TR-IT-0251

文脈自由文法の有限状態オートマトン近似アルゴリズムの実装と評価

Implementation and evaluation of algorithm approximating context-free grammar to finite-state automaton

甘粕 哲郎 Tetsuo Amakasu
山本 博史 Hirofumi Yamamoto
塚田 元 Hajime Tsukada

1997 年 12 月 26 日

文脈自由文法 (CFG) を有限状態オートマトン (FSA) に近似するアルゴリズムを実装するとともに，アルゴリズムの評価実験を行った．音声認識で CFG に基づく制約を用いる場合，発話が長くなるに従い，探索空間が爆発するという問題がある．この問題を解決するため，CFG に基づく制約を，FSA で近似するアルゴリズムがいくつか提案されている．その中から Pereira らのアルゴリズムを実装し，ホテル予約の模擬会話に現れる発話と，その発話を音声認識するために開発した文法を用いて，アルゴリズムの評価を行った．その結果，妥当な近似精度で，膨大な探索空間を削減できることが示せた．

目次

1 はじめに

現在，多くの音声理解システムでは，音声認識部と言語理解部で独立した言語モデルが用いられている．音声認識部では，n-gramに基づく統計的言語モデルが，言語理解部では文脈自由文法(CFG)に基づく文法的言語モデルが広く用いられる．音声理解レベルの性能を最大限高めるためには，音声認識部においても，言語理解部の文法的言語モデルを利用することが望ましいと考えられる．

しかし，CFGに基づく制約を音声認識に用いる場合，認識候補の探索空間は発話長の2乗オーダで増大するため，発話長が長くなると，探索空間が爆発するという問題もある．この問題を解決するため，CFGに基づく制約を，FSAで近似するアルゴリズムがいくつか提案されている[1][3]．本研究では，この中から，Pereiraらのアルゴリズム[1]を実装し，我々の想定する音声認識タスクにおける近似精度と探索空間の削減の両面から，アルゴリズムの評価を行った．

CFGの近似アルゴリズムとしては，他にも文法規則を有限回展開してFSAを生成する手法[3]も提案されている．しかし，この手法では，FSAが受理する言語と元のCFGの生成する言語を比べたとき，包含関係が保証されないという欠点がある．それに対して，Pereiraらのアルゴリズムでは，近似されたFSAの受理する言語は，元のCFGが生成する言語を包含することが保証されるばかりか，多くの文法のクラスでは，一致するという性質が証明されている．この性質は，音声認識のための言語制約の近似手法として，ふさわしいものだと考えられる．

まず第2節で，Pereiraらのアルゴリズムを説明する．それから第3節では，アルゴリズムの評価実験について述べる．この実験の結果，我々の想定する音声認識タスクにおいては，本近似アルゴリズムを用いることによって，妥当な近似精度で，膨大な探索空間を削減できることが示せた．

2 アルゴリズム

2.1 シフト還元認識器

CFGをFSAに近似するPereiraらのアルゴリズム[1]は，スタックを持つプッシュダウンシフト還元認識器(pushdown shift-reduce recognizer)(図1)を基に，近似的なFSAを生成するというものである．シフト還元認識器はCFGから生成される言語と同じ言語を受理する機械である．これは，有限状態集を持った制御部(control)と入力系列を読み込むヘッド，そして，スタックを持っている．動作としてはヘッドから系列を読み込むシフト動作と，ある状態に到達した場合に，スタックを積み替える還元動作からなる．シフト還元認識器は，SLR法によってLR構文解析表を作る時に用いられるLR(0) characteristic machine[2]を基に構成することができる．

LR(0) characteristic machineはシフト還元認識器の有限状態集である．それぞれの状態はドットルールと呼ばれるものの集合である．ドットルールとはある文法Gの各文法生成規則 $A \rightarrow abc$ の右辺中のどこかに $[A \rightarrow a \cdot bc]$ のようにドットを挿入したものであり，LR(0) characteristic machine M(G) のそれぞれの状態中のドットルールは，直観的には，構文解析のある時点において，解析が生成規則のどこまで進んでいるか表すものとなる．

プッシュダウンシフト還元認識器R(G)の状態を $\langle s, \sigma, \omega \rangle$ として表す．s はM(G)の状態，σ はスタックの系列，ω は入力系列である．スタックは，$\langle s, X \rangle$ という状態と文法記号 X の対の系列である．またM(G)の状態遷移関数を $\delta(s, X)$ とする．M(G)の初期状態を s_0 とする

