人 工 知 能 77−16
（1991. 7. 18）

実例に基づく翻訳の超並列化に向けて

佐藤理史
京都大学工学部
606-01 京都市左京区吉田本町
(ssato@kuee.kyoto-u.ac.jp)

要旨

本稿では、超並列実例型翻訳の第一版の核となる部分の概要について述べる。その核となる部分は、超並列実例型解析であり、実例の検索と最適照合を超並列に行なう。提案する demander-supplier モデルでは、解析木(実例)の各節点が自立して動作する agent となる。自分の支配下の部分木が入力の一部と完全照合した節点は、その部分木を供給する supplier となる。一方、自分の支配下が入力と照合しなかった節点は、demander となって、supplier からの木の供給を要求する。入力文の解析は、この demander と supplier が互いに通信し合うことによって、実現される。

Towards Massively Parallel Example-Based Translation

Satoshi Sato
Dept. of Electrical Engineering, Kyoto University
Kyoto, 606-01, JAPAN
(ssato@kuee.kyoto-u.ac.jp)

Abstract

This paper describes the main part of the *Massively Parallel Example-Based Translation*: it is a massively parallel example-based parsing that produces some parse trees for the given input sentence by utilizing examples, i.e. parsed trees. In the proposed model, which called *demander-supplier model*, each node of examples (parsed trees) is an agent. A *supplier* is a node that can supply a tree for a subsequence of the given input sentence, and a *demander* is a node that requires a tree for constructing a larger tree. The parsing is implemented by the comunications between demanders and supplyers.

1 はじめに

長尾による Translation by Analogy の提案 [2] と、それに端を発する研究の展開 [7][4][5][3][6] により、翻訳例を積極的に利用することによって機械翻訳を実現しようという新しいパラダイムが形成されつつある。実例に基づく翻訳 (Example-Based Transation, EBT) がそれである。

研究者によって、多少の立場の相違はあるが、EBT は、おおよそ、以下のような特徴を持ったパラダイムということができる。

- 大量の実例 (翻訳例) を含むデータベースを作成し、
- データベース中の翻訳例を模倣利用することによって、翻訳を実現する。

これによって得られる最大の利点は、

- 規則を必要としないので、システムの構築・改良が容易である。

であり、欠点は、

- 実例データベースの検索と最適照合 (best match) に膨大な計算が必要である。

である。

この欠点を克服 (計算時間を短縮) するために、今後、EBT の並列化を押し進める必要がある。その可能性・見通しは有望であると筆者は考える。その理由は、

- データベースの検索については、並列化が比較的容易であろう。
- 知識源は各々の翻訳例に分散されているので、分散協調型問題解決に向いていると考えられる。

本稿では、現在、筆者が計画している超並列実例型翻訳 (Massively Parallel Example-Based Translation, MP EBT) の第 1 版の核となる部分の概要について述べる。第 1 版では、実例の検索と最適照合を超並列化することを目指す。つまり、その核となる部分は、超並列実例型解析である。

2 非公式な議論

2.1 問題設定

解くべき問題を以下のように設定する。

1. 入力言語側の解析例 (木構造) の集合が与えられている。
2. 未知の単語列に対して、妥当な解析木を求める。その解析木は、照合表現 (既知の解析木をどう切り貼りすると入力を覆えるかを表したもの)[5] として表現する。
3. より妥当だと考えられるものの順に解析木を出力するようにする。つまり、優先順位を自然な形で時間軸に写像する。

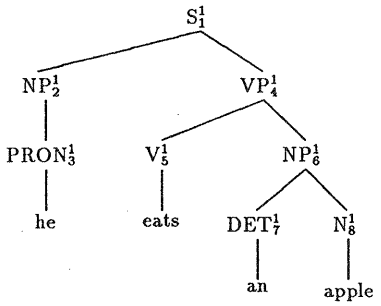
2. の、照合表現、すなわち、実例をどのように組み合わせて用いるかを決定することは、実例型翻訳の処理の 7 割方を占める。故に、表面上は問題を解析に限定しているが、実質上は、翻訳処理の大部分をカバーするものである。3. は、並列化における筆者の野心と考えていただきたい。つまり、優先順位の決定に必要な計算を、並列版では、できるだけさぼろう (暗黙的にそうなってしまうようにしよう) ということである。これを実現することによって、単なる並列化以上の効果をあげようと目論むものである。

