# 汎用依存構造処理モジュール KuraLang

岩倉友哉 *1 乾健太郎 *2 高橋哲朗 *2 飯田龍 *2 藤田篤 *2
*1 九州工業大学大学院情報工学研究科情報科学専攻
t_iwa@pluto.ai.kyutech.ac.jp
*2 奈良先端科学技術大学院大学 情報科学研究科
{inui, tetsu-ta, ryu-i, atsush-f}@is.aist-nara.ac.jp

## 1 はじめに

近年，高精度な構文解析器[1] が開発され，意味解析研究や言語知識の獲得といった様々な自然言語処理研究において幅広く利用されるようになってきた．構文解析で得られた構文木に対して別の処理を施したい場合，必要な処理を手続き的に実装するのが一般的であるが，必ずしも効率的な開発方法ではない．木に対する照合や書き換えの処理は，複雑な手続きの塊りに肥大化しやすく，プログラムの可読性や再利用性を下げる原因となる．

この問題は，(注釈付きの) 木の照合や書き換えを扱える適当なプログラミング言語やツールを我々が持ち合わせていないことに少なからず起因している．文字列については，照合や書き換えのパタンを表現する正規表現が広く普及し，正規表現をプログラム中に埋め込める *Perl* や *Ruby* のような言語や *grep* などのツールも発展をとげた．しかし，木となると，そのパタンを表現する広く普及した記述形式も，それを扱えるプログラミング言語やツールも存在しないのが現状である[2]．*Perl* や *Ruby* で文字列処理を行うのと同様の感覚で木の処理を記述できれば，「構文解析以降」の処理に関する研究・実験の効率を大きく改善できるのではないか．こうした動機が本研究の背景である．

本稿では，汎用依存構造処理モジュール KuraLang について報告する．KuraLang は，形態素依存構造木に対する照合・書き換えを扱えるモジュールであり，以下の様な特徴を持つ．

(1) 形態素依存構造木に対する照合・書き換えの手続きを，「$N1$ が $N2$ に $V$ られる $\rightarrow$ $N2$ が $N1$ を $V$」のような平易な表現で記述することができる．

(2) *Perl*，*Ruby* といった様々なプログラミング言語に直接組み込んで利用することができる．

以下，KuraLang の概要を順次説明する．

[1] http://cl.aist-nara.ac.jp/~taku-ku/software/cabocha/ http://pine.kuee.kyoto-u.ac.jp/nl-resource/knp.html

[2] 木のパタンは，文脈自由文法のような文法表現で規定するのが最も一般的である．しかし，木に対する処理を記述する手段として文法表現をコード中に埋め込めるプログラミング言語やツールは，筆者の知る限りは存在しない．

## 2 KuraLang

### 2.1 データ構造

既存の解析器の多くは文節ベースの係り受け構造を出力する．しかし，文節ベースの依存木では，文節間の構造に対する照合・書き換えと文節内の形態素列に対する照合・書き換えを別々に扱わなければならない．そのため，両者が組合わさる場合の処理が扱いにくいという問題点がある．そこで，KuraLang では，文の構造を表現するデータ構造として，形態素をノードとする依存構造木 MDS (Morpheme-based Dependency Structure) を採用した[3]．

KuraLang では，(a) 形態素・構文解析の結果，(b) 照合・書き換えパタン，の両方を MDS を用いて表現する．MDS では，統語情報の他，必要に応じて種々の意味情報を素性として注釈づけすることが可能である．

現在，MDS は構文解析器の結果を基に，文節の主辞を決定し，係りもとの文節の最後の形態素が，係り先の文節の主辞に係るという取り決めで作成している (図 8 参照)．構文解析器は CaboCha と KNP をサポートしている．

### 2.2 実現方法

KuraLang は，Kura[1, 9] で開発された技術を基に，MDS に対する処理を LiLFeS [3] (以下 MDS 処理エンジン)，その他の部分を C++ (以下 KuraLang クラス) で実装を行った．KuraLang クラスは，MDS 処理エンジンを別プロセスとして動作させ，MDS 中の各ノードへの識別子を使って，MDS 処理エンジン内に保持されている MDS にアクセスする [4](図 1 参照)．

KuraLang は，C++で作成された KuraLang クラスを swig[5] を用いて拡張することで様々なプログラミング言語において利用可能となる[6]．以下，「MDS 処理の記述方法」と「MDS 処理の実行」を順に説明する．