と R(G) の初期状態は $\langle s_0, \epsilon, \omega \rangle$ であり，シフト動作，還元動作は

$$\text{シフト動作}: \langle s, \sigma, x\omega \rangle \vdash \langle \delta(s,x), \sigma\langle s,x \rangle, \omega \rangle$$

$$\text{還元動作}: \langle s, \sigma\tau, \omega \rangle \vdash \langle \delta(s_1, A), \sigma\langle s_1, A \rangle, \omega \rangle$$

のように表されて，ここで、還元動作の場合にはコンプリートドットルール $[A \to X_1, \cdots, X_n\cdot]$ が s 中にあり，スタック系列の上 n 系列が $\tau = \langle s_1, X_1 \rangle \cdots \langle s_n, X_n \rangle$ である必要がある．R(G) の終了状態は $\langle s_F, \langle s_0, S \rangle, \epsilon \rangle$ でこの状態に到達することができれば入力系列は受理されたことになる．ただし，s_F は M(G) の終了状態の一つ,S は文法 G の文法開始記号を表す．

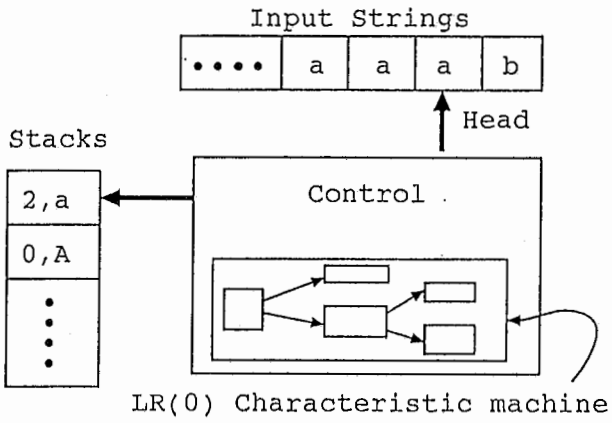

図 1: Pushdown shift-reduce recognizer

2.2　Characteristic machine M(G) の生成

次に，文法規則からドットルールの集合をどのように生成して characteristic machine を生成するかを述べる．最初，必要な手続きとして閉包演算 closure(I) と遷移先演算 goto(I,X) を定義する．ここで I は文法 G の規則から生成したドットルールの集合，X は文法記号を表す．

closure は characteristic machine の状態としてのドットルールの集合をもとめる手続きである．

```
function closure(I);
begin
      J := I;
      repeat
            for  J の各ドットルール [A → α·Bβ] と G の各生成規則 B → γ について
            ただし，[B → ·γ] は J に含まれていない do
                  [B → ·γ] を J に加える．;
      until J に追加できるドットルールがもうない;
      return J;
end;
```

一方 goto 演算はある集合から遷移先となる集合を求める手続きである．

```
function goto(I,X);
begin
```

I の中のドットルールで $[A \to \alpha \cdot X\beta]$ となるようなすべてのドットルールについてのドットルール $[A \to \alpha X \cdot \beta]$ を集めた集合 J を作る.
return closure(J);
end;

次に，この closure, goto 演算を用いて characteristic machine を生成する方法について述べる．文法 G の開始記号 S に対してさらに新しい開始記号 S' を考え，文法規則 $S' \to S$ を G に追加する．この文法の拡張によって G から生成される言語は変化しない．この追加した規則を使って，characteristic machine M(G) を作成するアルゴリズム items(G) は以下のように定義されている．

procedure items(G);
begin
C := { closure ({$[S' \to \cdot S]$}) };
repeat
for C の各集合 I と各文法記号 X, ただし goto(I,X) が空でない **do**
if goto(I,X)が C に含まれていない **then**
goto(I,X) を C に加える ;
ラベルに X をもつエッジを C 中の goto(I,X) に向けて引く;
until C に追加できるドットルール集がもうない;
end;

例えば，以下のような文法 G_1 に対して characteristic machine は図 2のようになる.

$$G_1 \quad \begin{array}{l} S \to Ab \\ A \to Aa \\ A \to \epsilon \end{array}$$