2.2 方針

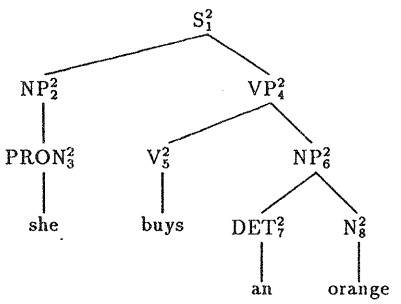
実現のストーリーを考えていこう。なお、ここでは、上記の2. だけに集中し、3. は、一旦棚上げする。

まず、実例(解析例)とは何かということを明らかにしておこう。ここでは、解析例として、構文解析木(句構造)を採用する[1]。以下に例を示そう。

[解析例 1]



[解析例 2]



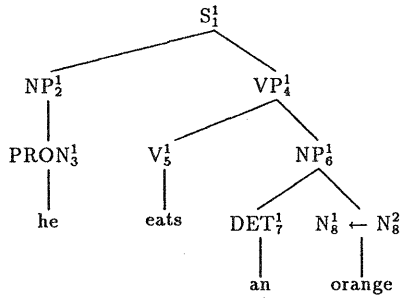
なお、ここで、各ノードに付けられた上添字は、解析例番号を表し、下添字は、解析例内のノード番号を表すものとする。

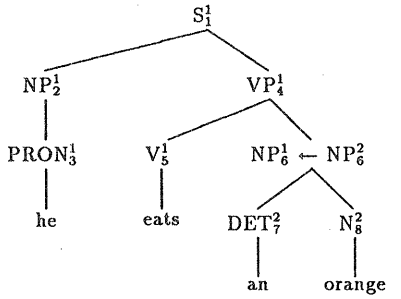
次に、このような解析例に基づいて入力単語列を解析するとは、どのようなことを意味するを明らかにしよう。それは、解析例の構文解析木を切り貼りして、入力単語列に対する構文解析木を作るということである。ここで、どのような切り貼りのし方を許すかということが問題になるが、ここでは、ある節点を根とする(部分)木構造を、その節点の文法カテゴリと同じ文法カテゴリを持つ別の節点を根とする木構造で置き換えることを許すことにしよう。

以上のような仮定に立って、

He eats an orange.

なる入力を解析することを考えよう。答えとしては、以下のような解析木が考えられることになる。[2]





ここで、置換される節点について注目しよう。

$N_8^1 \leftarrow N_8^2$

$NP_6^1 \leftarrow NP_6^2$

ここで、矢印の左側の節点は、その節点下の構造が入力とうまく合わないような節点であり、入力に対する解析木を作るためには、そこに当てはまるような構造を必要としている節点である。このような節点のことを demander と呼ぼう。逆に、矢印の右側の節点は、その節点下の構造が入力とうまく照合した節点であり、demander からの要求に答え、木構造を供給できる節点である。このような節点のことを、supplier と呼ぼう。

このように考えると、解析は、demander からの要求に対して、supplier が答えることによって実現できると考えられる。すなわち、アルゴリズムは、

[1]句構造は、アルゴリズムを考えやすいだろうという理由によって採用した。他の木構造、例えば、単語係り受け構造に対しても、同様のアルゴリズムを考えることができる。これについては、4.2節で述べる。

[2]この他にもいくつか考えられる。

1. demander を求める。

2. supplier を求める。

3. demander と supplier をつなぐ。

の3つの方法を与えればよい。

なお、上記の考え方は、並列オブジェクト指向モデルに向いている。すなわち、demander や supplier は、自立的に動作する並列オブジェクトとして考え、要求やそれに対する応答は、メッセージ通信として考えるということである。以下のアルゴリズムでは、並列オブジェクト指向モデルを前提とする。

3 アルゴリズムの概要

前節の議論によって、アルゴリズムは、1) demander を求める、2) supplier を求める、3) demander と supplier をつなぐ、の3つの方法を与えればよいことがわかった。本節では、2),1),3) の順序でこれらの方法の概要を示す。

