選択するかを任意に切り換えることができます．

6.3 辞書エディタ

辞書エディタを新規で起動すると画面3の初期設定画面が現れます．ここでは辞書の名前(ソース名)を始めとして，状態遷移表名や，辞書の分割数[5]，各種の値の設定をおこないます．これらは内蔵辞書の生成と中間形式ファイルの作成に必要な情報です．値を設定しなければ，左側のデフォルト値が採用されます．設定が終了すると，画面4の編集画面に移ります．

編集画面へは，上述の初期設定画面から移行する場合と，トップ・ウインドウから辞書エディタの「継続」を選択して移る場合，そして後述のインスペクタから呼び出す場合があります．インスペクタから呼ばれる場合を除くと，初めは画面には，「カテゴリ」，「表層」などの項目名が並んでいるだけです．ここで画面4のようにすべての項目に適切なデータを入力して追加コマンドを実行するとその形態素の定義が完了します．

形態素の検索は，表層をキーとするものと，形態素カテゴリをキーにするものがあり，いずれの場合も検索結果はホルダと呼ばれる単位で管理されます．そして検索直後はホルダの先頭要素の内容が編集画面に表示されます．ホルダ内の他の形態素の記述は順に閲覧することができ，その各々に対して削除や記述の修正が可能です．検索されたデータを元に別の形態素の定義を行うこともできます．

6.4 インスペクタ

インスペクタは，トップ・ウインドウか辞書エディタから呼ばれます．画面5と画面6にあるように，インスペクタは内部に解析モードとインスペクトモードを持っています．呼び出された直後は解析モードになっています．解析モードでは形態素解析を行い，期待通りの解析結果がでているかどうかを確かめられます．解析モードでは解析結果だけでなく，解析の成功失敗，解析の曖昧さ，入力文字数，解析時間などの情報も同時に得られます．

画面5は，文入力コマンドを用いて入力ウインドウから文を入力し形態素解析をおこなったところです．解析結果表示ウインドウに一行あたり一語が表示されます．一語が複数の形態素から構成される場合は，形態素は黒丸(・)で分割されます．また，語の作り方に曖昧性が合った場合には例えば次のようにスラッシュ(／)で分割されて表示されます．

入　　力：　食べていく。

解析結果：　食・べ・て・い・く。／食・べ・て

[5]ESP言語では，1クラスに定義できる形態素数に限度があるため辞書を分割する必要がある

い・く。

解析結果の表示にはこの他に形態素カテゴリまでつけて表示するモードと，それに加えて接続した素性も表示するモードの２種類あります．形態素の接続に失敗した場合には，失敗した形態素の後ろにアスタリスクマーク (*) が表示され，修正すべき形態素をすぐに知ることができます．文の入力にはこの他に，あらかじめ入力用の文をファイルにしておいてそれをメニューで呼び出して形態素解析をおこなう文選択コマンドがあります．

画面５の状態で，入力ウインドウの入力文字の任意の一文字をマウスクリックすることにより画面６のインスペクトモードに移行します．画面６の例では「･･･ミカンを･･･」の「を」の文字をクリックしたところです．すると解析結果表示画面が四つに分割されて，右上にクリックした文字から始まる形態素が表示され，左上にはクリックした文字から始まる形態素の左側に隣接している形態素が表示されます．言い換えれば，クリックした文字から始まる形態素と接続する可能性のある形態素がすべて表示されます．

この時点で，任意の形態素をマウス左１回クリックすることにより，すぐ下のウインドウにその形態素の辞書記述内容が表示されます．マウス左２回クリックすることにより，クリックされた形態素に接続しうる形態素すべてに番号が振られます．そのなかでアスタリスクマークがつけられた形態素は，下に辞書内容が表示されます．これらの機能により，形態素の記述の不備が容易に発見できます．

また，各形態素をマウス中１回クリックすることにより，その形態素の辞書エントリを辞書エディタのホルダに登録することができます．マウス中２回クリックでは，ホルダへの登録と辞書エディタの起動を一度におこないます．この方法で起動されると，辞書エディタはホルダの最初のエントリを表示して待機しています．従って，必要な変更を施した後にインスペクタに戻り，すぐに修正の結果を確かめることができます．

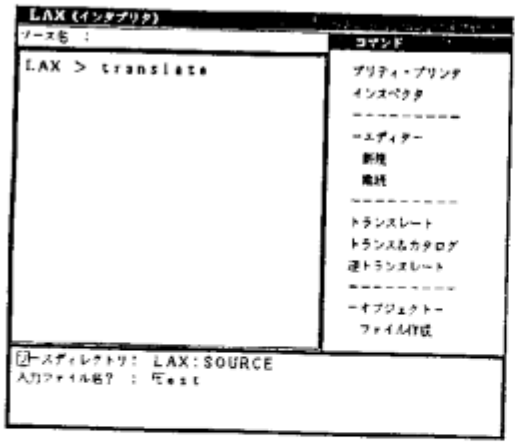
6.5　プリティ・プリンタ

画面２のようにプリティ・プリンタは，形態素解析をおこない，その結果を曖昧さが分かり易いような形で表示するためのツールです．文の入力方法はインスペクタと同じです．解析結果が大き過ぎて画面に入り切らない時は矢印記号のコマンドをマウスクリックすることにより，見たい部分を画面内に移動させることができます．

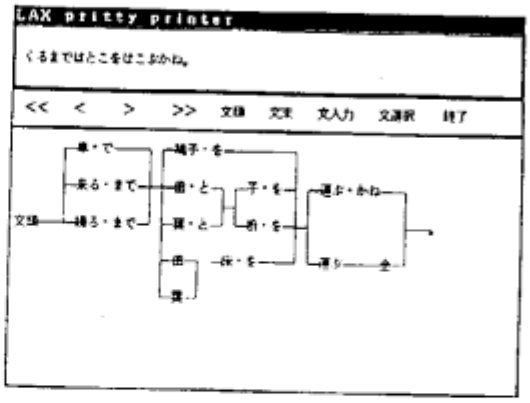
7　おわりに

最後に，付録 A に LAX 辞書のサンプルを掲げておきます．非常に小さな辞書ですが，LAX 辞書のシンタクスのほとんどを使用していますので，理解の助けになるかと思います．

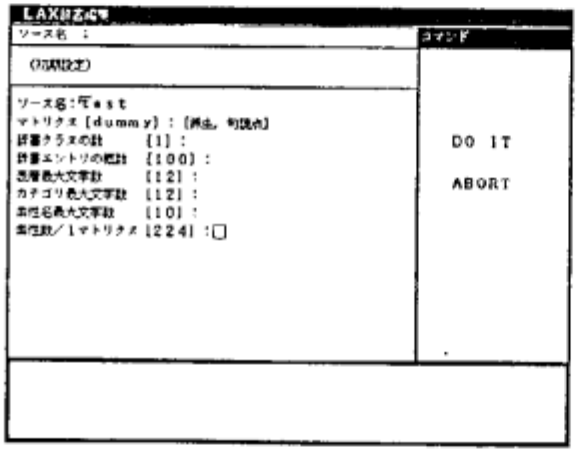
LTB のツールの一つである形態素解析システム LAX の特徴とツールの操作法について簡単に説明しました．LAX は誕生してから日が浅くツールの機能に未整理な部分も若干ありますが，全体的にはかなり使い易いものができたと考えています．今回，成果物指定を受けたのを機会に少しでも多くの方に使用して頂き，不満なところや改良点のご指摘，あるいは新しいツールのアイデアなどを頂いて，少しでも使い易い，役に立つツールにしていきたいと考えています．



