自然言語処理 73－6
(1989. 6. 29)

未登録語を含む文の一解析法
A Parsing Method for Sentences with undefined words

塚田孝則† 西野敏行‡ 小柳和子†
Takanori Tsukada†, Toshiyuki Nishino‡, Kazuko Oyanagi†

†日立ソフトウェアエンジニアリング株式会社
†Hitachi Software Engineering Co., Ltd.
6-81, Onoe-Machi, Naka-Ku, Yokohama 231 Japan

‡日立エスケイソフトウェア株式会社
‡Hitachi SK Software Co., Ltd.
5-79, Onoe-Machi, Naka-Ku, Yokohama 231 Japan

あらまし データベース用の自然言語インタフェースなどのアプリケーションでは辞書に登録されていない単語(未登録語)を検索キーワードとして含む文を扱う場合がある。しかし日本語の文は分かち書きされないので英語などに比べ未登録語の位置を推定するのが難しい。

本論文で述べる方式は構文解析アルゴリズムSAXで未登録語を扱えるようにするために次のような拡張を行ったものである。すなわち、まず形態素解析と構文解析を融合し、文を1文字づつ解析するようにした。次に文の解析が途中で失敗した場合にはバックトラックをして最後の文字を特別な非終端記号"補助記号"として解析し直すようにした。そして、文中のバックトラックした部分を未登録語と認識するようにした。

また本方式は形態素解析、構文解析、意味解析を融合したシステムになっているため、未登録語の抽出に文法や意味知識を用いることができる。

Abstract In the applications like natural language interfaces with databases, it is necessary to parse sentences with words not listed in the dictionary (*undefined words*). Since Japanese sentences are not written word-by-word, it is more difficult to locate undefined words in Japanese than in English.

The method described in this paper modifies the parsing algorithm SAX in the following ways: Morphological and syntactic analysis phases are integrated together to parse character-by-character, not word-by-word. Second, when the parser fails, it backtracks and parses the last terminal as a special nonterminal "HELP". Finally, the parser recognizes the backtracked part of the sentence as an undefined word.

As this method integrates all of morphological, syntactic, and semantic analysis phases, it can utilizes the grammar and the semantic knowledge to recognize undefined words.

1 はじめに

1.1 未登録語処理の必要性

自然言語処理での文の解析では辞書に登録されていない単語(未登録語)を扱う必要がある。例えばデータベースの日本語文インタフェースのようなアプリケーションでは、検索のキーワードが未登録語として文中に出て来る。そのようなアプリケーションの多くは未登録語を「」で囲むとか、データベース中のデータから辞書を作るなどして文を解析している。

1.2 未登録語処理の難しさ

分かち書きしていない日本語文の解析では単語の区切りがあいまいなため、未登録語の正しい抽出が英語などに比べて難しい。

1.3 これまでの未登録語処理

これまでにも未登録語を扱う研究が数多くなされてきたが [3][4][7][8][9][12][13] 、次のような問題点があった。

(1) 完全横型探索の構文解析方法では未登録語を扱うと解析量が指数関数的に増大する。

(2) これまでの未登録語処理の多くは文法に依存した特殊なもので、構文解析と文法が独立していない。

(3) 形態素解析や構文解析のみで意味処理まで含めたものは少ない。一般に意味処理に書かれる制約条件は統語的制約と意味的制約条件がある。未登録語の正しい抽出にはこれらの制約条件を用いる必要がある。

1.4 SAX の特徴

SAX(Sequential Analyzer for syntaX and semantiCS)[6][10] は Prolog 用の横型探索型の構文解析システムである。本方式は、SAX を未登録語解析のための拡張をしたものである。SAX には次のような特徴がある。

(1) 文の先頭(または末尾)から順に解析していく。

(2) 横型であるので全ての解析結果を並列的に計算していく。

(3) 解析の途中での副作用はない。

(4) 下降型予測を行いながら解析を進める。

(5) 解析の途中で意味処理を行うことができる。

(6) DCG(Definite Clause Grammar) で記述した文法は Prolog のプログラムへ変換して用いるので、文法と構文解析のアルゴリズムは完全ではないが互いに独立している。

