

木構造変換言語 Allegro-C

4S-1

川越 睦、青山 昇一、杉村 領一

松下電器産業(株) 情報通信関西研究所

1. はじめに

機械翻訳用に開発した木構造変換用言語、Allegro-Cの言語仕様と実装手法を紹介する。

Allegro-Cは木構造変換規則を記述した文法と、これをC言語プログラムへ変換するコンパイラ等から構成される。Allegro-Cの主な特徴は、

- 1) 木のノードが持つ属性/属性値をC言語プログラムへの変換過程で内部の数値へ変換し、得られたプログラムにより属性の操作を高速に行なう点、
- 2) C言語とのインタフェイスを持ち、木構造変換文法中にC言語の呼び出しを記述できる点、の以上である。

コンパイラにより生成されたCプログラムはユーザの作成したC言語プログラムと結合することにより、所望の変換動作をする。

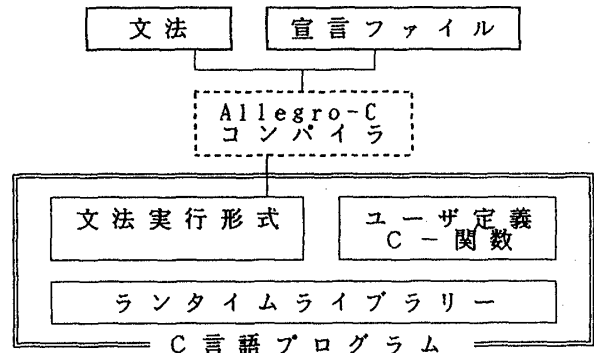
LISP版(LISPとのI/Fのみ)とC版の簡単な比較結果も報告する。

2. Allegro-Cのシステム概要

図1へAllegro-Cを用いた機械翻訳システムの構成を示す。

Allegro-Cでは、文法で用いる属性とその値を宣言ファイルへ予め登録する。コンパイラはこの宣言を読み込み、属性とその値を数値化し、生成するプログラムでこの数値を処理させる。文法記述用の基本的な関数はランタイムライブラリとして用意されているが、ユーザ専用の関数も自由に用いることができる。

できる。



- 図.1 -

3. 文法記述形式

3. 1. 属性宣言ファイル

Allegro-Cでは、文法で扱うノードの全ての属性情報を属性宣言ファイルへ宣言する。これにより属性操作および使用領域の効率を向上させている。属性の宣言は下記のように行う。

属性名:型:{属性値₁, ..., 属性_n};

型としては、属性値の取り得る値の決まっている属性(C型)、整数値型、文字列型の3つが許される。C型の場合取り得る属性値をすべて宣言する必要がある。

3. 2. 文法ファイル

文法ファイルに記述する文法の形式は、京都大学で開発されたGrade[1]の

Tree to Tree Transducer Allegro-C

Mutsumi Kawagoe, Shoichi Aoyama, and Ryoichi Sugimura

Matsushita Electric Industrial Co.LTD.

仕様を参考にしており、書換え規則そのものを定義するルールと、この書換え規則の適用を制御するサブプログラムの2種類の規則を基本に構成される。さらに、C言語プログラムと木のノードのデフォルト属性をこのファイルに記述することができる。

1) ルール

書換え規則宣言は、規則を適応すべき木構造との照合動作を指定する部分と、このマッチングの条件を指定する部分と、マッチングが成立した時点で生成すべき木構造パターンを宣言する部分から成る。計算量が高くなるため、書換え動作後のバックトラックは許していない。但し、照合動作が高速に行われるので、ルール適用時に徹底的に条件を詳細に記述することができる。これにより決定的な動作による変換を意味のあるものに行っている。

2) ルール制御 (サブプログラマ)

サブプログラマでは複数のルールをまとめて、その適用を制御する。

ルール群の適用方法としては、複数のルール群を順次適用し、成功した時点で適用を中止する方法と、ルール群全ての規則を適用する方法がある。

また、一群のルールの適応結果に基づいてその後適用すべきルールを切り替えるIF文も使用できる。

3) デフォルト属性宣言では、前記C型ノードに対してそのデフォルト値を宣言できる。

4) C言語宣言では、#{で始まり、}で終る部分にCプログラムを記述できる。

4. コンパイラの実装

文法の解析にはLRパーサを用いている。

宣言ファイルに定義されたノードの

属性は、予め一次元の配列へ変換される。ノード参照更新はすべてこのテーブルを直接参照・変更する事によって行われる。木構造はC言語の構造体を用いた二分木構造へ変換される。

木構造を操作するために必要な領域はAllegro-Cの初期動作時に一括して主記憶上に取られる。速度効率を上げるためにGCは一切行わない。

一つの規則は、一つのC関数へ変換される。

5. Lisp版との比較

Allegro-Cと同等の仕様の言語処理系をSymbolics-3640上へ構築し、処理速度を比較した。ルール数399、サブプログラマ数124、合計523の規則を用いて翻訳を実行した結果、LISP版に比べて最低でも20倍の処理効率を得た。

6. おわりに

開発当初、単純にLISP処理系をCへ書き換えるだけで約5倍の速度効率を得たが、言語仕様からバックトラック等の機能を除き、かつ、属性操作を宣言を用いて高速化することにより、結果として前記のような速度効率を得た。今後は、木の部分構造に対して適用できる規則の選択を統制する手法[2]を用いることにより、より高速に変換動作を実行できるよう改良を加える予定である。

[参考文献]

- [1] 中村順一, "文法記述用ソフトウェアGRADE", 自然言語処理38-4, 1983
- [2] 田中穂積, "自然言語解析の基礎", 産業図書, pp.76-78, 1989