## 3 MDS 処理の記述方法

本節では，MDS 処理に用いる記述方式「SSR パタン」を 3.1 節で紹介し，「SSR パタンの処理方法」について

[3] 機械翻訳における研究 (たとえば，[5, 2]) が示すように，依存構造は句構造に比べて文の構成素の移動や置換を行いやすいという利点がある．

[4] これにより，LiLFeS の型付素性構造処理機能を活用できる

[5] http://www.swig.org/

[6] 現在，*Perl*，*Ruby* での動作を確認している．

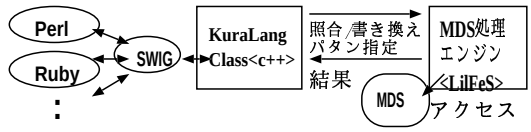


図 1: KuraLang のイメージ

3.2 節で説明する．

## 3.1 簡易依存木書き換えパタン (SSR パタン)

KuraLang では，Kura[1, 9] の規則記述方式である SSR (Simplified dependency-Structure Rewriting) パタンにより，MDS の照合・書き換えが行える．また，照合に正規表現が利用でき，必要に応じて変数に MDS 中のノードを束縛し，プログラムの別の場所で参照することもできる．

### MDS 照合パタン

MDS の照合パタンは以下の要素からなる．

- **変数：**図 2 の (R1) の “N” は名詞，“V” は動詞を意味する変数である．これら 2 つの変数はデフォルトで定義されている．助動詞や副詞といったその他の品詞も，ユーザがマクロ定義をすることで変数として利用できる．“→” の左側は照合パタンを表し，変数にマッチしたノードは “→” の右側の書き換えパタンにおいて参照できる．一つのパタンで同じ種類の変数を複数使いたい場合は，(R2) の，“N1”，“N2” のように変数に数字を添えることで別の変数と判断される．
- **形態素属性の指定：**品詞，意味属性などの特定の形態素属性の指定は，“(”，“)” の中で行う．(R3) の例では，“Thesaurus” の意味属性が “人名” であるノードとマッチする．
- **形態素属性の否定：** “!” は特定の形態素属性の値を否定する際に用いる，(R4) は “Thesaurus” の意味属性が “人名” ではないノードとマッチする．
- **依存構造情報の指定：**依存構造の情報を付与する場合は，(R5) のように係り先のノードを指定する．この指定を行われていない部分は，SSR パタンの形態素・構文解析の結果より依存構造を決定する．
- **例外：** “!!” は例外パタンを指定する場合に用いる．(R6) は，“決して” の係り先である “V” の係り先が “ない” 以外のノードである場合にマッチする．

### MDS 書き換えパタン

書き換えパタンも照合パタンと同じく SSR パタンで記述する．書き換えは，→ 左側のパタンとの照合が成功した場合に行われ，書き換えパタンにおいて指定された形態素属性は，新たな属性として追加されることになる．(R7) の書き換えが行われた場合は，ノード “N1” と “N2” の “CASE” に “ガ” と “ニ” が新たな属性として追加される．

### 正規表現

KuraLang では，形態素列と部分木に対して正規表現が利用できる．

- **VMSs (Vertical Morpheme Sequences)**
  **VMS** は，ノードの縦の並びに対する正規表現である．(R1) の V と “ない” の間にはアスペクト，ヴォイス等を意味する形態素列が存在する場合がある．(R1) では，(1)「野菜しか食べない.」，(2)「野菜しか食べられない.」，(3)「野菜しか食べていない.」のうち，(1) の文にしかマッチしない．一方，(R8) では，変数 “V” に属性 VMS:“suffix” (図 3 参照) を指定することで (1) から (3) の全ての文にマッチする．
- **HMSs (Horizontal Morpheme Sequences)**
  記号 “?” で与えられる **HMS** は，オプショナルな部分木の存在を指定する表現である．(R9) は，“N は” にマッチする部分木が存在しなかった場合は (R9-a) と同じ意味になり，存在する場合には，(R9-b) と同じ意味となる．(R9) は,「太郎は転んだが，泣かなかった.」と「転んだが，泣かなかった.」の両方の文にマッチし,「太郎は転んだ．しかし，泣かなかった.」,「転んだ．しかし，泣かなかった.」の書き換え結果が生成される．