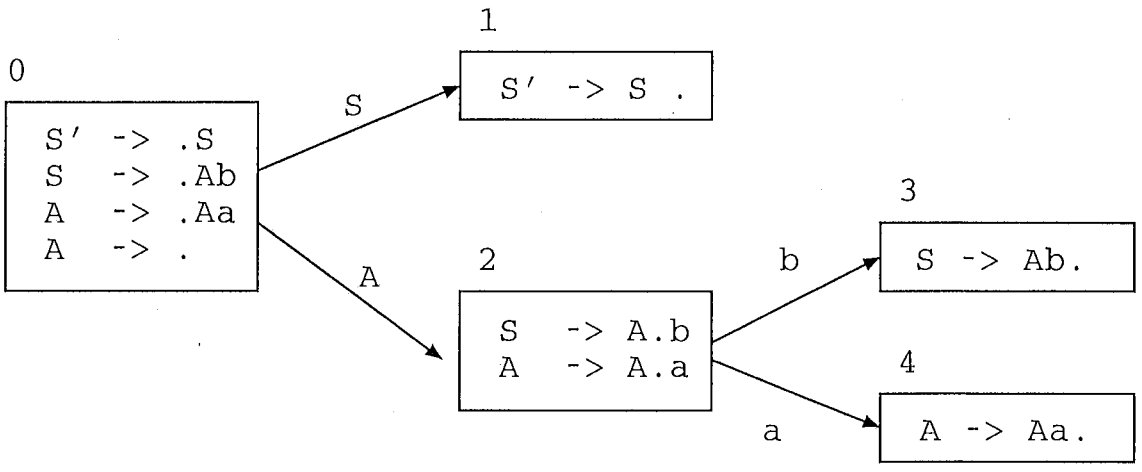

図 2: G_1 の LR0 characteristic machine

2.3 FSA への近似

FSA へ近似するには，コンプリートドットルール (complete dot rule) $[A \rightarrow \alpha\cdot](|\alpha| \geq 0)$ を含む characteristic machine M(G) 中の集合から，シフト還元認識器動作時に還元動作を行なった時に遷移先となりうる集合すべてに対して $\epsilon-$ 遷移 を作り出だすという作業を行なう．この作業によって，M(G) そのものが非決定性有限状態オートマトンに変換される．

本来，還元後の状態 $s'(s' = \delta(s_1, A))$ は s と還元前の R(G) のスタック系列によって決まるものである．本近似アルゴリズムではスタック系列を無視し s' としてとりうる状態すべて，すなわち，M(G) 中の各集合で $\cdot$ の直前に記号 A があるドットルールを含むすべての集合に対して $\epsilon-$ 遷移 を作る．以下にその手続きを示す．

```
function approximation(M(G));
begin
    F := M(G);
    for F の状態中コンプリートドットルールを含む状態 I について do
      for I の全てのコンプリートドットルール Rc について do
        for Rc の左辺の記号 L をドットの直前にもつドットルール
        を持つ F 中の状態 J について do
          I から J に ε− 遷移 をつくる;
    F 中の非終端記号をラベルに持つエッジを全て取り除く;
    return F;
end;
```

