3M-4

インデックスを再利用する再帰下降型解析器の提案

東京理科大学大学院
理工学研究科 情報科学専攻
東 達軌

東京理科大学
理工学部 情報科学科
山口 文彦　山崎 克典

1 はじめに

プログラミング言語やマークアップ言語の構文解析には、主に文脈自由文法 (以下 CFG) の構文解析器が用いられている。しかし、それらの中には CFG で解析できない文法や構造が用いられている場合がある。たとえばプログラミング言語における変数宣言と変数使用の対応や、マークアップ言語の YAML や wiki 書式に見られる箇条書きで木構造を表現する表記等が挙げられる。これらに対しては、構文解析の処理の一部を意味解析に任せ環境を用いる事で、CFG の足りない部分を補って解析を行う事が多い。

しかし、これらの構文を文脈依存文法 (以下 CSG) の構文解析器を用いて解析する研究は、余り見られない。入力文字列から木構造を作成する際に、構文解析の範囲においてその処理を完結させ、それ以降の処理に構文的に正しい構文解析木を渡すことが出来れば、字句解析に始まる一連の解析処理を簡潔に記述できるのではないかと考えた。そのためには、問題領域以上の解析能力を持つ CSG のサブセットを用いる必要がある。そこで CFG に index と呼ばれる要素を追加することで選択的に文脈を残す事が出来る、CSG のサブセットである Indexed Grammar [1] の構文解析器について検討を行った。

特に、Indexed Grammar のサブセットで決定性構文解析が可能なクラスとして strong indexed LL(1) [3](以下 strong ILL(1)) に注目した。strong ILL(1) は strong LL(1) を完全に含む CSG のサブクラスであり、解析手法として strong LL(1) とほぼ同様の表駆動解析を行う事が出来る [2]。だが長さ n の入力文字列の解析で、途中に生成される記号列 (以下、中間生成記号列) の長さは strong LL(1) の場合は最大 n であるが、strong ILL(1) では $O(n^2)$ である。表駆動解析では中間生成記号列をスタックで表すので、記号長が長くなると積み降ろし回数が増加する事があり計算速度において非効率な場合がある。

本稿では strong ILL(1) の構文解析時に中間生成記号列長を短く抑える為の手法として、index 分配時の配置回数を削減する再帰下降型解析器を提案する。

2 Indexed Grammar

Indexed Grammar は 5 つ組 $G = (N, T, I, P, S)$ で表される。N, T, I は記号の有限集合で、それぞれの要素を非終端記号、終端記号、index と呼ぶ。3 つの集合は互いに共通部分を持たない。P は production と indexed production と呼ばれる Indexed Grammar 固有の生成規則の有限集合である。その 2 つを総称して rule と呼ぶことにする。S は開始記号であり、$S \in N$ である。

production は $A \to \alpha$、indexed production は $Af \to \alpha$ の形で表される。ただし、$A \in N, \alpha \in (NI^* \cup T)^*$ である。

2.1 導出

Indexed Grammar $G = (N, T, I, P, S)$ における $\alpha \Rightarrow \beta$ $(\alpha, \beta \in (NI^* \cup T)^*)$ の導出は以下の規則にしたがって行われる。以下で $\gamma, \delta \in (NI^* \cup T)^*$, $f \in I$, $\zeta, \eta_i \in I^*$ $(i = 0, \cdots, k)$ である。

1. production $A \to X_1\eta_1 \cdots X_k\eta_k$ を適用する場合
   $\alpha = \gamma A \zeta \delta$ に対して
   $\beta = \gamma X_1\theta_1 X_2\theta_2 \cdots X_k\theta_k\delta$ を得る。
2. indexed production $Af \to X_1\eta_1 \cdots X_k\eta_k$ を適用する場合
   $\alpha = \gamma A f \zeta \delta$ に対して
   $\beta = \gamma X_1\theta_1 X_2\theta_2 \cdots X_k\theta_k\delta$ を得る。

ただし $\theta_i = \begin{cases} \eta_i\zeta & (X_i \in N) \\ \epsilon & (X_i \in T) \end{cases}$ $(i = 1, \cdots, k)$

rule が非線型 (rule の右辺に非終端記号が複数存在する) の場合、index を配置する行為を index の分配と呼ぶ。

2.2 strong ILL(k)

Indexed Grammar のサブクラスで決定性構文解析が可能な文法クラスに、 strong ILL(k) がある [3]。このクラスは strong LL(k) を完全に含む。Indexed Grammar $G = (N, T, I, P, S)$ が strong ILL(k) であるとは、導出の際に適用する rule が以下の 3 要素のみで一意に定められる事をいう。

- 生成列の最左非終端記号
- その非終端記号に付いている index
- 長さ k の先読み

構文解析手法は strong LL(1) と同じく表駆動で行う方法が提案されている [2, 3]。

3 表駆動解析の問題点と改善案

先に述べたように、表駆動解析の問題点は、長さ n の入力文字列の解析に必要な計算量が、strong LL(1) の場合には $O(n)$ であるのに対して、index 分配が必要な strong ILL(1) では $O(n^2)$ である点である [2]。これは、rule を適用したとき中間生成記号列のスタックに、非終端記号を積む毎に index も積まなければならない事に起因する。

$A\eta$ $(A \in N, \eta \in I^*)$ に rule を適用した際に、右辺の非終端記号に配置する index 列は η の部分列か、η を部分列として持つ列である。そのため、既存の index 列を再利用することで、index の配置回数を削減できると考えた。これは index 列を単方向リストで表現することによって実現する。

また、表駆動解析ではなく再帰下降型解析を用いることで中間生成記号列を再帰計算のスタック中に持つようにすれば、index 分配の際に同時に複数の index 列を生成する必要が無くなる。そうすることで、同じ index 列