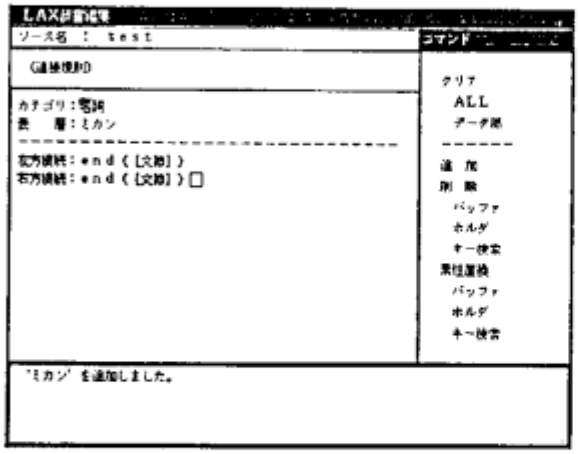
画面 1: トップ・ウインドウ



画面 2: プリティ・プリンタ

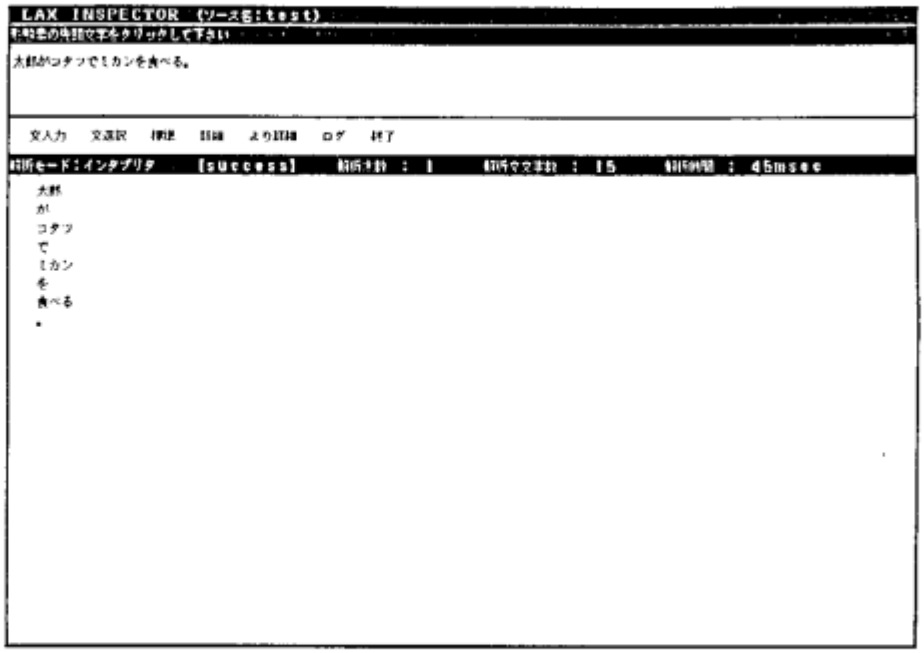


画面 3: 辞書エディタ - 初期設定画面 -

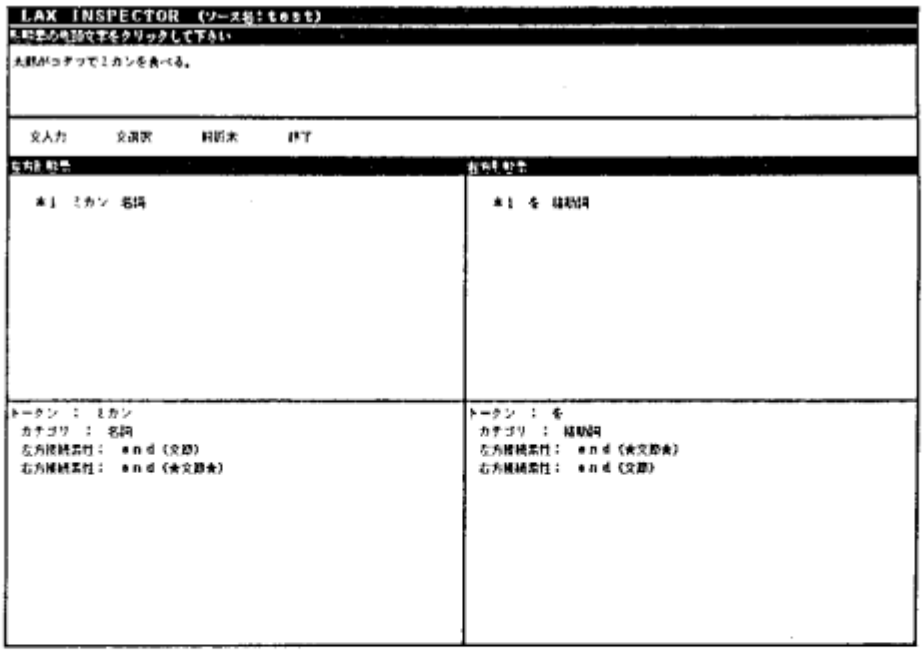


画面 4: 辞書エディタ - 編集画面 -

図 9: 各ツールの画面 I



画面 5: インスペクタ - 解析画面 -



画面 6: インスペクタ - インスペクト画面 -

図 10: 各ツールの画面 2

付録 A：LAX 辞書記述例

```
% Sample morphological dictionary for LAX

conn_matrix([連接,句読点]).    % 使用する状態遷移表名(end はデフォルト)

classes(1).                    % 辞書の分割数(800 語 / クラスが目安), 2 の冪乗数を指定する
entry_number(100).             % 形態素数の概数, 省略可
word_length(10).               % 形態素の最大文字数, 省略可
category_length(10).           % 形態素カテゴリの最大文字数, 省略可
feature_length(10).            % 接続素性の最大文字数, 省略可
feature_number(224).           % 状態遷移表内の最大素性数, 省略可

begin(spec).

%========== 名詞 ==========
begin(名詞).                   % 形態素カテゴリのブロック指定
くるま(車) ::   end([文節])
          && [ end([文節]),
               連接([格助詞]),
               句読点([読点,句点]) ].
はとと(鳩子) :: same.          % 最も新しく定義された接続素性と同じ内容を持つ
かね(金)     :: same.
は(葉)       :: same.
と(子)       :: same.
とと(床)     :: same.
end(名詞).

%========== 格助詞 ==========
begin(格助詞).
が ::   連接([格助詞])         % トークンが表層と同じ場合は省略可能
   && [ end([文節]),
        句読点([読点,句点]) ].
で   :: same.
を   :: same.
に   :: same.
end(格助詞).

%========== 動詞 ==========
begin(動詞).
はこぶ(運ぶ) ::   end([文節])
             && [ end([文節]),
                  連接([終助詞,副用助詞]),
                  句読点([読点,句点]) ].
くる(経る) :: same.
くる(来る) :: same.
end(動詞).

%========== 副用助詞 ==========
begin(副用助詞).
まで ::   連接([副用助詞])
     && [ end([文節]),
          句読点([読点,句点]) ].

end(副用助詞).

%========== 句読点 ==========
begin(句読点).
。 ::   句読点([句点])
   && [ end([文末]) ].

、 ::   句読点([読点])
   && [ end([文節]) ].
end(句読点).

%========== 終助詞 ==========
begin(終助詞).
かね ::   連接([終助詞])
     && [ end([文節]),
          句読点([句点]) ].
end(終助詞).
end(spec).
```

参考文献

[1] 杉村領一, 赤坂宏二, 久保幸弘, 松本裕治, 佐野洋, 論理型形態素解析 LAX, *Proceedings of the Logic Programming Conference '88*, pp213-222,1988

[2] 奥村晃, 松本裕治, レイヤードストリームを用いた並列プログラミング, *Proceedings of the Logic Programming Conference '87*, pp223-232,1987

[3] 久保幸弘, 妙泉正隆, 佐野洋, 赤坂宏二, 杉村領一, LTB- 形態素解析システム LAX の開発環境, 情報処理学会第 37 回 (昭和 63 年後期) 全国大会論文集 (II), pp1078-1079,1988

[4] 佐野洋, 赤坂宏二, 久保幸弘, 杉村領一, 語構成に基づく形態素解析, 情報処理学会第 36 回 (昭和 63 年前期) 全国大会論文集,1988

[5] 森岡健二, 語彙の構成, 現代語研究シリーズ 1, 明治書院

[6] ICOT, 小型化 PSI ESP 説明書 (simpos 3.1 版), pp150-151, 1988

[7] ICOT 第二研究室, LTB1.1 版 操作説明書 - LAX 編 -, 1989

[8] ICOT 第二研究室, LTB1.1 版 ソフトウェア仕様書 - LAX 編 -, 1989

第 IV 部
構文解析システム SAX

汎用日本語処理系 (Language Tool Box) の一モジュールである構文解析システム SAX について紹介します。まず、SAX の概要について述べます。次に SAX を実行する際必要なデータ、すなわち文法の記述形式（記述方法）について説明した後、本システムが提供する文法開発環境について述べます。

8 SAX の概要

8.1 SAX とは

構文解析システム SAX (Sequential Analyzer for syntaX and semantics) は、ユーザによって与えられた文法に従い、自然言語の文の構文解析を行なうためのシステムです。ここでいう構文解析とは、「入力」として 与えられた単語の列（文）の構文的構造を決定する手続きを言います。「入力」は、現在

- 形態素解析システム LAX の結果得られた構文意味情報
- キーボードからタイプされた単語列

の 2 通りが可能です。

8.2 解析アルゴリズム

SAX の解析アルゴリズムは、左隅上昇型のアルゴリズムです。すなわち、ある（非）終端記号が構成されると、それを基により大きな解析木を構成します。その時、その（非）終端記号を文法規則中右辺の左端の要素として持つ文法規則を選択し、その文法規則の適応を試みますが、適応可能な文法規則が複数存在する場合、それらすべてを同時に実行します。例えば下の文法規則において、非終端記号 det が構成されると、det を右辺の左端にもつ規則 (2),(3) が同時に適応されます。

```
(1) sentence → np,vp.
(2) np       → det,noun.
(3) np       → det,adj,noun.
(4) vp       → tv,np.
(5) vp       → iv.
(6) np       → ['I'].
(7) tv       → [have].
(8) det      → [a].
(9) noun     → [pen].
```