### 束縛変数の参照

KuraLang では，“[”，“]” により指定したノードをプログラムの別の場所で参照することができる．例えば，(R10) が,「花子が饅頭を食べた.」という文にマッチした場合は,「花子」,「饅頭」,「食べた」の 3 つのノードが参照可能となる．

```
(R1) : N しか V しない → N だけだ.
(R2) : N1 だけが N2 ではない →
           N1 ばがりが N2 ではない.
(R3) : N(Thesaurus:人名)
(R4) : N(Thesaurus!:人名)
(R5) : N1 は (M:1)N2 で成り立つ (1)
(R6) : 決して V する!!決して V しない
(R7) : N1 が N2 に V させられる →
           N2(CASE:ニ) が N1(CASE:ガ) を V する
(R8) : N だけ V し (VMS:suffix) ない → N だけだ.
(R9) : N は? V1 したが，V2 した。 →
           N は V1 した。しかし，V2 した。
(R9-a) : V1 したが，V2 した。 →
           V1 した。しかし，V2 した。
(R9-b) : N は V1 したが，V2 した。 →
           N は V1 した。しかし V2 した。
(R10) : [N] が [N2] を [V する]
```

図 2: SSR パタンの例

```
def_vms(suffix,[(付属語活用あり,*)]).
def_vms(particle, [(助詞, *)]).
```

図 3: VMSs の例

## 3.2 SSR パタンの処理方法

KuraLang の内部では，SSR パタンの他に，SR パタン, SP オペレータという二つの階層が存在する．こ

れらは記述の抽象度の観点から図 4 のような関係にある．①から②への変換はトランスレータにより行われる．トランスレータは，SSR パタンに指定された情報と SSR パタンの形態素・構文解析の結果より，照合パタン，書き換えパタンの依存構造木の対である SR パタンを生成する．② から③への変換では，コンパイラが，SR パタンに対応する，最適な照合・書き換えオペレータ (SP オペレータ) 列を生成する．MDS へのアクセスは③のレベルの照合・書き換えオペレータ列により行われる．詳しくは，[1, 9] を参照願いたい．

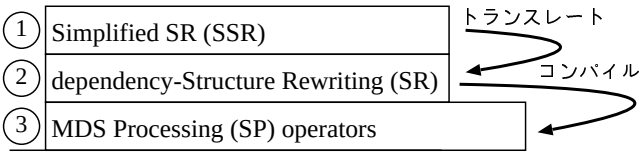


図 4: 変換階層

# 4 MDS 処理の実行

KuraLang では，様々な用途で，効率的な処理が行えるように，3 種類のパタン適用方法を用意している[7]．

## 4.1 基本的な照合・書き換え

MDS の基本的な照合・書き換えはメソッド **match**，**rewrite** により行える．図 5 の 1) は，「N しか V しない」というパタンとの照合，2) は，「N しかない」から「N だけだ」への書き換え例である．

## 4.2 照合・書き換えパタン集合の適用

MDS に対し，大規模な数の照合・書き換えパタンを適用する場合は，それらをパタン集合として作成することで，効率的な適用が行える．パタン集合は，各パタンの最も出現頻度が低いノード[8]を各パタンへのインデックスとして持つ．パタン集合の適用では，対象の MDS が，パタン集合のインデックスから適用するパタンを選別し、適用を行う．パタン集合の適用は，メソッド **transfer** により行う．図 5 の 3) は，パタン集合 “negation” に属する照合・書き換えパタンを適用する例である．

## 4.3 MDS 集合に対する照合

コーパスなどから作成した大規模な数の MDS に対する照合を行う場合は，それらを MDS 集合として作成することで，効率的な照合が行える．MDS 集合は，MDS の各ノードの表層文字列と依存構造に基づく品詞バイグラムを各 MDS へのインデックスとして持つ．MDS 集合に対する照合では，照合パタンが，MDS 集合のインデックスから候補の MDS を決定し，照合を行う．MDS 集合への照合はメソッド **matchMdsSet** により行う．図 5 の 4) は，MDS 集合 “kyoto” に対し，「N しかない」というパタンを持つ MDS を検索する例である．

[7]以降の例は，*Perl* で記述する．変数$k は KuraLang クラスオブジェクト，$MDS は入力の MDS を意味する．

[8]あらかじめ計算された単語ユニグラム情報に基づき決定する．

```
1) $k→match($MDS,'N しか V しない');
2) $k→rewrite($MDS,'N しかない', 'N だけだ');
3) $k→transfer($MDS,'negation');
4) $k→matchMdsSet('kyoto','N しかない');
```