本方式では、後に述べるように (1)(2)(3) の特徴を利用している。

1.5 本論文の構成

2章ではまず本方式の考え方、概要を示した後、詳細なアルゴリズムについて述べる。2.4節では本方式を未登録語を含む文に適用した結果について述べる。3章では本方式の特徴と問題点について考察する。

2 本方式

2.1 考え方

次のような考え方でこの方式を考案した。

(1) 形態素解析の仕事量を最小限に抑え、構文解析以降に自由度を残しておくようにする。つまり、形態素解析の段階で誤った単語分割を行わず、構文解析に単語分割も行わせることで、形態素解析だけでは難しかった未登録語の抽出を構文解析と意味処理を併用して行うようにする。

(2) SAX では中間解析結果の数が 0 個になった時点で解析が失敗したことがわかるので、その付近に未登録語があることが推定できる。すなわち、この時点で未登録語の抽出を始めれば、横型解析であっても解析数が指数関数的に増加することを防ぐことができる。

2.2 入力文の種類

本方式は、入力文の文字の種類としてローマ字文、ひらがな文、漢字混じり文が扱える。また、分かち書きした文、べた書きした文の両方を対象にしている。

2.3 構文解析アルゴリズムのの拡張

2.3.1 概要

本方式は、構文解析アルゴリズム SAX で未登録語を扱えるようにするために次のような拡張を行ったものである。

(1) 形態素解析と構文解析を融合し、文を 1 文字ずつ解析するようにした。

(2) 文の解析の途中で中間結果が 0 個になった場合は解析が失敗した場合で、バックトラックをして直前の文字を特別な非終端記号“補助記号”として解析し直すようにした。その後は再び通常の解析に戻る。そして、文中のバックトラックした部分を未登録語と認識するようにした。

つまり、本方式では文を先頭から読んでいき、文法に当てはまらない文字の並びを未登録語として解析する。この時一つ一つの未登録語の範囲は一番短くなるように選ぶ。以下これを更に詳しく説明する。

2.3.2 文字単位の形態素解析

図 1に本方式での文字分割と構文解析の役割を示す。

(1) 形態素解析では、構文や意味の情報を使わずに文字種による文字分割だけを行う。漢字、ひらがなどは 1 文字づつに分割するが、数字列、アルファベット列、例えば“100”、“Pereira”などは全部で 1 文字とみなし、便宜上、属性付きの非終端記号とする。以降は、形態素解析とは呼ばずに“文字分割”と呼ぶ。図 2に文字分割の例を示す。

「それは 1980 年に Pereira が書いた。」

⇓

文字分割

⇓

番号	記号の種別	名前	属性
1	終端記号	”そ”	—
2	終端記号	”れ”	—
3	終端記号	”は”	—
4	非終端記号	数	”1987”
5	終端記号	”年”	—
6	終端記号	”に”	—
7	終端記号	”Pereira”	—
8	終端記号	”が”	—
9	終端記号	”書”	—
10	終端記号	”い”	—
11	終端記号	”た”	—
12	非終端記号	区切り	”。”

図 2: 文字分割

(2) (1) に合わせて構文解析部 (SAX) の入力は単語列ではなく文字の列とする。

(3) 文法 (DCG) の終端記号は単語ではなく文字とする。例えば「書く」という動詞は次のように文法で記述する。

動詞 (書く) ⟶ "書", "く".