8.3 システム構成

SAX のシステムは以下のようなサブモジュールから構成されています。

1. トランスレータ
   ユーザによって与えられた「文法」を計算機上で実行可能な「構文解析プログラム」に変換する。

2. 文法開発支援ツール (デバッガ)
   文法のデバッグを支援する。トレーサ、ビジュアルデバッガ、ツリーブラウザがある。

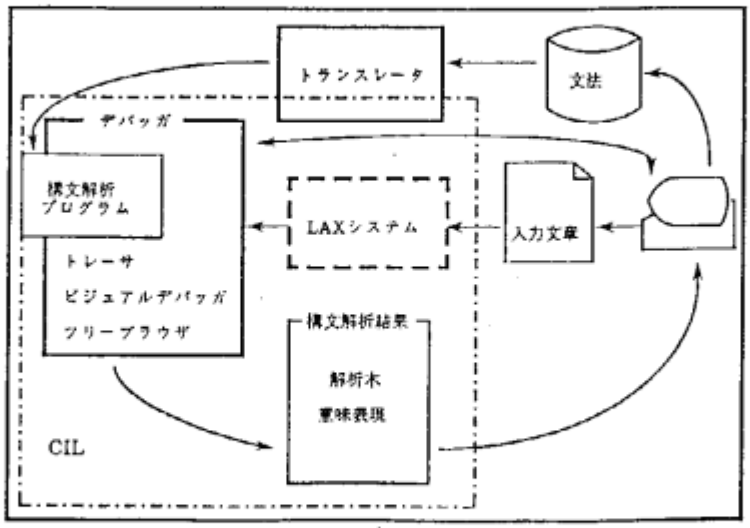


図 11: SAX のシステム構成

9 文法の記述形式

SAX で扱う文法記述形式は、DCG(definite clause grammar) を基本にしていますが、先に述べた解析アルゴリズムからの制約のため、DCG の記述に一部制限を加えています。一方、自然言語の文を効率よく実現するために、DCG にはない記法も提供しています [3]。以下では、SAX の文法記述形式を拡張と、制限とに分けて述べます。

9.1 拡張

1. 遅延補強項

   本システムでは、構文解析過程で評価される補強項の他に、構文解析終了後に評価を行なう遅延補強項の記述が可能です。これにより、構文的に不適当な解析木の生成を抑制するための条件を補強項で記述し、それに直接関係しない意味構造の構成規則などを遅延補強項に記述することができます。

   記述例

```
vp(V,Np,Sem) → vt(V),np(Np),{agreement(V,Np)}
                &{construct_sem(V,Np)}.
```

   遅延補強項は文法規則の最後に 1 つのみ “ & ” で区切って記述することができます。

2. 優先度計算

   一般に自然言語の構文解析を行なった結果、複数の解析結果が得られます。ここで得られた構文レベルでの曖昧性を無理に絞り込まず、談話処理の中で解決する手法も提案されて

いますが、本システムでは、解析途中に解析過程での局所的な情報を用いて、解析結果の尤もらしさを計算する機能を備えています。また、この規則の中で様々な情報を参照するための記法を予約語として用意しています。

(a) 文法規則中の右辺の各文法カテゴリと、左辺の文法カテゴリとの、結びつきの強さの計算が行なえる。

予約語

pref_cat … 各文法カテゴリが持つ点数。例えば下の記述例 (1) の pref_cat は、 vt が持つ優先度点数。

pref … 各文法カテゴリが、その文法規則に含まれる尤もらしさを表す点数。

(b) ある文法規則が成立したときに、その解析木が持つ優先度の計算を行なう。

予約語

pref_CAT … この文法規則が完成した時に文法規則右辺に与えられる点数。

prefs(i) … 文法規則の右辺の第 i 番目の文法カテゴリの優先度計算。

記述例

```
vp(V,Np,Sem) → vt(V)::{pref_rule1(V,pref_cat,pref)},          (1)
               np(Np)::{pref_rule2(V,Np,pref_cat,pref)}       (2)
               &&{pref_CAT = prefs(1) + prefs(2)}.            (3)
```

文法カテゴリの後ろに“：：”あるいは“＆＆”で区切って記述することができます。

3. CIL の導入

CIL は部分項による記述や部分項による単一化操作の扱いが可能です。これは自然言語の意味処理を行なうのに適しています。SAX は、DCG の補強項に CIL のプログラムを記述することができ、これによって構文解析と同時に意味処理を行なうことが可能です。CIL プログラムの評価は、SAX の解析実行部が補強項の評価が必要となった時点で、CIL インタプリタを呼び出すことによって実現しています。

記述例

```
sentence(S) → np(Np),vp(Vp),{S!subj = NP,S = 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}}.
```

9.2 制限

1. ε 規則が許されない。
   従って

```
rel_clause → [].
```

   といった記述はできません。

2. 補強項を右辺の先頭に書くことはできない。
   よって

```
np(Np) → {check_np(Np)},det(Det),noun(Noun).
```

   といった記述はできません。

3. 補強項の評価は一度のみ行われる。

   SAX によって得られる構文解析プログラムは決定的に動作し、バックトラックはおこらないため、補強項の評価も一度のみ行なわれます。

4. 文法規則右辺に現れる変数には束縛しないような規則を書く。

```
(1) sentence(S) → np(Np),vp(Vp).
(2) np(Np)      → det(Det),noun(Noun).
(3) np(Np)      → det(Det),noun(Noun),wh_close(Wh).
(4) vp(Vp)      → vt(Vt),np(Np).
(5) noun(Boy)   → [boy],{dic(boy,Boy)} .
(6) noun(Dog)   → [dog],{dic(dog,Dog)}.
(7) vt(Hits)    → [hits],{dic(hits,Hits)}.
(8) det(The)    → [the],{dic(the,The)}.
(9) det(A)      → [a],{dic(a,A)}.
```

   一般に、文法規則右辺の左端のカテゴリが同じ規則は、文法規則中に複数存在します。例えば上の文法規則では、(2),(3) がこれに相当します。先にも述べたように、SAX は複数の可能性を同時に実行し一つの部分木は複数の上位の木に共有されるために、それらの部分木 (det) が持つ変数も共有されます。従って

```
(2)' np(Np) → det(Det),{Det = the},noun(Noun).
```

   のような書き方をすると (3) の Det もこの値を持つことになります。本来は (2),(3) は独立に行なわれるべき解析ですが、このように互いに影響しあうために、ユーザが全く予期しない結果が出力されてしまうことがあります。(これを多環境の問題と呼んでいます)

10 文法開発支援ツール

SAX システムでは、文法を開発するための支援ツールとしてトレーサ、ビジュアルデバッガ、ツリーブラウザを提供しています。

- トレーサ (動的デバッグ環境)

- 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。（図 12参照）

- ビジュアルデバッガ(静的デバッグ環境)

  - 構文解析時の解の探索履歴を用いて、部分木情報の表示などを行なう。解析の失敗位置を特定するために使用される。（図 13, 14, 15参照）

- ツリーブラウザ

  - 構文解析の結果、得られた木構造の数、解析所要時間、優先度得点の表示。
  - それらの木構造を縮小したもののメニュー表示。（図 16参照）
  - 縮小された木構造の各々についてその詳細をウィンドウへ表示。
  - 木構造の表示範囲の指定。（図 17参照）

参考文献

[1] 松本裕治, 「並列構文解析」自然言語処理研究会, 1986.

[2] Y.Matsumoto and R.Sugimura, "A parsing system based on Logic Programming", *Proceedings of the International Joint Conference of Artificial Intelligence*, 1987.

[3] 松本裕治, 杉村領一, 「構文解析システムSAXのための文法記述言語」, In *5th Conference Proceedings of Japan Society for Software Science and Technology*,1988.

[4] 杉村領一, 「論理型文法における制約解析」, In *Proceedings of the 2nd Annual Conference of Japanese society for Artificial Intelligence*,1988.

[5] 杉村領一, 赤坂宏二, 久保幸弘, 佐野洋, 松本裕治, 「論理型形態素解析LAX」, In *Proceedings of the Logic Programming Conference*,1988.

[6] 赤坂宏二, 「構文解析システムSAXの構成」, *NLUSG2*,1988.

[7] 山崎重一郎 他, 「LTBにおける構文解析システムSAXについて」, 情報処理学会全国大会,1987.

[8] 山崎重一郎 ,杉村領一, 赤坂宏二, 松本裕治, 「構文解析システムSAXのデバッグ環境」, In *Proceedings of the 2nd Annual Conference of Japanese Society for Artificial Intelligence*,1988.