図 5: 照合・書き換えパタン適用メソッド

# 5 利用例

## 5.1 動詞のガ格，ヲ格の名詞の抽出

図 6 は，与えられた MDS から動詞のガ格とヲ格の名詞を抽出し，印字するプログラムの例である．一行目で $k に KuraLang クラスのオブジェクトが代入され，2 行目で$MDS に「浩が父親がくれた本を読む.」の $MDS$ へのポインタが渡される．3 行目で **match**(4.1 節参照) により照合を行う．ノードは “[”，“]” で指定された順番に束縛されており，メソッド **getBoundNode** により束縛したノードへのポインタが得られる．この場合，4 行目で$N1 に「浩」，5 行目で$N2 に「本」にあたるノードへのポインタが返される．ノードへのアクセスは **getFt** により行う．6，7 行目にてそれぞれのノードの表層文字列へとアクセスし，返り値を出力している．

```
# KuraLang オブジェクト作成
1:$k = new KuraLang;
# MDS 作成
2:$MDS = $k->makeMds(' 浩が父親がくれた本を読む ');
# 照合
3:$k->match($MDS,'[N1] が [N2] を V する');
# 束縛されたノードのポインタを得る.
4:$N1 = $k->getBoundNode($MDS, 1);
5:$N2 = $k->getBoundNode($MDS, 2);
# 束縛されたノードの表層文字列,「浩」,「本」を出力.
6:print $k->getFt($MDS,$N1,'[SYN\,WORD_FORM\]');
7:print $k->getFt($MDS,$N2,'[SYN\,WORD_FORM\]');
```

図 6: 動詞の ' が格 ' と ' を格 ' にあたる名詞を表示する例

## 5.2 言い換えパタン集合の適用

図 7 は，2 行目の文に対し，4 行目の **transfer**(4.2 節参照) でパタン集合 “negation” により言い換えを行う例である．仮に，図 2 の $(R1)$，$(R2)$ が “negation” に属しているとすると，1 回目は “お金しかない” の部分が “お金だけだ” に言い換えられ，2 回目は，1 回目に言い換えられた文の “お金だけが人生ではない” の部分が “お金ばかりが人生ではない” と言い換えられる．

```
1: $k = new KuraLang;
2: $in = '「僕にはお金しかない」と言っていた彼が「お
金だけが人生ではない」と言っている．';
3: $MDS = $k->makeMds($in);
#パタン集合の適用.
4: while($k->transfer($MDS, 'negation'){
5:  $k->printMds($MDS); }; # 言い換え後の MDS を出力
```

図 7: 言い換えパタン集合適用例

## 5.3 コーパス検索ツール

図 8 は， MDS に対する検索を SSR パタン (3.1 節参照) により行えるコーパス検索ツールである．インター

フェースは，*ruby/Tk* で作成し，検索部分は，4.3 節で紹介した機能により作成した．

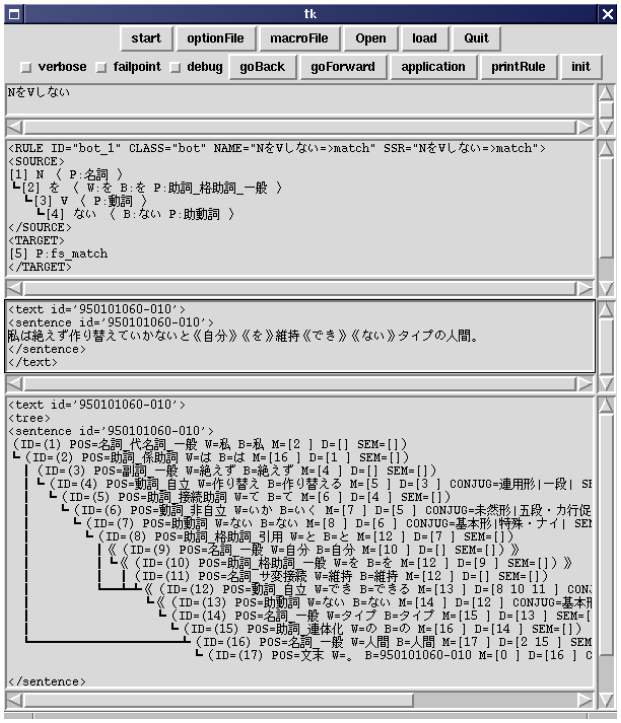


図 8: コーパス検索ツール

## 6 先行研究

文の照合・書き換えは機械翻訳の分野で長年研究されており，今までに数多くのシステムやツールが開発されてきた．例えば，CoGenTex [9]では，MTT[4] の SSyntSs から ConSs (形態・統語レベルから概念レベル) 間の変換処理および，DSyntSs (意味レベル) における他言語間トランスファーを一様に扱えるモジュール[10] を開発しており，文献 [2] でそのモジュールを用いた多言語間翻訳／生成システムを開発するためのフレームワークについて報告している．