2.3.3 バックトラックの追加

文の解析の途中で中間解析結果が 0 個になった場合は、そこから未登録語の抽出を始める。まずバックトラックをして直前の

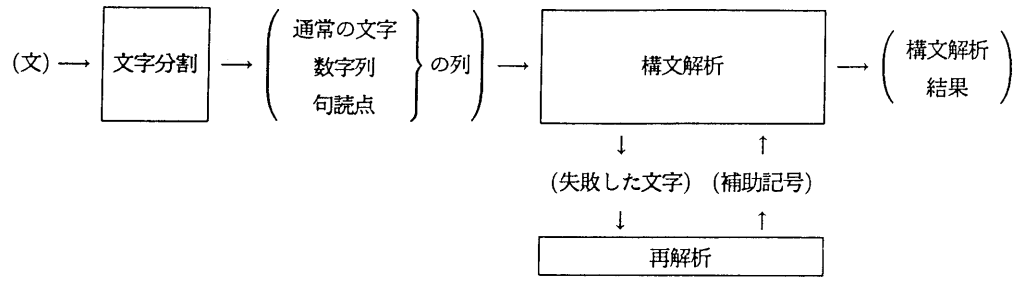

図 1: 文字分割と構文解析の役割

文字を特別な非終端記号補助記号として解析し直す。全ての文法ルールを用いて解析しても解析に失敗する場合に限り補助記号を用いたバックトラックによる解析を行う。その場合は以下のルールを例外的に適用する。補助記号の名前は構文解析プログラムに指定することができる。

補助記号 ⟶ 任意の文字.

図3に拡張したSAXの構文解析アルゴリズムを示す。

2.3.4 文法への補助記号の追加

文法に特別な非終端記号補助記号を追加する。この補助記号が並んだものを未登録語とみなす。

未登録語 ⟶ 未登録語, 補助記号.
未登録語 ⟶ 補助記号.

また、現在はデータベースインタフェースを扱っているために、未登録語はデータベースのキーワードの場合が多い。そこで未登録語は名詞として扱っていて次のような文法を用いている。

名詞 ⟶ 未登録語.

2.4 例

図4の文法を用いた「それはひたちだ」という文の解析例を図5に示す。図では各々の文字を解析したときにどの文法ルールが適用されたのかを示す。

文	⟶	補語, 動詞.
補語	⟶	名詞, 助詞.
名詞	⟶	"そ","れ".
助詞	⟶	"は".
動詞	⟶	名詞,"だ".
名詞	⟶	未登録語.
未登録語	⟶	未登録語, 補助記号.

図 4: 文法例

実際には未登録語であっても、はじめの解析では通常の単語として解析してしまう場合もある。しかしその場合でも、以降の文の解析がどこかで失敗してバックトラックをして戻って来るので、結果的には未登録語部分が補助記号として解析される。

3 考察

3.1 構文解析アルゴリズムとの関係

本方式は SAX を拡張したが、1.4節の(1)(2)(3) の特徴を満たしている構文解析ア

変更した部分を†で示す。

(1) 構文解析の入力は次の二つである。

(a) 文: $a_1 a_2 \ldots a_n$(n は文の長さ)

(b) スタートシンボル: S

(c) †補助記号: HELP

(2) 内部変数は次がある。

(a) 中間解析結果: $\mathrm{IDS}_i (i = 0, \ldots n)$。ただし IDS_0 は begin というid だけである。

(3) 結果として次を返す。

(a) 解析結果

(4) i $(i = 1, \ldots, n)$ ステップ目:

(a) もし IDS_i が空でないなら:

(i) 文字 a_i が辞書にあるなら、そのまま IDS_{i-1} を用いて解析する。

(ii) †文字 a_i が辞書にないなら、HELP として IDS_{i-1} を用いて解析する。

その結果を IDS_i として $i+1$ ステップ目以降を行う。

(i) もし $i+1$ ステップ目以降が空でないものを返したならば i ステップ目もそれを返す。

(ii) †もし $i+1$ ステップ目以降が空を返した(失敗した)ならば a_i を非終端記号 HELP として a_i を解析し直す。そして $i+1$ ステップ目以降の結果を返す。

(b) もし IDS_i が空なら、空を返す。

(5) $n+1$ ステップ目: 文の最後まで解析したのだから、IDS_n のうちで、スタートシンボル S に対応したものだけを最終解析結果として返す。

図 3: 拡張した構文解析アルゴリズム (SAX)