図 3 は G_1 を非決定性オートマトンに近似した例である．さらにそれを決定化し最小化を施したものが図 4 である．

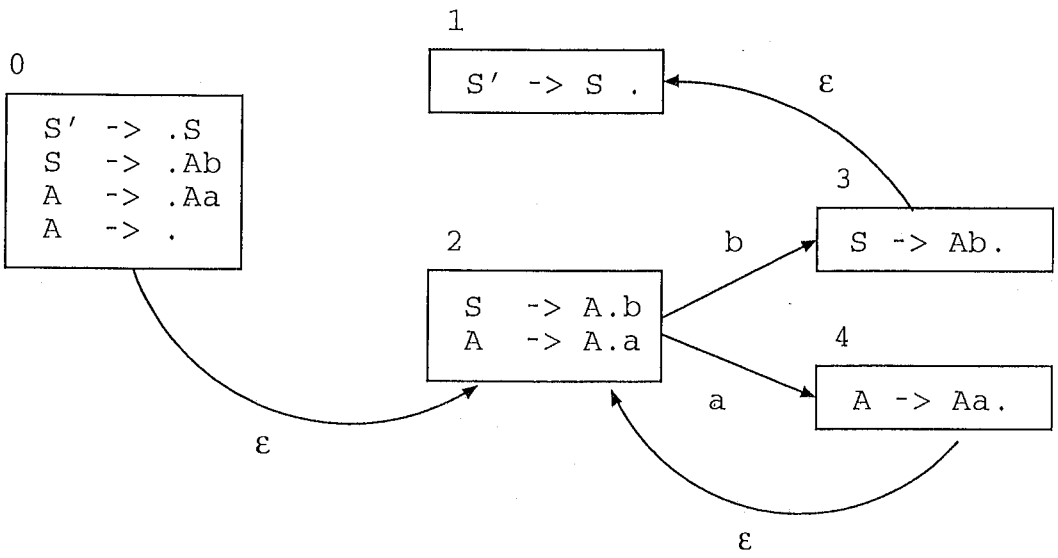

図 3: G_1 を FSA に近似したもの

このアルゴリズムによって近似された FSA F(G) は文法規則 G によって生成される言語 L(G) を全て受理できる．なぜなら，characteristic machine M(G) は G に対する認識器 R(G) の有限状態制御部であり，上記のアルゴリズムにおいて作られた $\epsilon-$ 遷移 先の状態は R(G) の還元動作における状態遷移で遷移しうる状態をすべて含むからである．

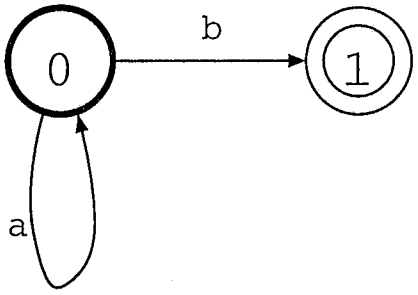

図 4: 図 3 に決定化，最小化を施したもの

ただし，R(G) で還元動作を行なうべき状態でスタック系列を見ずに $\epsilon-$ 遷移 を行なうため，この FSA は本来 L(G) に含まれない言語を受理してしまう．例えば次のような文法 G_2 においては $L(G_2) = \{aca, bcb\}$ となるが，この近似した FSA で受理される言語は $L(F(G_2)) = \{aca, bcb, acb, bca\}$ となってしまう．それは本来 R(G) ならば図 5中状態 5 から状態 4 に遷移するためには，スタックの状態が $\langle 0, a\rangle\langle 2, c\rangle$ という系列になっていなければ遷移できなかったが，FSA 化したことによりそのような制約がなくなってしまったためである (図 6)．

$$G_2 \qquad S \to aXa|bXb$$
$$X \to c$$

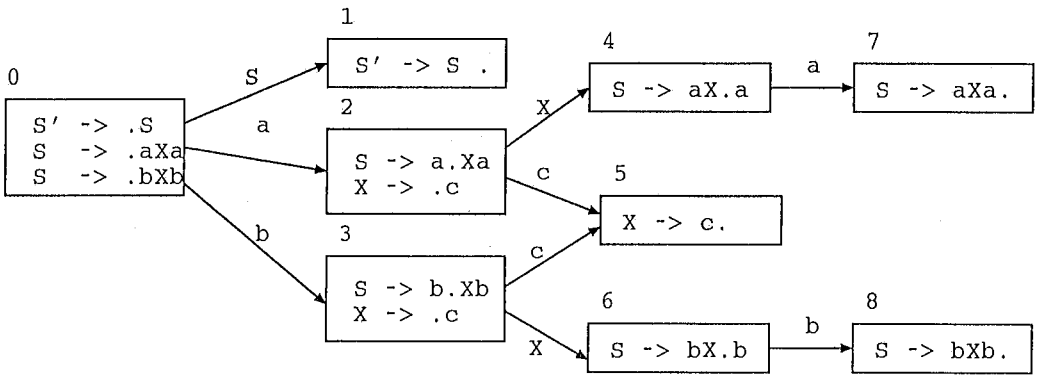

図 5: G_2 の characteristic machine

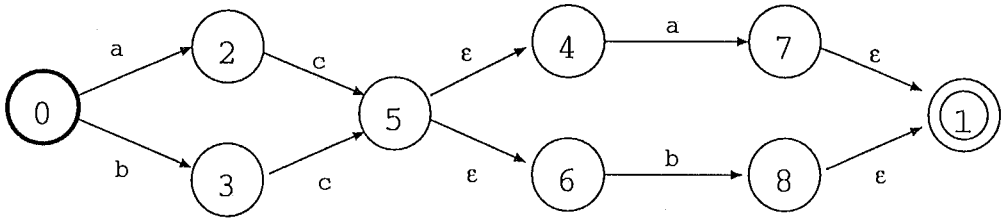

図 6: G_2 を非決定性オートマトンに近似したもの

3 実験

全節においても述べたように，この近似 FSA は CFG で定義された言語を全て受理できるが，逆に CFG では定義されない言語も受理してしまう．このため，言語的制約を FSA に近似