文献 [7] では，機械翻訳システム **KANTOO**[6] のモジュールである解析器と生成ツールを利用し，制限言語テキストから [8] の知識表現を生成する研究を試みている．このように，機械翻訳の分野では高機能なシステム／ツールが開発され，同分野だけの利用にとどまらず，いくつかの応用も行われている．ただし，これらの試みは KuraLang と以下の点で異なる．

- 機械翻訳は原言語のすべてを目的言語へと変換するタスクであり，上述のツールは文全体の書き換えを前提に設計されている．一方，KuraLang は，部分的な照合・書き換えにおいても利用でき，正規表現の利用，ノード束縛といった照合時における種々の機能を用意している．
- システムを構成するツールを再利用する場合と異なり，KuraLang は，様々なプログラミング言語に直接組み込んで使用できるので，各言語の特徴と組み合わせた利用が容易に行える．

[9] http://www.cogentex.com/

[10] DSyntSs から ConSs への変換といった，特定のタスク毎に必要な知識や変換規則をモジュール上に実装し，インスタンスを作成する形での利用であり，KuraLang のように直接コードに埋めこんで利用する形ではない．

## 7 おわりに

形態素・依存構造に対する処理を平易な記述で実装でき，様々なプログラミング言語に組み込んで使用できる依存構造処理モジュール KuraLang を紹介した．本モジュールは，研究目的に無償で配布している．詳しくは，
*http://cl.aist-nara.ac.jp/lab/kura/KuraLang/*
を参照願いたい．

## 謝辞

実装にあたり，東京大学辻井研究室で開発されたプログラミング言語 LiLFeS を使用させていただき，度重なる質問に丁寧に回答くださいました．形態素・構文解析に奈良先端大松本研究室で開発された ChaSen と CaboCha，および京都大学言語メディア研究室で開発された JUMAN，KNP を使用させていただきました．深く感謝申し上げます．

## 参 考 文 献

[1] http://cl.aist-nara.ac.jp/lab/kura/doc/

[2] Lavoie, B., Kittredge, R., Korelsky, T. and Rambow, O. A Framework for MT and Multilingual NLG Systems Based on Uniform Lexico-Structural Processing. In *Proceedings of ANLP/NAACL 2000*.

[3] Makino, T., Torisawa, K. and Tsujii, J. LiLFeS - Practical Programming Language For Typed Feature Structures. In *Proceedings of Natural Language Pacific Rim Symposium '97*, 1997.

[4] Mel'čuk, A. I. and Polguère, A. A Formal Lexicon in the Meaning-Text Theory (OR How to do Lexica with Words). Computational Linguistics, Volume 13, Numbers 3-4, pp. 261-274,1987.

[5] Meyers, A., Yangarber, R. and Grishman, R. Alignment of Shared Forests for Bilingual Corpora. In *Proceeding of the 16th International Conference on Computational Linguistics*, pp. 460-465, 1996.

[6] Nyberg, E. and Mitamura, T. The KANTOO Machine Translation Environment. In *Proceedings of AMTA 2000*.

[7] Nyberg, E., Mitamura, T., Baker, K., Svoboda, D., Peterson, B. and Williams, J. Deriving Semantic Knowledge from Descriptive Texts using an MT System. In *Proceedings of AMTA 2002*.

[8] OWL and the IODE. The Ontology Works White Paper. Available at http://www.ontologyworks.com/whitepaper.pdf. August (2001).

[9] Takahashi, T., Iwakura, T. Iida, R., Fujita, A. and Inui, K. KURA: A Transfer-Based Lexico-Structural Paraphrasing Engine. *The 6th Natural Language Processing Pacific Rim Symposium, Workshop on Automatic Paraphrasing: Theories and Applications*, 2001.