AIプロセサを利用した機械翻訳の高速化

4B-9

伊藤悦雄・野上宏康・今井 徹・平川秀樹
(株)東芝 総合研究所

1. はじめに

機械翻訳システムの高速化は、大量文書の高速翻訳、対話的編集における即時応答性の実現のために不可欠なものである。また、一般に組み合せ的爆発の問題により、文のすべての可能性を解析することは困難であり、処理の高速化は限られた時間内での文解析の成功率を高める。すなわち、翻訳処理の高速化は翻訳品質の向上にも寄与することとなる。

本稿では、当社で開発したAIプロセサ(AIP)[1]を利用した高速機械翻訳システムについて報告する。

2. 高速化の方針

機械翻訳は、構文解析、語彙変換、構造変換、訳文生成など各種の処理により行なわれるが、我々はこのうち所要時間の割合いの最も大きい構文解析処理に着目した。

我々の翻訳システムの構文解析処理はATN(拡張遷移ネットワーク)文法に基づいており、インタプリタ方式で解析が行なわれる。技術マニュアル文などの構文解析を行なう場合には、一般に探索空間が広大となり、構文解析時間が増加する原因となっている。従って、機械翻訳の高速化には、この部分の高速化が大きな課題となる。

構文解析処理の高速化について検討した結果、次の点の改善が有効であることが判明した。

(1) インタプリタ方式である点
(2) バックトラックの回数が非常に多い点
(3)文字列比較の回数が多い点

この様な点を考慮し、我々は解析用文法をWAM命令セット(Warren Abstruct Machine Instruction Set)[2]にコンパイルして実行する方法を考案した。

WAM命令セットはPrologを効率良く実行するための命令セットである。したがって、バックトラック処理、ユニフィケーション処理の高速実行が可能であり、構文解析処理の高速化に適している。また、ATNによる文の解析過程はPrologの実行過程と類似している。そのため、Prologコンパイラ用に考案されたWAM命令セットをATNコンパイラのオブジェクト言語とすることにより、比較的短期間で開発が可能である。

3. AIプロセサの利用

我々の翻訳システムはC言語で記述されている。このためWAMコード(コンパイルした構文解析文法)をC言語のプログラムから呼出し、C言語とWAM命令セットの両者を高速に実行する必要がある。

ところが、WAM命令セットは従来の汎用命令セットと大幅に異なるため、従来のプロセサではC言語のプログラムからWAMコードを呼出し、実行することは不可能であった。

これに対し、AIPは

(1) WAM命令セット、汎用命令セットの双方を同一マシン上で高速実行可能(例えば、WAM命令セットを用いてPrologを従来の5~10倍の速度で、C言語を従来の3~4倍の速度で実行可能)
(2) C言語とWAM命令セットでデータの共有が可能

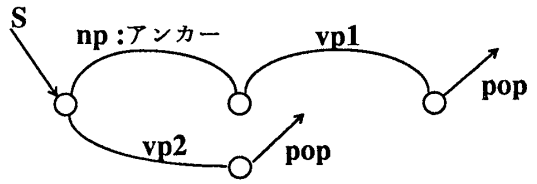
という特徴を有している。このため、上記の問題が解決でき、構文解析処理のWAM命令セット化による高速化が実現可能となった。

4. アンカー命令

ATNとProlog(WAM命令セット)の類似性により、ATNの大部分は、自然な形でWAMコードに変換することが可能であった。しかしながら、効率良い実行の実現のためにはWAM命令セットの仕様では十分でないことも判明した。それは、WAM命令セットには、ATNのアンカー機能[3]に相当する

High-speed Machine Translation System with AI Processor
Etsuo ITOH, Hiroyasu NOGAMI, Toru IMAI, Hideki HIRAKAWA
TOSHIBA Corp.

命令がないことである。アンカーはATNの制御機能の一つであり、他のアークへのバックトラックのみを禁止し、アンカーされたアーク内のバックトラックは禁止しないという機能を提供する。例えば、下図のATNにおいてアーク「**np**」がアンカーされている場合に、「**vp1**」で失敗しバックトラックが生じたとする。このとき、「**np**」へのバックトラックは行なわれるが「**vp2**」へのバックトラックは行なわれない。



アンカーを用いたATN

この動作を従来のPrologで記述すると、次の様になる。

```
s :- np', !, np, vp1.
s :- vp2.
```

ここで、「**np'**」は実行時に変数に値をバインドしない「**np**」である。この「**np'**」が成功した場合はカットオペレータが働き「**vp2**」へのバックトラックを禁止する。次に、変数に値をバインドするために「**np**」を再度実行する必要がある。また、この時点で「**np**」を実行することによって、「**np**」へのバックトラックが可能となり、アンカー機能を実現できる。しかし、この方法ではテストのために「**np'**」を実行するため、効率が悪い。この非効率性を解消するために、我々はアンカーオペレータを考案し、これを「**@**」で表現することとした。これを利用して上記の例を記述すると、

```
s :- np, @, vp1.
s :- vp2.
```

となる。この場合は、「**np'**」を実行する必要がなく、アンカー命令を用いるとATNの機能をWAM命令セットで効率良く実行できる。

5. 実験結果

上記のアンカー命令を含むWAM命令セットへ構文解析文法をコンパイルする文法コンパイラを作成し、これを利用した機械翻訳システムを構築した。このシステムをAIP上で実行し、EWS AS3160上のインタプリタ方式のシステムと比較した。下表にその結果を示す。

この結果、構文解析速度は従来の約10倍となり、予想通りの速度が得られた。また、文によって速度比に5~20倍という差があるが、これは解析時のバックトラック、ユニフィケーションの回数等に起因している。

6. おわりに

今回我々は、構文解析処理に着目して機械翻訳の高速化を行なった。構文解析処理の高速化では、アンカー命令を追加したWAM命令セットにATNをコンパイルし、AIP上で実行した。その結果、従来の約10倍の速度が得られ、この方法の有効性が確認された。

参考文献

[1] 斎藤他, "AIワークステーション(WINE)の開発", 第35回情報処理学会全国大会論文集, pp.1665-1678, 1987.

[2] Warren, D.H.D., "An Abstruct Prolog Instruction Set", Tech. Note 309, Artificial Intelligence Center, SRI International, 1983.

[3] 野上他, "TAURASの構文解析について", 第30回情報処理学会全国大会論文集, pp. 1567-1568, 1985.

構文解析処理速度測定結果

単位: 秒

文(単語数)	AIP利用	インタプリタ方式	速度比
1 (4)	0.05	1.06	21.2
2(18)	0.18	1.62	9.0
3(16)	0.17	1.54	9.1
4(31)	0.31	1.66	5.3
5 (8)	0.12	1.54	12.8
6 (6)	0.08	1.23	15.4
7(11)	0.09	1.32	14.7
8 (9)	0.09	1.17	13.0
9 (5)	0.07	1.12	16.0
10(8)	0.11	1.24	11.3
平均	0.13	1.35	10.4