ステップ	文字	適用したルール(かっこの中は対応する文字列)
⋮	⋮	⋮
(4)		補語(それは) ⟶ 名詞(それ), 助詞(は).
(5)	ひ	失敗(バックトラック)
(6)	ひ	補助記号(ひ) ⟶ "ひ".
(7)		未登録語(ひ) ⟶ 補助記号(ひ).
(8)	た	失敗(バックトラック)
(9)	た	補助記号(た) ⟶ "た".
(10)		未登録語(ひた) ⟶ 未登録語(ひ), 補助記号(た).
(11)	ち	失敗(バックトラック)
(12)	ち	補助記号(ち) ⟶ "ち".
(13)		未登録語(ひたち) ⟶ 未登録語(ひた), 補助記号(ち).
⋮	⋮	⋮

図 5:「それはひたちだ」の解析例

ルゴリズムなら他のもの、例えばEarleyのアルゴリズム[1]、Tomitaのアルゴリズム[2]などにも応用することができる。

3.2 文法と構文解析アルゴリズムの独立性

構文解析のアルゴリズムに含まれている文法知識は任意の文字が補助記号になるというルールだけであり、文法と構文解析のアルゴリズムの独立性が高い。

3.3 未登録語のない場合の解析時間

未登録語が文に含まれない場合は、バックトラックは起きないので従来のSAX同様の良い効率が得られる。ただし、バックトラックができるように構文解析の中間結果を保存しておかなくてはならないのでメモリ効率は悪くなっている。また文法が文字単位になっているための構文解析処理時間の増加があることが考えられる。

4 実験結果

以上説明してきたアルゴリズムを、70文法ルール、165単語で実験した結果について考察してみる。なお、動詞の各変化型は1単語と数え、同一単語のローマ字読み、ひらがな、漢字読みは全部合わせて1単語と数えている。

また句読点に関する制約として次を文法、意味処理に記述してある。

(1) 文字分割では空白、“、”、“。”などの句読点も、普通の文字と同様に1単語と考えている。

(2) 句読点は特定の語句、例えば補語、の次にしか来られないという制約を文法に次のように記述している。

文 ⟶ 補語, 句読点, 動詞.
文 ⟶ 補語, 動詞.

(3) 句読点は補助記号にはならないという制約を意味処理に記述している。

表 1: 文の解析の CPU 時間

番号	種類	入力文	CPU 時間
1	べた書き	もりやがかいたほんはありますか	17.06 秒
2	分かち書き	もりやが␣かいた␣ほんは␣ありますか	1.26 秒
3	漢字混じり	盛屋が書いた本はありますか	0.98 秒
4	べた書き	1987 ねんにしゅっぱんされたろんぶんからもりやがかいたものをえらべ	13.88 秒
5	分かち書き	1987 ねんに␣しゅっぱんされた␣ろんぶんから␣もりやが␣かいた␣ものを␣えらべ	2.12 秒
6	漢字混じり	1987 年に出版された論文から盛屋が書いたものを選べ	1.62 秒

下線を引いた部分が未登録語である。
␣は空白を示し、これが句読点になる。
ハードウェア: Hewlett Packard 9000 シリーズ モデル 350
プログラミング言語: CommonLisp

以上の制約を付けた上で、べた書きの文と句読点をいれ分かち書きした文、漢字混じり文で文の解析を行ってみた。表1に入力文と構文解析にかかった CPU 時間を示す。

句読点がある文では、無い場合に比べて約 5 分の 1 から 10 分の 1 の時間で解析ができることがわかる。これは句読点を付けると補語の区切りが明確になり、誤った解析が早いタイミングでフィルタされ、これにより句の区切りがより明らかになるので文の解析時間が短くなるためである。また漢字混じり文では、未登録語と助詞の区別がつき易いため処理時間はさらに短くなる。これは人間の場合と同様である。

4.1 問題点

4.1.1 バックトラックによる再計算

バックトラックする区間が長くなると解析時間が指数関数的に増加してしまう。これは同じ部分の解析を何度も行ってしまうからである。この対策として、BUP(Bottom-Up Parser in Prolog) で行われたような、一度成功した解析を記憶しておくような仕組み [11] を入れることを考えている。