が配置される非終端記号に対しては同じ index 列を使い回して配置する事が可能になる。

index 列の再利用と、再帰下降型解析を組み合わせることで解析時の index の配置回数を削減する手法について検討を行った。

4 再帰下降型 strong ILL(1) 解析器

index を再利用するための手法として、再帰下降型 strong ILL(1) 解析器を提案する。

4.1 解析手法

strong ILL(1) での再帰下降型解析では入力文字列の他に、導出の対象となっている終端記号に付いている index 列を表す単方向リストを用いて解析を行う。この リスト を index リスト と呼ぶことにする。

再帰下降型解析を行うための、各非終端記号に対応した手続きは次のように定義する。手続きは、入力として残りの入力文字列と、手続きに対応する非終端記号に付いている index 列をあらわす index リスト をとる。手続きでは入力文字列を先読みして、その記号と index リストの先頭 index に対応する rule が存在するかどうか調べる。存在する場合はその rule に対応する副手続きを呼び出す。存在しない場合は誤りとして手続きを終了する。

rule に対応する副手続きは以下のように定義する。

1. indexed production ならば
   rule の左辺に付いている index と index リスト の先頭が一致しているか判定する。
   - 一致している場合
     index リスト から先頭を除いた部分を この副手続き中の index リスト として用いる (I)
   - 一致しない場合
     誤りとして処理を終了する。
2. 右辺に並んだ記号の順に以下の処理を並べる
   - 終端記号の場合
     先読みとその終端記号が一致しているならば入力文字列を一つ読み進める。先読みと異なる場合誤りとして処理を終了する
   - index の付いていない非終端記号の場合
     その記号に対応する手続きを呼び出す。
   - index の付いている非終端記号の場合
     その index を 現在の index リストの先頭に配置した index リストを作成する。新しく作成した index リストを入力として、その記号に対応する手続きを呼び出す (II)。

(I) が index の消費、(II) が index の配置に対応している。解析は開始記号に対応する手続きを呼び出して行う。その手続き終了時に全ての入力文字列が読まれていた場合、その文字列は受理される。終了時に文字が残っていたり、途中で誤りと解かった場合は誤りである。

先読みと適用すべき rule は strong LL(1) と同様の手法で求めることが出来る [4]。

5 検討と考察

ここで再帰下降型解析では、index の配置回数が最悪の場合でも表駆動解析のそれと一致し、それ以外の場合は表駆動解析よりも少なくなることを示す。

index が n 個付いている非終端記号 A に対して以下の rule を適用したときの配置回数を比較する。記号列 α の記号長を $|\alpha|$ とあらわす事にする。

1. production $A \to X_1\eta_1X_2\eta_2\cdots X_k\eta_k \in P$ を適用した場合
   - 表駆動解析では $X_i \in N$ 毎に $(n + |\eta_i|)$ 回
   - 再帰下降型解析では $X_i \in N$ 毎に $|\eta_i|$ 回
2. indexed production$Af \to X_1\eta_1X_2\eta_2\cdots X_k\eta_k \in P$ を適用した場合
   - 表駆動解析では $X_i \in N$ 毎に $(n - 1 + |\eta_i|)$ 回
   - 再帰下降型解析では $X_i \in N$ 毎に $|\eta_i|$ 回

ただし $i = 1, \cdots, m$ とする。

再帰下降型解析では配置回数は単純に、rule の右辺に現れる index の数だけに依存する。表駆動解析ではそれに加えて、index の付いた非終端記号毎に n 回の配置が追加で発生する。まとめると、rule 右辺の index が付いた非終端記号数を m' としたとき、production を適用した場合は nm' 回、indexed production を適用した場合 $(n-1)m'$ 回だけ再帰下降型解析のほうが配置回数が少ない。それぞれ $n = 0, n = 1$ の時に配置回数の差が 0 となる。再帰下降型解析では rule の右辺に index が現れなければ、長さ n の入力文字列の解析の計算量は $O(n)$ であり、strong LL(1) の表駆動解析と同等である。

ここで YAML [5] の木構造を解析した場合について考察する。YAML の木構造解析はインデントの量を index で表現して行う。木を作っていく際に、兄弟や子へインデントの深さを index 分配によって伝えるため、木が深く広くなるにつれて、index の配置回数は表駆動解析と比較して少なく抑えることが出来る。逆に浅く狭い木の解析では index 分配がほとんど起こらないため、表駆動解析とほぼ同等の回数 index を配置しなければならない。

6 まとめ

本稿で提案した再帰下降型 strong ILL(1) 解析器は、表駆動手法と比較して index の配置回数を抑える事が可能で、導出の際に index が配置されなければ LL(1) と同等の計算量で解析出来ることが分かった。また、木構造の解析では木が大きければ大きいほど index の配置回数を減少させられるので、実用上有用な手法であると考えられる。

今後はこの解析器を用いて CFG では解析出来ない対象の解析を行い、既存手法との計算時間などを比較していきたい。

参考文献

[1] Alfred V.Aho, Indexed Grammars – An Extension of Context-Free Grammar, J. ACM, 15(4):647–671 (1968).

[2] Robert W. Sebesta, Neil D .Jones, Parsers for Indexed Grammar, Internat. J. Comput. and Inform. Sci., 7:345–359 (1978).

[3] R. Parchmann, J. Duske, J. Specht, Indexed LL(k) Grammars, Acta Cybernetica, 7(1):33–53 (1986).

[4] 井上 謙蔵, コンパイラ - プログラム言語処理の基礎, 丸善株式会社 (1994).

[5] YAML.org, `http://www.yaml.org/`.