して用いるアプローチは，CFG の制約を直接用いる場合と比べて近似精度と探索空間のトレードオフがある．

そこで，この 2 つの観点から，近似アルゴリズムの評価を行なった．3.1節では近似精度について，3.2節では探索空間についての実験を行なった．

3.1 近似の精度

我々が想定する音声認識タスクにおいて，CFG の生成する言語と，近似 FSA の受理する言語がどれだけ近いかを調べる．自然発話音声データベース [4][5] に含まれる膨大な発話のうち，CFG で生成可能な発話によって，想定タスクにおける CFG の生成する言語を規定する．同様にして，近似 FSA の受理する言語を規定し，両者の一致の度合いを測る．

(3.1.1) 実験内容

全発話文を使って，近似した FSA の受理する発話セットと，CFG で構文解析出来る発話セットを求める．そして，FSA の発話セットが，CFG の発話セットよりどれくらい大きくなっているかを調べるとともに，CFG の発話セット中に FSA の発話セットに含まれない発話が無いことを確認する．

(3.1.2) 実験条件

入力発話文：ホテル予約タスク 565 発話 (テストセット S1,S2,SLTA1)
CFG: 813 生成規則 319 連接規則 341 プレターミナル記号 (クラス)
FSA: CFG を FSA 化し，決定化，最小化を行なったもの (275 状態 , 5167 リンク)

本実験で用いた CFG [6] は，MSLR 法 [7] に基づいて記述されている．この記述法では生成規則を，一般の書き換え規則と連接規則に分けて記述する．この実験で用た FSA は CFG の生成規則を近似アルゴリズムで近似した FSA と，それとは別に連接規則から作成した FSA の結びとして求めた．

(3.1.3) 実験結果

上記の条件で，入力文をそれぞれの手法で受理できるかどうかを調べた結果を表 1 に示す．

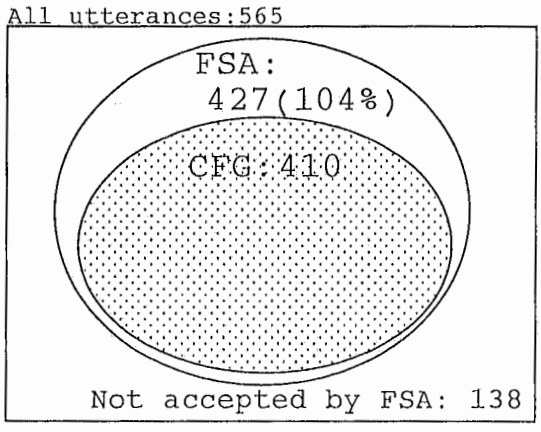

表 1: 実験 1 の結果

CFG で受理された発話は全て FSA で受理されることが確認でき，実装の妥当性が示せた．また，FSA によって受理された文のうち CFG で解析できなかったものは 4% ほどしかなかった．この数字は近似の精度としても妥当なものと考えられる．

3.2 探索空間

一般に音声認識の言語制約として，CFG を持ちいた場合の探索空間は入力発話長の 2 乗オーダで増大する．一方，FSA 用いた場合は，探索空間は入力発話長に比例する．そのため，入力発話が長くなったときほど，FSA による認識の方が有利であると考えられる．そこで，我々の想定する認識タスクにおいて，CFG を用いた時の探索空間の爆発を確認する実験を行なった．

(3.2.1) 実験内容

ホテル予約タスクを想定した発話文の正解系列のうち CFG で解析できるものを CFG による構文解析をし，その入力発話長に対する構文木数を数える．構文木数はその探索空間の大きさを反映していると考えられる．

(3.2.2) 実験条件

入力発話文：ホテル予約タスク 565 発話 (テストセット S1,S2,SLTA1)
CFG: 813 生成規則 319 連接規則 341 プレターミナル記号 (クラス)