3.1 Supplier を求める

Supplier とは、その節点下の構造が入力とうまく照合した節点であり、demander からの要求に答え、木構造を供給できる節点である。このような節点は、活性伝搬の考え方で、容易に求めることができる。

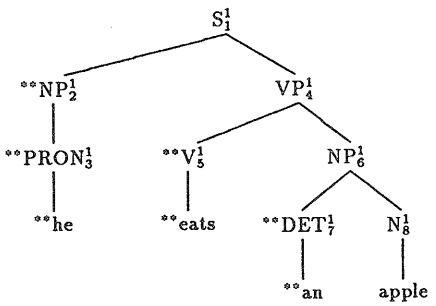
初期活性化規則：入力単語列に含まれる単語に対応する葉節点を強活性化する。

上昇型強活性伝搬規則：ある節点において、その節点の全ての子節点が強活性化しており、それが順序制約を満たすならば、自分自身を強活性化する。

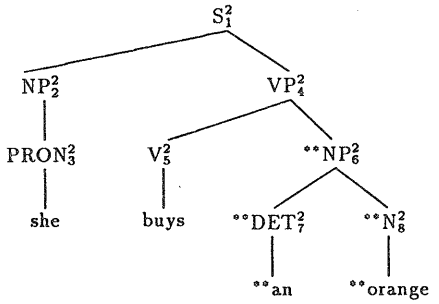
ここで、「強活性化」としたのは、後で、「弱活性化」という用語が出てくるので、それとの区別を明確にするためである。また、「順序制約」とは、その節点下の木構造の葉の並びが、入力単語列の部分列となっていなければならないという制約である。このような2つの活性化規則によって強活性化した節点を supplier とする。(但し、葉節点は、supplier とはしない。)

上記の解析例と入力に対して得られる結果を以下に示す。** が付けられた節点が、強活性化した節点である。

[解析例 1]



[解析例 2]



3.2 Demander を求める

次に、demander を求める方法を考えよう。Demander とは、その節点下の構造が入力とうまく合わないような節点であり、入力に対する解析木を作るためには、そこに当てはまるような構造を必要としている節点である。

上記の例では、N_8^1 や NP_6^1 が demander となってほしい節点である一方、NP_2^2 や $PRON_3^2$ や V_5^2 は、どちらかというと demander にはなってほしくない節点である。このような区別をどのようにして導入すればよいであろうか。そ

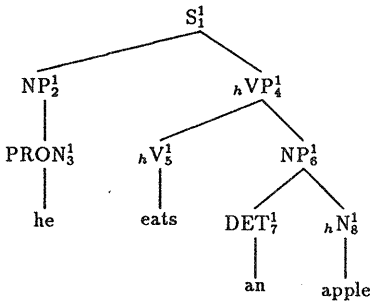
の答えは、なぜ、このような区別があるのかをよく考えれば、おのずと明らかになる。例えば、V_5^2 に注目しよう。なぜ、その節点を置き換えたくないかというと、その節点下の構造、すなわち、動詞 buys は、その解析例のなかで、最も主要な (中心的な) 語であるからである。主要な部分を置き換えるよりは、主要でない部分を置き換える方が、妥当な解析木を導く可能性が大きいと考えられる。

以上の議論に従って、主要部 (head) という概念を持ち込もう。

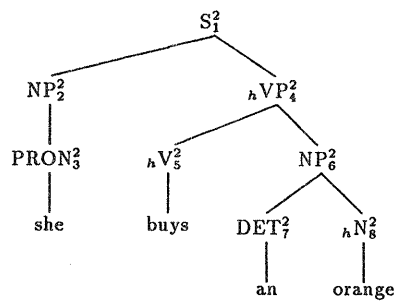
ある節点 (親節点) において、一番中心となる子節点をその親節点の主要部 (head) と呼ぶ。

親節点が子節点を 1 つしか持たない場合は、その子節点が自動的に主要部になる。そうでない場合は、あらかじめ解析例にどの節点が主要部であるかを書いておくことにしよう。以下に、主要部情報を付加した解析例 1,2 を示す。なお、主要部は、前下添字 $_h$ で示す。また、子節点が 1 つの場合は、省略する。

[解析例 1]



[解析例 2]