4.1.2 不適切な解析

これまでにも報告があったが、ローマ字入力や平仮名入力の場合、「さとう」のような未登録語が「“さ”という未登録語と“う”という未登録語」のように解析される。この対策としては、例えば「ひらがなは未登録語としない」というようなチェックを行う補強項を文法に加えることを考えている。

4.1.3 誤った文の入力に対する問題

誤った文を入力すると、バックトラックの回数が非常に多くなり、解析に非常に長い時間がかかる。これは、文を入力する際に句読点を入れるなどして未登録語の範囲をわかりやすくすることである程度解決できるが、例えば未登録語の長さに 5 文字までといった制限を設けたり、バックトラックの回数に制限を設けたりすることを考えている。

5 まとめ

5.1 結論

構文解析システム SAX に、補助記号とバックトラックを導入して未登録語処理を行えるようにすることができた。特徴としては構文解析アルゴリズムの変更が少なく、文法との独立性も高いことが挙げられる。しかし、実用化にはバックトラックの制御と未登録語の判定基準などで改善の余地がある。

5.2 課題

5.2.1 単語数の増大の影響の考察

現在は単語数約 165 程度で実験を行っている。単語数が多くなった場合に解析時間がどう変化するのかについての実験を今後行う必要がある。

5.2.2 文法情報の推定

未登録語の解析では位置だけではなくその文法情報などの推定も重要である [5]。文法情報としては例えば、抽出した未登録語が人名を示しているのか、団体名を示しているのかの区別をすることなどが挙げられる。本論文では触れなかったが未登録語のより正確な位置を推定するためにも未登録語の文法情報の推定は重要である。

参考文献

[1] A. V. Aho and J. D. Ullman: *The Theory of Parsing, Translation and Compiling Volume 1*, Prentice-Hall, 1972

[2] Masaru Tomita: *An Efficient Context-free Parsing Algorithm for Natural Languages*, in Proceedings of IJCAI-85, 1985

[3] 芦沢, 平井, 梶: 日英機械翻訳前編集支援システム, 情報処理学会第 36 回 (昭和 63 年前期) 全国大会 2U-3, 1988

[4] 大場, 元吉, 井佐原, 横山, 石崎, 板橋: 未定義語を含む文の多段階構文解析法, 情報処理学会自然言語処理研究会 70-4, 1989

[5] 亀田, 森田, 倉島, 藤崎: 未知語の分類とその処理規則, 情報処理学会第 36 回 (昭和 63 年前期) 全国大会 5T-5, 1988

[6] 杉村, 松本: 構文解析システム SAX の CIL による実現, 情報処理学会第 33 回 (昭和 61 年後期) 全国大会 3K-1, 1986

[7] 高橋, 奥村, 伊東, 小川: 構文解析における未定義語の抽出, 情報処理学会第 33 回 (昭和 61 年後期) 全国大会 3K-7, 1986

[8] 長瀬: ATLAS II における未登録語の抽出とその扱い, 情報処理学会第 36 回 (昭和 63 年前期) 全国大会 4U-7, 1988

[9] 藤崎, 亀田, 森田, 倉島: 高検索機能データベース作成のための形態素解析と統語解析, 情報処理学会第 36 回 (昭和 63 年前期) 全国大会 6u-7, 1988

[10] 松本, 杉村: 論理型言語に基づく構文解析システム SAX, コンピュータソフトウェア, 日本ソフトウェア科学会, Vol. 3, No. 4, pp. 4–11, 1986

[11] 松本, 杉村: 自然言語の基礎理論, 第 2 章 論理型言語による自然言語へのアプローチ, 共立出版, pp. 51–86, 1986

[12] 元吉, 井佐原, 石崎: 日本語用完全横型探索構文解析法, 情報処理学会第 32 回 (昭和 61 年前期) 全国大会 4S-3, 1988

[13] 吉村, 武内, 津田, 首藤: 未登録語を含む日本語文の形態素解析, 情報処理学会論文誌, Vol. 30, No. 3, 1989