(3.2.3) 実験結果

図 7にその結果を示す．構文木数は平均を示した．

構文木数の増加は解析時の探索空間が大きくなることを示している．

このタスクで 20% 以上出現する可能性のある 15 単語を越えた長さの発話系列あたりから桁違いに構文木が増加していることが見てとれる．一方 FSA を用いる場合は，この様な探索空間の爆発は生じないため，長い発話に対しても，効率的な探索ができると考えられる．

4 まとめ

音声認識において句構造文法の知識を効率的に利用することができるように CFG を FSA に変換するツールを Pereira のアルゴリズムに基づいて作成した．また，ホテルの予約タスクを想定した文法において実験を行ない，その実装の妥当性と近似精度の有効性を実験によって確認した．さらに，FSA に近似することにより CFG による計算量的な優位性も確認した．

謝辞

実験に用いるための文脈自由文法規則を提供下さった竹澤 寿幸研究員に心から感謝いたします．本実習で作成したプログラムの ATR SPREC への取り込みにあたり，C++ 固有の問題が発生した際，解決のために多大な御協力をいただいた，Harald Singer 研究員ならびに岩澤 亮祐氏に感謝致します．また，実習のさまざまな面においてお世話をしていただいた第一研究室の皆様に感謝致します．

最後に，本実習の機会を与えて下さった山本 誠一社長，匂坂 芳典 第一研究室室長ならびに牧野 正三 東北大学情報科学研究科教授に心より深く感謝致します．

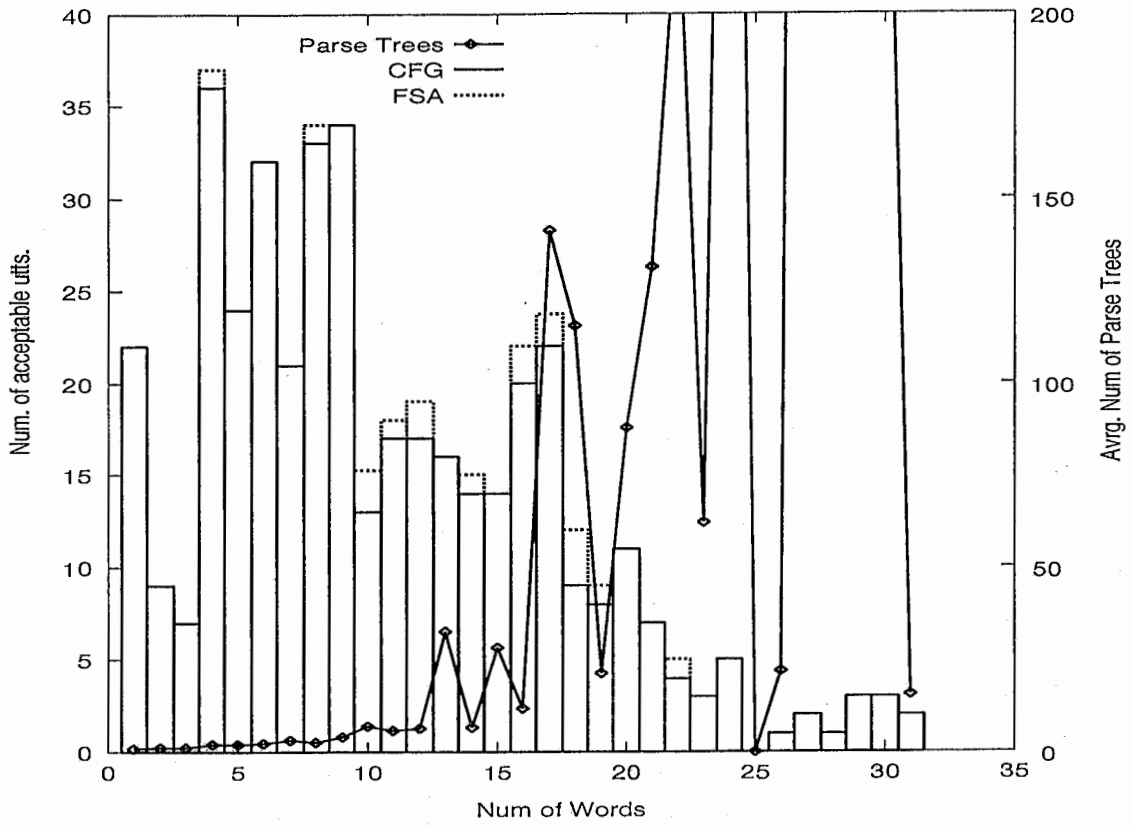

図 7: 1 発話あたりの単語数とその構文木数

参考文献

[1] Fernando C. N. Pereira,Rebecca N. Wright, "Finite-State approximation of phrase structure grammars", In 29th Annual Meeting of the Association for Computational Linguistics, pages 246-255,1991 .