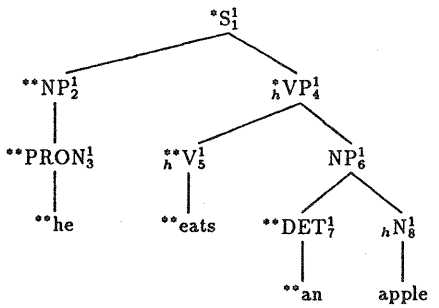


主要部が活性化した節点下の構造は、入力単語列に対する解析木を構成する有望な候補となりうる。そこで、これを示すために、弱活性化という状態を導入しよう。弱活性化は、以下のような規則で伝搬する。

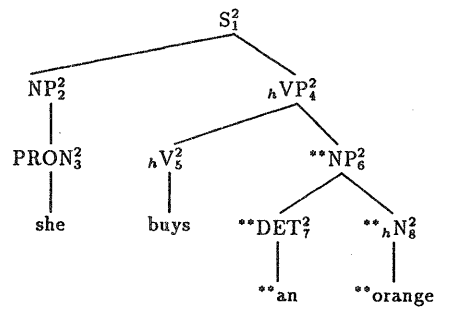
上昇型弱活性伝搬規則：ある節点において、その主要部である子節点が強活性化、又は、弱活性化したならば、自分自身を弱活性化する。

この結果、解析例 1,2 の状態は、以下のようになる。弱活性化した節点は、* で表す。

[解析例 1]



[解析例 2]



以上の準備を経て、demander を以下のように設定する。

下降型要求発動規則：ある節点が強活性化、又は、弱活性化したならば、その節点の子節点のうち、主要部を除く他の全ての子節点を demander とする。

この規則が意味する所は、

- 主要部が活性化している場合のみ、主要部を除く部分を置換の対象とする。
- 主要部は置換の対象としない。

ということである。

この例では、NP_2^1, NP_6^1, DET_7^2 が demander となる。

3.3 Demander と Supplier をつなぐ

上記のアルゴリズムによって、demander や supplier となる節点自身は、自分がそうなっていることを知っている状態になる。残された部分は、demander と supplier が通信しあって、supplier から demander へ木構造を供給する部分である。一旦、木構造が供給されると、あたかも demander が強活性化したように振舞うことによって、後は、上昇型強活性伝搬規則によって、解析が進む。

Demander と supplier をつなぐ方法は、幾つかの候補が考えられる。

3.3.1 ブロードキャスト

最も野蛮な方法は、demander が全ての節点に木構造の要求をブロードキャストするという方法である。要求を受けた節点が supplier である場合は、demander に木構造を供給し、そうでない場合は、demander を記憶しておく (後に、supplier になった場合は、demander に木を供給する。) この方法は一番単純であるが、通信量が膨大になってしまうだろう。

3.3.2 黒板

Demander と supplier をつなぐために、黒板を利用する方法がある。Demander は黒板に、要求を書き込み、supplier は、供給できる木構造を書き込む。この黒板を通して、supplier から demander への木構造の受け渡しを行なう。問題は、この黒板にメッセージが集中するという点である。黒板を非終端記号ごとに用意することによって、ある程度この問題を回避できると考えられるが、どのくらいの逐次性が生じるかは、実験によって確かめる必要がある。

3.3.3 シソーラス

最も有望な方法は、demander(要求メッセージ) と supplier(供給メッセージ) を階層的な単語シソーラス上で走らせる方法である。基本的には、

1. demander は、その主要葉[3]の単語を入口として、痕跡を残しながらシソーラスを上へ登る。
2. supplier は、その主要葉の単語を入口として、痕跡を残しながらシソーラスを上へ登る。
3. ぶつかったところで、supplier から demmander へ木構造を供給する。

この方法で、階層的なシソーラスの節点を並列オブジェクトとすれば、メッセージの集中の問題は、回避できる。また、demander と supplier の主要葉が似ているほど、木構造の受渡しが早く行なわれる。これによって、似ている構造を置換したもの方がより早く出力されるということが、一部実現できそうである。