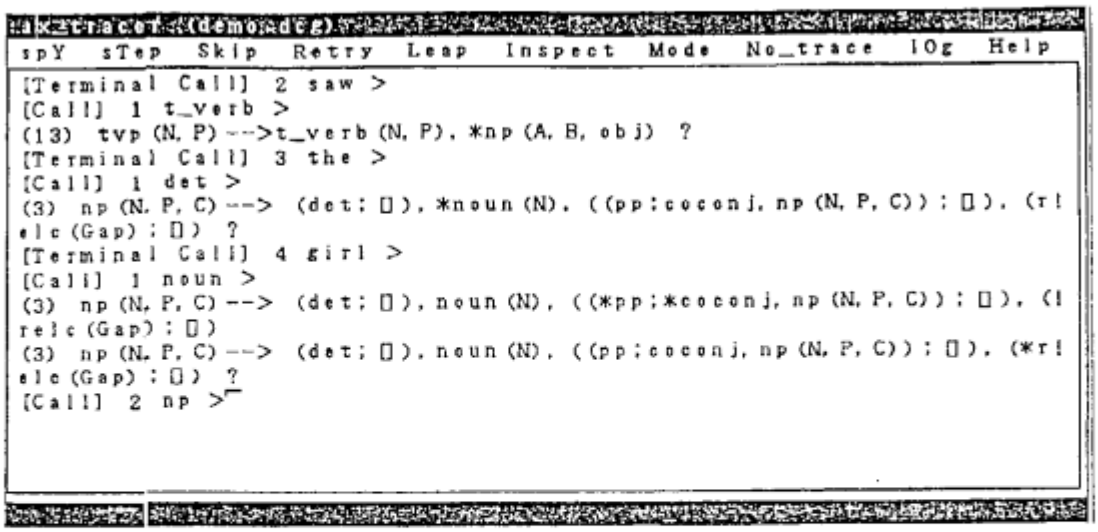


図 12: トレース過程

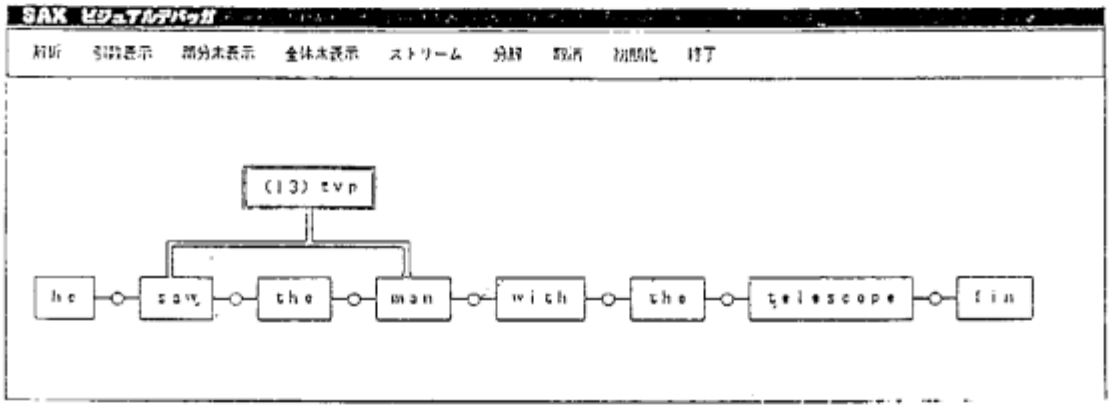


図 13: ビジュアルデバッガ

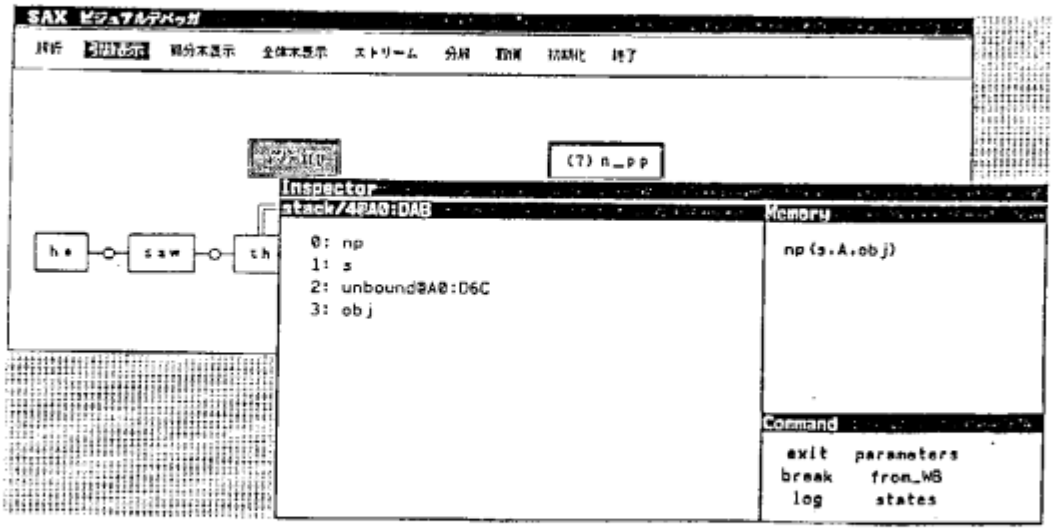


図 14: ビジュアルデバッガでのインスペクタ

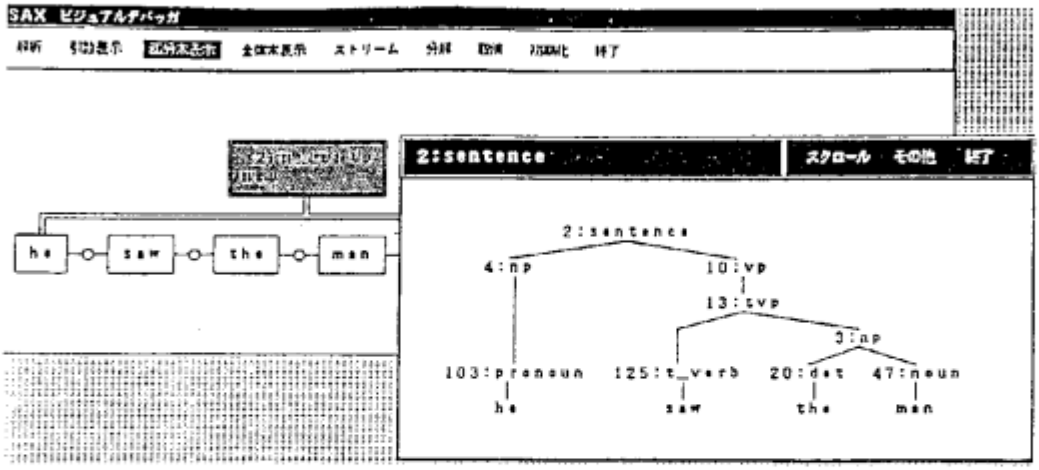
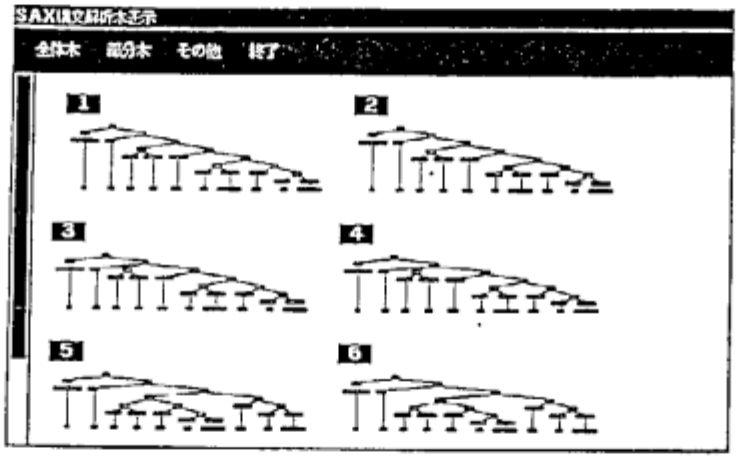


図 15: ビジュアルデバッガでの部分木表示

スクロール前



スクロール後

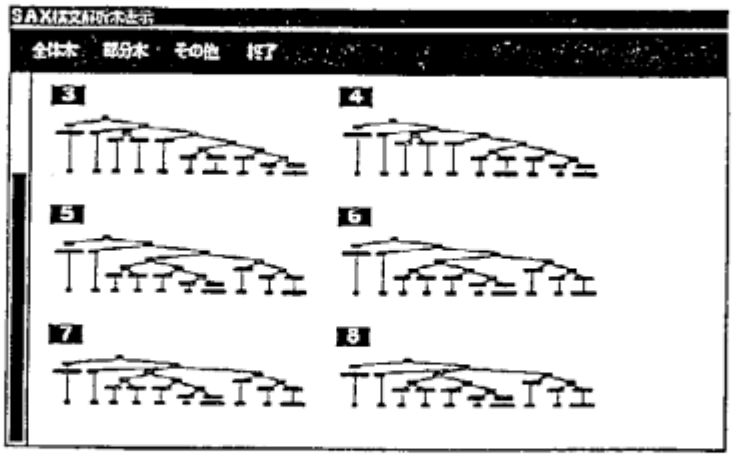
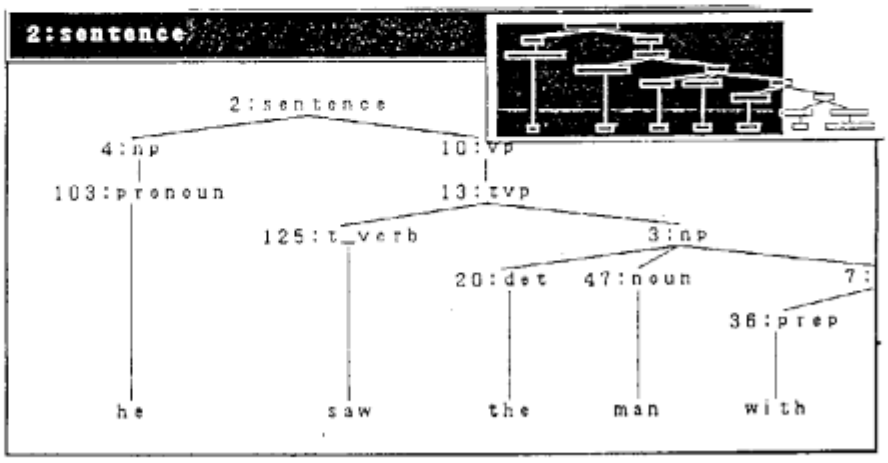