[2] Alfred V.Aho,Ravi Sethi and Jeffrey D. Ullman, "Compilers Principals Techniques and Tools", Addison-Wesley, Reading ,Massachusetts,1986, 邦訳 原田賢一訳, "コンパイラ I 原理・技法・ツール", サイエンス社,

[3] Alan W. Black, "Finite state machines from feature grammars", in Masaru Tomita, editor, International Workshop on Parsing Technologies, pages 277-285,Pittsburgh, Pennsylvania,1989,Carnegie Mellon University.

[4] Morimoto et. al., "Speech and language database for speech translation research," Proc. of ICSLP'94, pp.1791–1794, 1994.

[5] Nakamura et. al., "Japanese Speech Databases for Robust Speech Recognition," Proc. of ICSLP'96, pp.2199–2202, 1996.

[6] 竹澤 寿幸ら, "部分木に基づく構文規則と前終端記号バイグラムを併用する対話音声認識手法", 信学論 (D-II), Vol. J79-D-II, No. 12, 1996.

[7] 田中 穂積, 竹澤 寿幸, 後藤 純司, "MSLR 方を考慮した音声認識用日本語文法 -LR 表工学 (3)-" , 音声言語情報処理 15-25,pages 145-150,1997 .

付録 A　ATRcfg2fsa の利用方法

ATRcfg2fsa は指定された CFG を単語クラス列を受理する FSA に変換するプログラムであり，以下によって起動される．

% ATRcfg2fsa CFG 定義ファイル名 クラス分類ファイル [出力 SLF ファイル名]

出力ファイル名が指定されない場合は標準出力に処理結果を出力する．ここで出力される FSA は SLF 形式で記述され，リンクのラベルとしてクラス分類ファイル名中で定義されたクラス名を持った非決定性有限オートマトンである．なお，オートマトンの決定化，最小化には，ATRf-sautil_minimizer を用いることができる．

ATRcfg2fsa を利用するためには FSA 化したい CFG 文法を定義した**CFG 定義ファイル**と品詞 N-gram 等で用いる単語のクラス分類を記述した**クラス分類ファイル**が必要である．CFG 定義ファイルには単語クラス名の系列を生成する規則を定義する．クラス分類ファイルには，単語クラスおよびそのクラスに属する単語を定義する．

付録 B　CFG 定義ファイル

CFG 定義ファイルは

- 文法開始記号定義行
- 文法規則定義行
- コメント行あるいは空白行

からなる．

文法開始記号定義行は以下のように記述される．

top_node_category([空白 *] 文法開始記号 [空白 *]).

- 文法開始記号定義行は文法規則定義行よりも前にただ 1 行のみ記述されなければならない．
- 文法開始記号は特別な文法記号であり，書き換え規則の開始示す記号である．

文法規則定義行は以下のように記述される．

文法記号 [空白 *]-->[空白 *] 文法記号 [空白 *],...,[空白 *] 文法記号 [空白 *].

- 空白はスペース，TAB 記号を示す．
- 文法規則定義行は 1 行に 1 規則のみ記述すること．
- 文法開始記号定義行と文法規則定義行の各行の最後にピリオド (".") を必ずつけること．
- 各文法記号は小文字および数字で始まる文字列，または' '（シングルクォート）で囲まれた任意の文字列である．ただし，クラス分類ファイルにも表れる記号はクラス分類ファイルの仕様に従うこと．

- コメント行を示す記号として % が使える．% 以降その行はコメントとみなされる．また空白行も挿入することができる．

例:

```
%% CFG rules for ATR CARD
top_node_category(start).

% grammar definitions
start --> 'UTT-START', sent, 'UTT-END'.

sent --> content.
sent --> filler, content.
content --> suit, 'の', number.
content --> suit, 'の', picture.
```