4 現状と検討

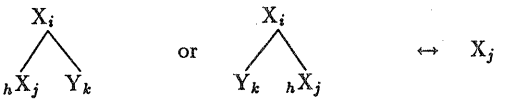
4.1 現状

上記のアルゴリズムを Sequent 上の Allegro CLiP で実現し、動作を確認する作業を行なっている。但し、実際のプログラムでは、

[3]主要部を下へ辿ることによって行き着く葉。

- Structure sharing を採用した。すなわち、実例データベース中には、全く同じ木構造は 1 つしか存在しない。
- 照合表現を出力するために、より複雑なデータの受け渡しを行なっている。
- demander と supplier をつなぐ部分は、黒板を用いる方法を取っている。
- 置換の他に、削除と挿入を制限した形で許している。

最後の削除と挿入の付加について簡単に補足する。ここで、制限した形とは、以下のことを意味する。



右矢印が削除であり、左矢印が挿入である。つまり、右のような構造があった場合 (X_i, X_j は、同じ非終端記号であることを示す)、Y_k を削除・挿入可能な構造と考え、実際には、$X_i \leftrightarrow X_j$ の置換として処理してしまおうというアイディアである。

解析の実行例を図 1 に示す。

4.2 単語係り受け構造の場合

前節のアルゴリズムは、句構造を対象としていたが、MBT2[5] で採用した単語係り受け構造 (木構造) に対しても、同様な方法で実現可能である。つまり、句構造の場合と同じように、demander と supplier を求め、それらをつなぐ方法を与えればよい。この方法は、以下の規則で与えられる。

初期活性化規則：入力単語列に含まれる単語に対応する節点を弱活性化する。

上昇型強活性伝搬規則：自分自身が弱活性化しており、かつ、全ての子節点が強活性化し、それが順序制約[4]を満たすたならば、自分自身を強活性化する。 ($\Rightarrow$ supplier)

上昇型弱活性伝搬規則：なし

下降型要求規則：ある節点が弱活性化したならば、その節点の子節点を demander とする。

4.3 問題点

現在までに、問題設定 (2.1節) の 2. をほぼクリアすることができたと考えている。今後は、問題設定の 3. をいかにしてクリアするかが焦点となってくる。ここで問題になるのが、demander と supplier をつなぐところである。つまり、問題設定の 3. をクリアするためには、次のような要請を満たす必要がある。

> 置換するものと置換されるものとの間の意味的整合性の度合が、その置換に要する時間 (すなわち、demander と supplier が出会うためにかかる時間) に反映されること。

階層的シソーラスを利用する方法は、これに対する 1 つの解答ではあるが、十分とは言えないだろう。

筆者の現在の考えは、実例が文法的構造だけしか持たないのでは、これを実現するのは、かなり困難であるというものである。つまり、実例を柔軟に利用できるためには、その実例を十分に「理解」している必要がある。意味的整合性まで考慮するためには、その実例を意味的に理解していなければならないだろう、という考えである。その帰結として、以下のような案を考えている。

[4] この場合の順序制約は、非交差条件から導くことができる。

```
(parse '(she eats the apple on the table) 's)
Parsing Start ...
Bottom Up Activation Start ...
#<WORD:SHE> is strongly activated for [0-1].
#<SNODE:16:PRON> is strongly activated for [0-1].
#<SNODE:15:NP> is weakly activated.
#<SNODE:15:NP> is strongly activated for [0-1].
#<WORD:EATS> is strongly activated for [1-2].
#<SNODE:5:V> is strongly activated for [1-2].
#<SNODE:4:VP> is weakly activated.
#<SNODE:1:S> is weakly activated.
#<WORD:THE> is strongly activated for [2-3].
#<SNODE:21:DET> is strongly activated for [2-3].
#<WORD:APPLE> is strongly activated for [3-4].
#<SNODE:8:N> is strongly activated for [3-4].
#<SNODE:6:NP> is weakly activated.
#<SNODE:6:NP> is strongly activated for [2-4].
#<SNODE:4:VP> is strongly activated for [1-4].
#<SNODE:1:S> is strongly activated for [0-4].
#<WORD:ON> is strongly activated for [4-5].
#<SNODE:24:P> is strongly activated for [4-5].
#<SNODE:23:PP> is weakly activated.
#<WORD:THE> is strongly activated for [5-6].
#<SNODE:21:DET> is strongly activated for [5-6].
#<WORD:TABLE> is strongly activated for [6-7].
#<SNODE:26:N> is strongly activated for [6-7].
#<SNODE:25:NP> is weakly activated.
#<SNODE:25:NP> is strongly activated for [5-7].
#<SNODE:23:PP> is strongly activated for [4-7].
#<SNODE:19:NP> is strongly activated for [2-7].
#<SNODE:4:VP> is strongly activated for [1-7].
#<SNODE:1:S> is strongly activated for [0-7].
```

```
Top Down Process Start ...
OUTPUT = ((S)
          ((NP)
           ((PRON) (SHE)))
          ((VP)
           ((V) (EATS))
           ((NP)
            ((NP)
             ((DET) (THE))
             ((N) (APPLE)))
            ((PP)
             ((P) (ON))
             ((NP)
              ((DET) (THE))
              ((N) (TABLE)))))))
MEXPR  = (#<SNODE:1:S>
          (REPLACE #<SNODE:2:NP>
            (#<SNODE:15:NP>))
          (REPLACE #<SNODE:6:NP>
            (#<SNODE:19:NP>
             (REPLACE #<SNODE:20:NP>
               (#<SNODE:6:NP>
                (REPLACE #<SNODE:7:DET>
                  (#<SNODE:21:DET>)))))))
```