図 16: ツリーブラウザ（メニューウィンドウ）

スクロール前



スクロール後

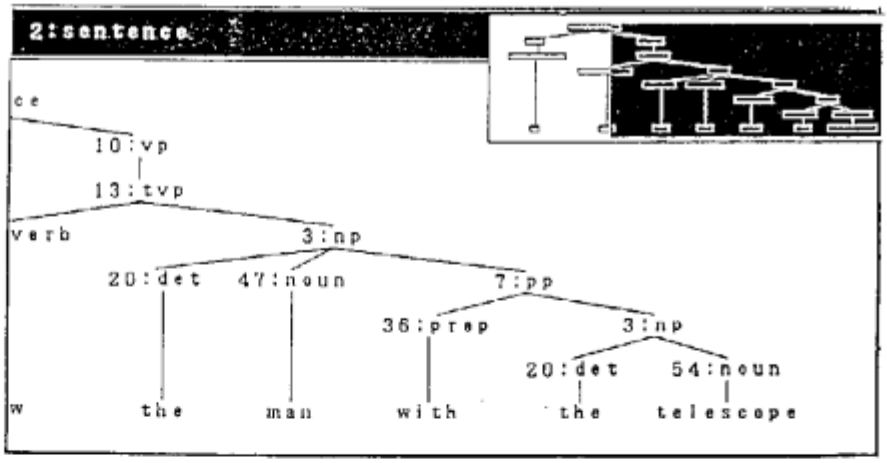


図 17: ツリーブラウザ（木構造の表示範囲の指定）

第 V 部
文生成部

11　概要

文生成部は一文に関する統語情報 (「中間表現」) から、その文の表層表現を得るものです。中間表現の記述は CIL の部分項 (PST:Partially Specified Term) を用い、比較的意味表現に近い「標準マクロ表現」と表層表現に近い「基本表現」による記述が可能です。文生成部のツール類としては、生成実験のための標準マクロ編集機能や実行制御機能等が用意されています。

12　文生成部の機能

文生成部の入力である「中間表現」は、文型の決定や語彙選択や省略処理などのいわゆる「文生成プラニング」処理の終了後に得られるもので、一文の統語情報が全て記述されています。即ち、文生成部は生成すべき文の統語情報がすべてわかった時点での表層文生成用のツールとして使用します。

文生成部の全体構成を図 18に示します。文生成部は大きく分けて「文生成機構」と「文生成用ツール」から構成されます。文生成機構は、「中間表現」を表層文に変換する文生成部の本体であり、CIL で記述されています。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を提供しており、これによりユーザは自分で作成した中間表現の生成実験等を行うことができます。

12.1　文生成機構

文生成機構は、「マクロ展開部」と「形態素生成部」に分かれます (図 18)。マクロ展開部は「標準マクロ表現」又は「基本表現」で記述された「中間表現」から、標準マクロの展開や「文生成辞書」の参照を行って「表層構造」と呼ばれる中間的なデータ形式を作り出します。形態素生成部はこの「表層構造」に対して活用処理を施して最終的に表層文を作り出します。

12.1.1　中間表現 (基本表現、標準マクロ表現)

文生成部の入力である中間表現には、ある一文の統語情報が記述されますが、その記述方式としては、比較的意味表現に近い「標準マクロ表現」と表層表現に近い「基本表現」の 2 つがあります。「標準マクロ表現」と「基本表現」は CIL の部分項で実現されています。

1. 標準マクロ表現
   図 19に“人間は自然の恵みに頼らなければならない”という表層文に対応した標準マクロ表現による中間表現の記述を示します。標準マクロ表現は基本表現に比べてより意味表現に近い表現形式でユーザはより簡潔に中間表現を記述できます[6]。即ち、深層格名による格の指定、連体修飾の指定等を記述できます。‘語彙’にはその単語の文生成用辞書上のエントリ名を書きます。

2. 基本表現
   図 20に同じ表層文に対応した基本表現による中間表現の記述を示します。基本表現はこ

[6]ユーザにとって記述が簡単になるような「標準マクロ」をシステムが提供しているといえます。

のように、文の構造を head(主辞) と complement(補語) で構成される 2 分木 (binary tree) で表現したものです。'lex' には、標準マクロ表現と同様にその単語の文生成用辞書上のエントリ名を書きます。この記法では接辞、語順などが陽に記述されています。

尚、この 2 つの表現形式を混在させた形で中間表現を記述することもできます。

12.1.2 表層構造

図 21に“自然に頼る”という表層文に対応した表層構造を示します。表層構造は基本表現で表していた 2 分木の構造をリストで表現したものですが、木のリーフはその単語の品詞名と文生成用辞書に記載されている辞書項目になります。尚、その単語が文生成用辞書に登録されていない場合は中間表現中のエントリ名がそのままリーフとなります。

12.2 文生成用ツール

文生成用ツールは、文生成機構の入出力データ (中間表現・表層構造) の編集、文生成機構の実行制御等の機能を提供します。ユーザはそれらを用いることで、自分の作成した中間表現の生成実験やユーザ自身で書いたマクロ展開プログラム (ユーザ定義マクロを用いたい場合) のデバッグができます。図 22に文生成用ツールのトップウインドウを図 23に中間表現エディタのウインドウを示します。

12.2.1 編集

中間表現や表層構造の編集に pmacs エディタの機能を提供しています。又、表示機能として、プリティプリントの他に、表層構造のグラフィック表示機能を提供しています。標準マクロ表現に関しては、そのシンタックス、語彙エントリの文生成用辞書上での登録の有無をチェックできます。

12.2.2 入出力先の指定

文生成機構の処理の入力元・出力先を指定する (ファイルかウインドウ) ことができます。ウインドウに出力する場合は中間表現の表示の深さも指定できます。

12.2.3 実行制御

文生成用ツールから CIL インタプリタのいくつかの機能が利用でき、文生成機構のコンサルト、コンパイル、トレースができます。又、連続して複数個の中間表現からの生成を行う一種のバッチ処理機能を持っています。

12.2.4 その他

システム立ち上げ時にコンサルトするファイル名等をデフォルトパラメータファイルに書いておくことで指定できます。

13　文生成機構の構成

13.1　マクロ展開部

マクロ展開部は「マクロ展開規則」と「マクロ展開エンジン」とから構成されます (図 24)。マクロ展開規則は標準マクロ表現を展開 (接辞や語順の決定等) して基本表現と等価な構造を作り出します。ユーザが標準マクロ以外のマクロ表記 (ユーザ定義マクロ) を使って中間表現を書きたい場合はユーザはマクロ展開規則のプログラムを書き換えることになります。

マクロ展開エンジンは基本表現やマクロ展開後の標準マクロ表現から表層構造を作り出すもので、文生成用辞書の辞書引きを行います。又、マクロ展開規則に対して各種のユーティリティ (例えば、動詞化や名詞化の処理など) を提供します。

13.2　形態素生成部

形態素生成部は「形態素生成用テーブル類」と「形態素生成エンジン」から構成されます (図 25)。形態素生成用テーブル類は活用処理に必要な各種データで、活用表や接続表等があります。形態素生成エンジンはこれらのテーブル類と表層構造内の辞書項目を参照して活用処理を行います。

13.3　文生成用辞書

文生成用辞書は単語の統語情報を CIL の部分項の形で与えるもので、マクロ展開エンジンは中間表現中の各単語のエントリ名からその辞書項目を検索します (図 26)。辞書項目はマスタ辞書の辞書項目と対応しており、主な記載事項として活用型や深層格 / 表層格の対応等があります。図 27に文生成用辞書の記述例を示します。

参考文献

[1] ICOT 第 2 研究室,“ LTB ソフトウェア仕様書 - 文生成編 -”

[2] ICOT 第 2 研究室,“ LTB 操作説明書 - 文生成編 -”

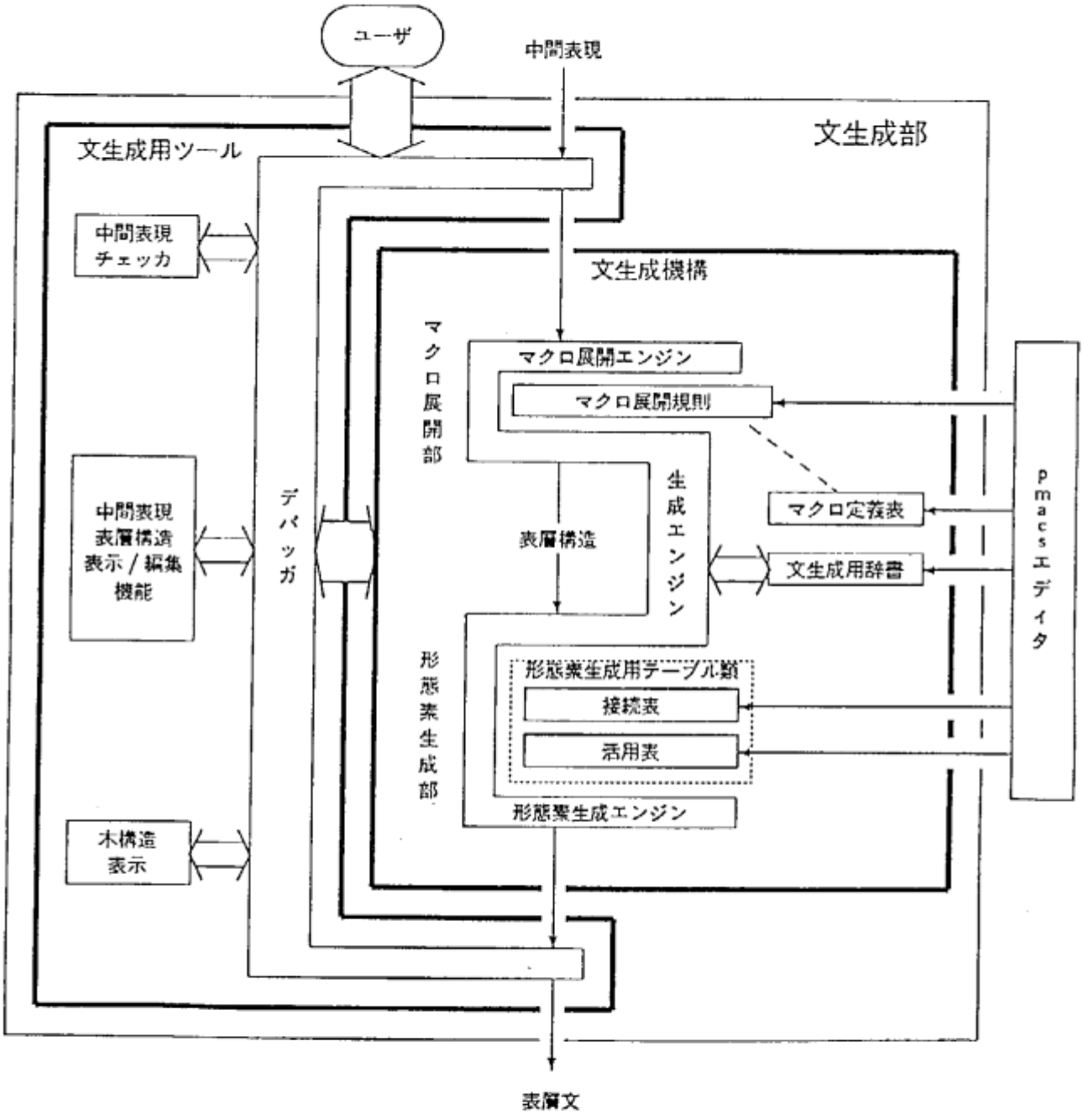


図 18: 文生成部の全体構成

```
{関係 /{語彙 / 頼る},
 ロール /{行為者 /{補語 /{語彙 / 人間}, 取り立て / は},
          終点 /{補語 /{被修飾 /{語彙 / 恵み},
                        連体修飾 /{語彙 / 自然}}}},
 モーダル /{ムード /[必然]}}
```

図 19: “人間は自然の恵みに頼らなければならない”に対応した標準マクロ表現

```
{head{lex/ なければならない},
 comp/{head/{head/{lex/ 頼る},
             comp/{head/{lex/ 恵み},
                   comp/{head/{lex/ の},
                         comp/{lex/ 自然}}}},
       comp/{head/{lex/ は},
             comp/{lex/ 人間}}}}
```

図 20: “人間は自然の恵みに頼らなければならない”に対応した基本表現

```
[[[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]}]]
```