解析例として、以下の3文の解析例をデータベースに持っている。

1. He eats an apple.
2. He buys an orange.
3. She ate the banana on the table.

文1(1:S) の he(2:NP) を she(15:NP) で置き換え、an apple(6:NP) を the banana on the table(19:NP) で置き換え、the banana(20:NP) を an apple(6:NP) で置き換え、an(7:DET) を the(21:DET) で置き換えることによって、入力文に対する解析木が求まっている。

図 1: 解析実行例

1. 各実例に対して、文法的な構造とは別に、意味的構造を表す表現を導入する。

2. 単なる単語のシソーラスではなく、意味的構造に対するシソーラスを導入する。

3. 部分的に照合した文法的構造から、それに対応する意味的構造を活性化する。

4. 活性化した意味的構造 (supplier) と欠けている意味的構造 (demander) をシソーラス上で走らせ、意味的整合性を計算する。

つまり、文法的な知識と意味的な知識を活性伝搬のような手法で結び付け、相互に影響しあって、協調的に動作させるというものである[5]。このようなアーキテクチャを確立できれば、文脈的な知識を採り入れることも可能であろう。

5 おわりに

本稿では、超並列実例型翻訳を実現するために必要な、超並列実例型解析のアルゴリズムの概要について述べた。今後の計画としては、

- 問題点で述べた、実例型翻訳における、文法、意味、文脈等の各種の情報を利用した超並列協調問題解決手法について、検討を進める。
- プロトタイプを並列マシン上に実現する。

を進めていく予定である。

参考文献

[1] Kitano, H., A Massively Parallel Model of Simultaneous Interpretation: The ΦDMDIALOG System, CMU-CMT-89-116, Carnegie Mellon University, 1989.

[2] Nagao, M., A Framework of a Mechanical Translation between Japanese and English by Analogy Principle, in *Artificial and Human Intelligence* (Elithorn & Banerji, Eds.), Elsevier Science Publishers, pp173-180, 1984.

[3] Sadler, V., *Working with Analogical Semantics*, Foris Publications, 1989.

[4] 佐藤理史, 長尾真, 実例に基づいた翻訳, 情報処理学会研究報告, NL-70-9, 1989.

[5] 佐藤理史, 実例に基づく翻訳 II, 情報処理学会研究報告, AI-70-3, 1990.

[6] 佐藤理史, 実例に基づく訳語選択, 人工知能学会誌, Vol. 6, No.4, 1991(掲載予定).

[7] Sumita, E. and Tsutsumi, Y., A Translation Aid System Using Flexible Text Retreval Based on Syntax-Matching, TRL Research Report, TR87-1019, IBM, 1988.

[8] Sumita, E., Iida, H., and Kohyama, H., Example-based Approach in Machine Translation, IPSJ, Proc. of InfoJapan'90, Part:2, pp65-72, 1990.

[5] Supplier と demander を動的に結び付けることを除けば、[1] がこのようなモデルを提案している。