図 21: “自然に頼る”に対応した表層構造

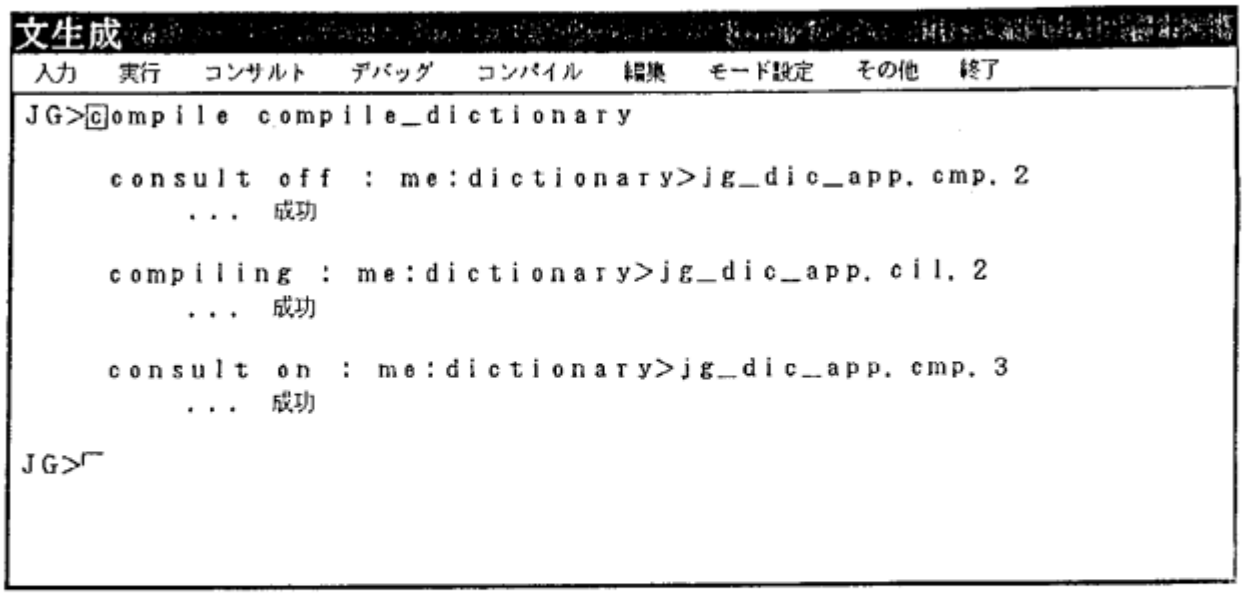


図 22: 文生成用ツールのトップウインドウ

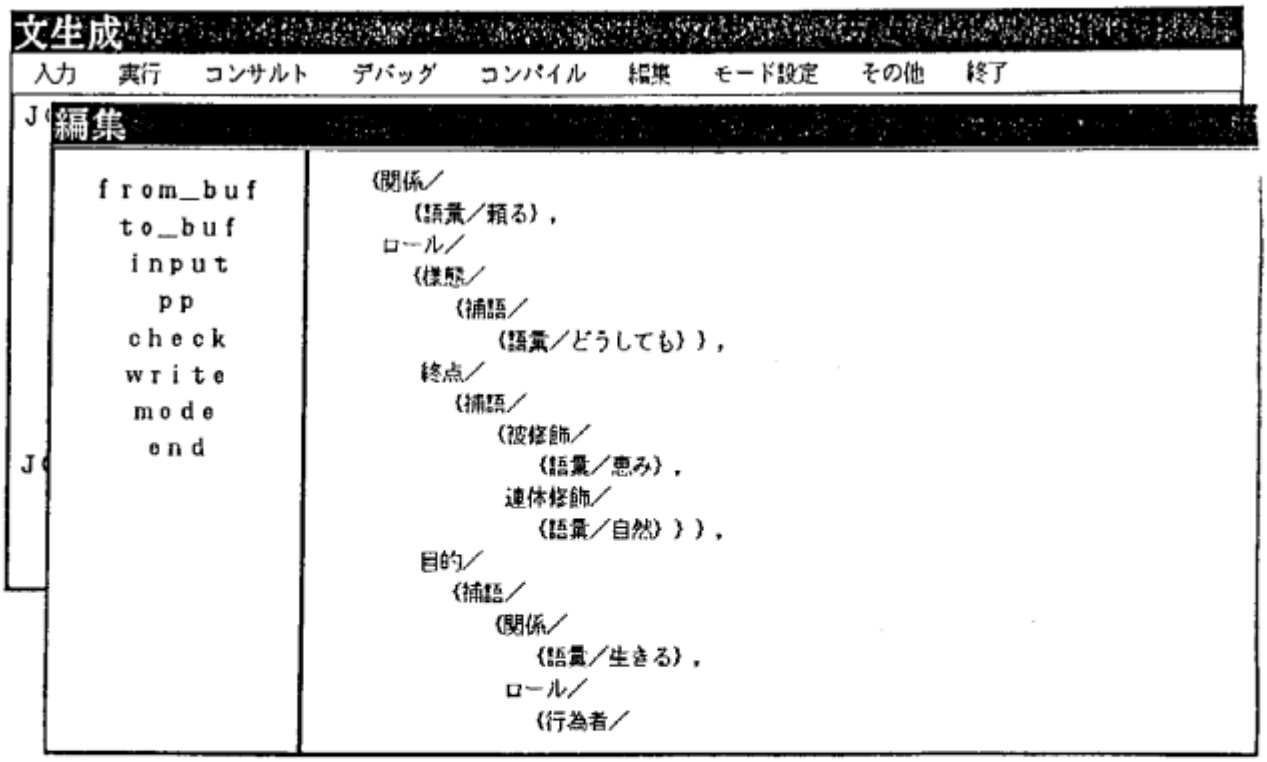


図 23: 中間表現エディタのウインドウ

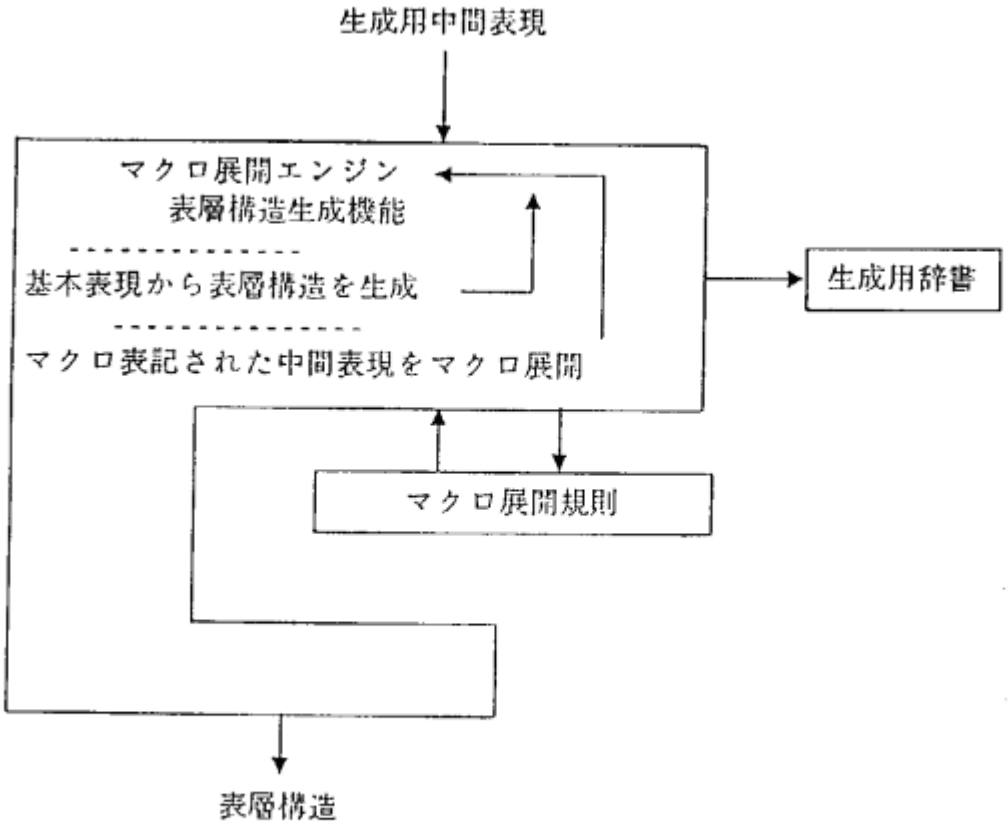


図 24: マクロ展開部の構成

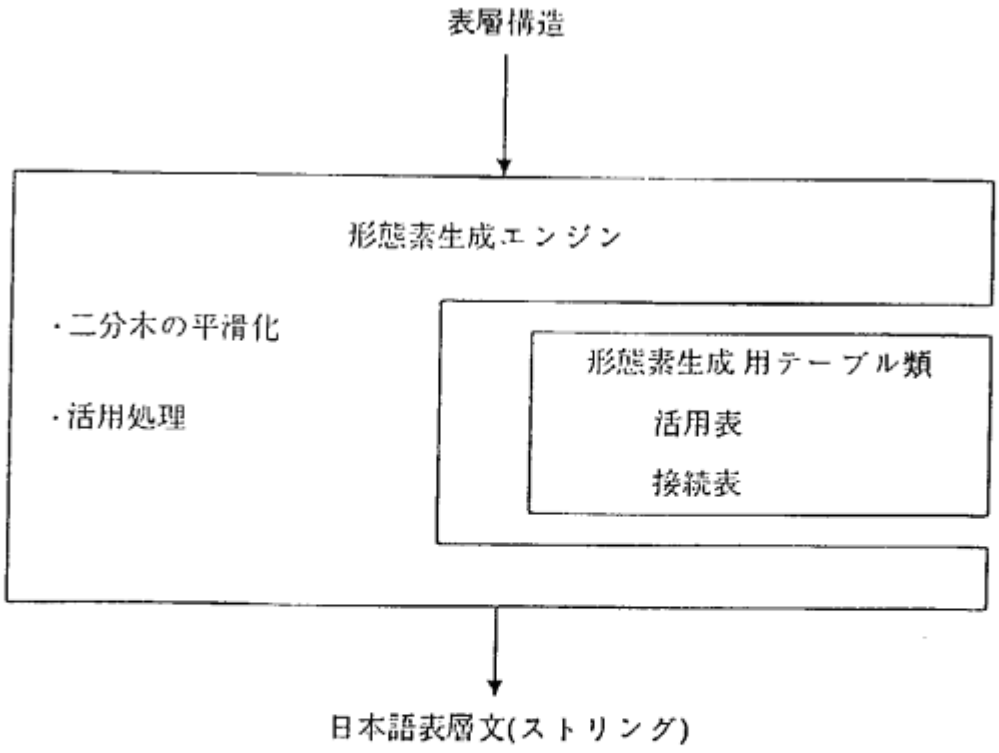


図 25: 形態素生成部の構成

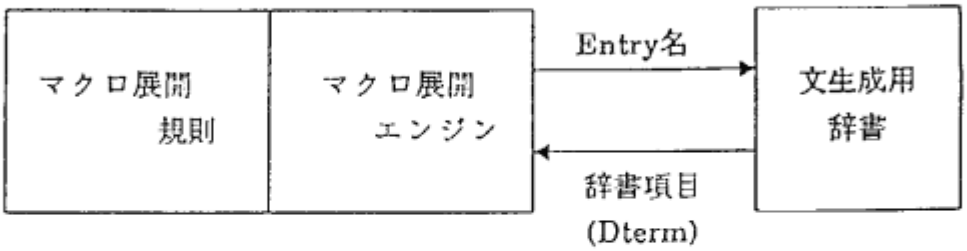


図 26: 文生成用辞書

```
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]}

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]})
```

図 27: 文生成用辞書の記述例

第 VI 部
マスタ辞書

14　概要

- **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が記載されている.
- LTB マスタ辞書の内容は, **アクセスメソッド**によって参照することができる.
- LTB において, 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.
- 現在の版において, LTB 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

15　構成と内容

- LTB 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. **辞書フォーマット**は, 品詞ごとにそれぞれ違った記載項目をもっている.
- LTB マスタ辞書の内容は LTB の文法に準拠している. この文法は, 本研究室が LTB の開発のために, 日本語文法研究の成果をふまえて独自に作成したもので, 従来の文法とは異なる部分も多い. たとえば, この文法では, 名詞を普通名詞, 形容動詞性名詞, サ変動詞性名詞の 3 つに分類している.
- 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**の対が並んだ形をしている. これは, CIL の部分項 (PST) の形式と等しい. 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列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仮定される.

- マスタ辞書の記載例を以下に示す.

  - 動詞【集まる】

(見出し語レコード)

```
{
  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 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］
}．

－ 普通名詞【空気】

（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］
}．

（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／［］，
量語用法／［］，
可算性／不明
}．

－ サ変動詞性名詞【加工】

（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］
}．

（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Ｖ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／［性］,
      畳語用法／［］,
      可算性／不明
    }.
```

－ 形容詞【新しい】

（見出し語レコード）

```
    {
      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 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／[],
  畳語用法／[]
 }.
```

16 アクセスメソッド

- LTB マスタ辞書には，その内容を容易に検索，利用するためのアクセスメソッドが用意されている．

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

- はじめに，あらかじめ定められた構文に従って作成されたマスタ辞書のテキストファイルを，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．

- マニピュレータの実行の様子を見るには，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，検索を行うと，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．

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

- 検索条件は，「2. 構成と内容」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．

- マニピュレータの実行例を以下に示す．

(例 1) テーブル "midasi" から，品詞が連体詞のものを検索し，その表層表現を S1 に，検索結果であるレコード全体を R1 にユニファイする．

```
?- :get_entry(Obj,midasi,''([品詞 / 連体詞, 表層表現 /S1]),R1).
R1= {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]},
S1= あくる
```

(例 2) テーブル "gogi" から，絶対自動かつ意志性のない単語を検索し，そのレコード全体を R2 にユニファイする．

```
?- :get_entry(Obj,gogi,''([自他 /[絶対自動], 意志性 / なし]),R2).
R2= {識別番号 / 動詞 000020A, 深層格 / (exp,par),
     能動表層格 / (exp: が,par: ∨(に, と)), 引数 / (exp:A,par:B),
     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 / あり, 授受表現 / なし],
     相 /[瞬間], 自他 /[絶対自動], 可能動詞化 / 不可, 意志性 / なし,
     派生名詞 /[]}
```

(例 3) テーブル "gogi" から，接尾辞が「達」のものを検索し，その表層表現を S3 に，検索結果であるレコード全体を R3 にユニファイする．

```
?- :get_entry(Obj,gogi,''([接尾辞 /[達], 識別番号 /S3]),R3).
R3= {識別番号 / 名普 001560A, 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 / 不可, 接頭辞 /[], 接尾辞 /[達],
     助数詞 / 羽, 畳語用法 /[], 可算性 / 数},
S3= 名普 001560A
```

第 VII 部
LTB−shell

今回のリリースには LTB−shell は含まれておりません。ここでは現在開発中の *LTB − shell* について説明します。

17 LTB−shell とは

LTB の各ツールはそれぞれ固有のユーザインタフェイスを備えており、それらの環境を用いることにより、ユーザは各ツールを使いこなすことができます（図 28）。

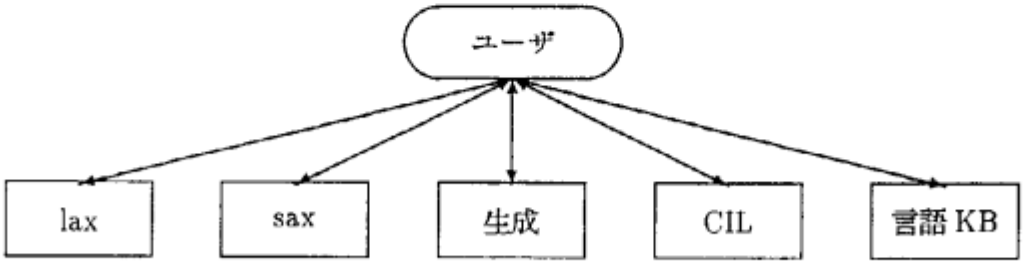


図 28: 従来の LTB ユーザインタフェイス

一方、LTB の各ツールをあたかも一つのコマンドのように取り扱い、複数のコマンドを自由に組み合わせたり、コマンドの入出力をコントロールしたいことがあります。例えば、

「lax の解析結果を sax に渡して、その結果をファイルに保存したい」

といった使い方は、現在の LTB ツールでは簡単にはできません。

LTB−shell は、このように、LTB の各ツールをあたかもコマンドのように取り扱うことにより、複数のツールを組み合わせたり、ツールの入出力のコントロールを可能にするユーザインタフェイスです（図 29）。

言い換えると、LTB−shell は既存の LTB ツール群や、ユーザが作成する新たなツール群を効率良く組み合わせて使うための統一されたユーザインタフェイスです。

LTB−shell の操作環境としては、従来のコマンド入力方式の他に、PSI のグラフィックス機能を生かした、ダイレクトマニピュレイション方式を考えています。ここでは、コマンド入力方式を例に説明を進めます。

18 LTB−shell の特徴

LTB−shell を利用すると以下のことが可能になります。

- 各ツールの利用方法の詳細を知らなくても、コマンド形式で主な機能が使えるようになります。

```
LTB> lax −g grammar1
  ; lax を起動する。文法オブジェクトとして grammar1 を指定
```

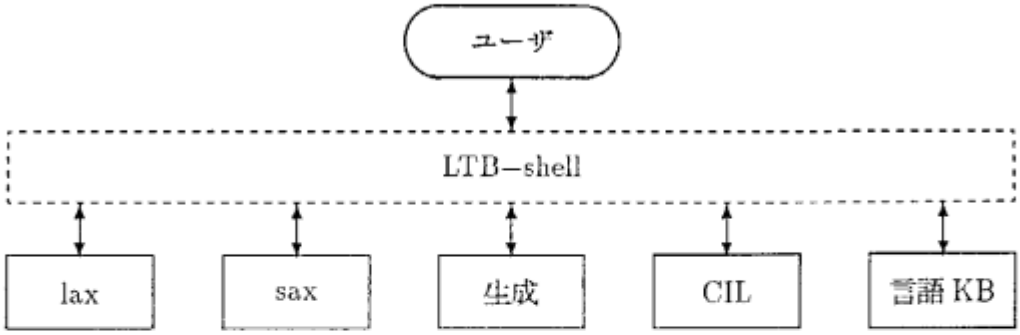


図 29: LTB−shell を介したユーザインタフェイス

- ツールの入出力の対象を自由に設定できます。

```
LTB> lax { file1 } output.txt
  ; ファイル file1 の内容を lax に入力し、解析結果をファイル
  ; output.txt に出力する。
```

- 複数のツールを組み合わせて用いることができます。

```
LTB> lax { file1 | sax
  ; ファイル file1 の内容を lax に入力し、解析結果を sax に入力する。
```

- 複数のジョブを並列に実行できます。

```
LTB> lax −g grammar1 | sax } file1 &
LTB> lax −g grammar1 | sax } file2 &
LTB> lax −g grammar1 | sax } file3 &
  ; 一つのジョブの終了を待たずに、次のジョブを並列に
  ; 実行させられます。
```

19 LTB−shell の利用方法

19.1 コマンドの起動方法

シェルに対する命令は、コマンドの形式で与えます。その際、コマンドの後に引数を与えることにより、コマンドへの入力データや細かい指定を与えることができます。

```
LTB> sax −g "grammar1"
  ; sax に「自然を守る」という入力を与える。文法オブジェクトとし
  ; て grammar1 を指定。
```

19.2 コマンドの入出力の指定方法

コマンドに対する入出力を指定する方法としては、コマンドの引数を用いて指定する方法以外に、標準入出力を用いることができます。

19.2.1 リダイレクション

コマンド { 入力対象
コマンド } 出力対象

入出力の対象としては、ファイル、テキストバッファ、ウィンドウ が与えられます。

```
LTB> lax { file1 } file2
  ; lax にファイル file1 の内容を入力として与える。
  ; その結果をファイル file2 に出力する。
```

19.2.2 パイプ

コマンド1 | コマンド2

コマンド1の出力がコマンド2への入力になります。

```
LTB> lax -g grammar1 | sax -g grammar2
  ; lax の解析結果を sax に受け渡す
```

19.3 ジョブの管理

ユーザがシェルに与えたコマンドのひと固まりをジョブと呼びます。時間のかかるジョブをシェル上で実行していると、結果が帰ってくるまで次のコマンド入力ができなくなってしまいます。そこで、シェルとジョブの実行を非同期に行う機能（バックグランド）を用います。ジョブをバックグランドで実行することにより、複数のジョブを並列に実行できるようになります。

```
LTB> lax −g grammar1 | sax } file1 LTB> lax −g grammar2 | sax } file2
    ; 一行目では、lax と sax を組み合わせて実行し、その結果を file1
  ; に出力させています。& によりこのジョブはバックグランドで
  ; 実行され、実行の終了を待たずに二行目の入力を受け付けます。
  ; 二つ目のジョブはフォアグランドジョブなので、実行が終了する
  ; まで、シェルへの入力はできません。
```

19.4 その他の機能

以上の機能がシェルの核になる機能です。これらの機能を生かすために次のような機能があります。

ヒストリ機能： コマンド入力履歴を保持し、ユーザがそれらを再利用するための機能

エリアス機能： 長いコマンドやコマンドの列に対し、ユーザが好きな別名を与える機能

```
LTB> alias slax 'lax −input !* — sax'
LTB> slax " 自然が破滅される "
```

19.5 LTB−shell の操作環境

LTB−shell の操作環境は、コマンド入力方式をベースにし、その上にグラフィク指向のダイレイクトマニピュレイション方式の操作環境を構築する予定です。

19.6 SIMPOS シェル からの LTB の利用法

SIMPOS にはコマンド入力方式、リダイレクション機能、パイプ機能が用意されています。現在の LTB の各ツールは直接 SIMPOS-shell 上では使えませんので、使用可能にするためのカスタマイズの例を紹介します。

19.6.1 ツールの修正

SIMPOS-shell 上で使うためには、各ツールに皮を被せる必要があります。例えば、LAX に皮を被せて demoLAX を作る方法は以下のようになります。

```
class demoLAX has
nature

    demo##io,
    as_program;

instance

    :goal(Obj) :-
        :getArguments(Obj,Args),
        goal(Args,Obj),
        :closeIO(Obj);

local

    goal(['-',g,LAXName],Obj) :-
        :object(#demo##tabOBJECT,LAXName,LAXObj),
        lax(LAXOBJ,Obj);

    lax(_,Obj) :-
        :endOfInput(Obj),!;
    lax(LAXObj,Obj) :-
        :getL(Obj,Input),
        :analyze(#demo##lax,LAXObj,Input,Output),
        :putT(Obj,Output),!,
        lax(LAXObj,Obj);

end.
```

19.6.2 login.com の修正

シェルのコマンド名とツールオブジェクトを対応させる必要があります。例えば、lax という名前で、上の demoLAX を起動する場合、以下の定義を login.com に加える必要があります。

```
shell_transient:-
       [{lax,        'demo##demoLAX'}].
```