Time Complexity of MACRO PEG

古賀 智裕*

2016/07/22†

概 要

最適化を行わない first order MACRO PEG(以下, foMPEG と書く) について, ステップ数の上界を 1 つ得ることに成功したので, ここに書き留めておく. また, worst case におけるステップ数の下界について, 指数オーダーよりも真に大きい値を得ることに成功したので, それも書き留めておく. なお, あくまでも上界・下界に過ぎず, それが最良の値であるわけではない.

1 イントロダクション

MACRO PEG [2] とは, PEG [1] にマクロのような機能を持たせて PEG を拡張したものである. 今回の文書では, 最適化を行わない first order MACRO PEG のステップ数について論じる. なお, MACRO PEG の定義の仕方によって, ステップ数の計測の仕方には若干の差異が生じる. この文書では, [3] の定義を MACRO PEG の定義として採用する. よって, その定義のもとでのステップ数ということになる. この文書では, 次の 3 つの定理を証明する:

定理 1.1 (foMPEG のステップ数の上界) $d \geq 1$ とする. $G = (V_T, V_N, R, e_S) \in \mathrm{foMPEG}_d$ を任意に取る (foMPEG_d は, 全ての規則が d 個の引数を取るような foMPEG のこととする). また,

$$D := 3(d+1) \sum_{A \in V_N} |R(A)| + 3(d+1)|e_S| + |V_N| + (d+1)$$

と置く. このとき, 次が成り立つ:

$$\forall v \in V_T^* \left[v \text{ を } e_S \text{に適用すると成功または失敗する} \Rightarrow \text{そのときのステップ数は } 5^{2^{D(|v|+4)!^d}} \text{未満} \right].$$

ただし, V_N の元の個数を $|V_N|$ と書いた. また, $R(A)$ と e_S の parsing expression としての文字列長をそれぞれ $|R(A)|, |e_S|$ と書いた.

補足 語弊がありそうなので補足しておく. 定理 1.1 では,

$$v \text{ を } e_S \text{に適用すると必ず終了し, そのときのステップ数は } 5^{2^{D(|v|+4)!^d}} \text{未満である}$$

と言っているわけではない. 定理 1.1 では,

v を e_Sに適用すると, 一般には無限ループに陥る可能性もあるのだが,
もし無限ループに陥らずに終了したなら, そのときのステップ数は
必ず $5^{2^{D(|v|+4)!^d}}$ 未満になっている

*こが としひろ, toshihiro1123omega_f_ma_mgkvv_sigma_w7.dion.ne.jp _sigma_ → @ omega →
†最終更新日 2018/12/14

という, if 文がついたような内容になっている (定理の主張に「⇒」という論理包含が使われていることに注意). これが何の役に立つのかというと, v を e_S に適用したときに無限ループに陥るか否かを実行時に判定できるのである. なぜなら, v を e_S に適用する前に, 背後で $5^{2^{D(|v|+4)!^d}}$ という値を取得して記憶しておけば (ここでは変数 n を用意して, n に記憶することにする), v を e_S に適用して 1 ステップずつカウントしていったときに,

- そのカウントが n 未満の状況下で終了したなら, それでよい.
- そのカウントが n 以上の状況下でもまだ評価が続いていたら,
  これは無限ループに陥っていることが確定する (定理 1.1 により).

$\cdots$ となるので, これで無限ループが検出できる. ただし, もともとの foMPEG の定義には, このような細工は存在しないので, 実用上は, foMPEG 本体に追加でこのような細工をプログラムすることになる.

補足 e が変数つきの parsing expression のとき, e の文字列長 $|e|$ のカウントの仕方は, [3] での parsing expression の定義に照らし合わせて整理すると, 次のようになる:

- 終端記号・非終端記号・変数はどれも 1 文字であるとカウントする.
- parsing expression の中に出現する 5 種類の文字 "(" ")" "[" "]" "/" は,
  どれも 1 文字であるとカウントする.

定理 1.2 (foMPEG のステップ数の上界) 規則の定義に使われる引数の個数が必ずしも揃ってないような, 一般の first order MACRO PEG $G = (V_T, V_N, R, e_S)$ を任意に取る. G 内の規則の定義で使われる引数の個数の最大値を d とする.

$$D := 3(d+1)(3d+1)\sum_{A \in V_N} |R(A)| + 3(d+1)(3d+1)|e_S| + |V_N| + (d+1)$$

と置く. このとき, 次が成り立つ:

$$\forall v \in V_T^* \left[v \text{ を } e_S \text{に適用すると成功または失敗する} \Rightarrow \text{そのときのステップ数は } 5^{2^{D(|v|+4)!^d}} \text{未満} \right].$$

定理 1.3 (foMPEG の worst case におけるステップ数の下界) 任意の $k \geq 1$ に対して, 次を満たす first order MACRO PEG $G = (V_T, V_N, R, e_S)$ が存在する:

- $\forall v \in V_T^*$ [v を e_Sに適用すると成功し, 文字列の消費は無い].
- $\forall v \in V_T^* \left[v \text{ を } e_S \text{に適用したときのステップ数は少なくとも } |v|^{|v|^k} \text{である} \right]$.

ただし, v の文字列長を $|v|$ と書いた.

補足 最適化を行わない PEG の場合には, 入力文字列の長さを n とするとき, n の多項式オーダーのステップ数では終わらないような $G \in$ PEG が構成できる. さらに,「n の指数オーダー」よりステップ数が嵩むような $G \in$ PEG は作れないことが証明できる. よって, PEG の worst case におけるステップ数は「指数オーダー」までの範囲である. 一方で, 定理 1.3 により, 最適化を行わない foMPEG の場合は, worst case におけるステップ数は指数オーダーよりずっと酷いことになってい

る. 残念ながら, foMPEG の worst case におけるステップ数を確定することはできなかった. 定理 1.2,1.3 によれば, 入力文字列の長さを n とするとき, worst case におけるステップ数は

$$n^{n^k} \text{ から } 5^{2^{D(n+4)!^d}} \text{ の範囲内}$$

となる.

2 定理 1.1,1.2 の証明

定理 1.1 の証明 この定理は任意の foMPEG_d のステップ数を対象にしているため, 本格的に解析しなければ上界の存在性すら疑わしく, 証明の規模もかなりの長さになってしまった. 正式な証明は [3] に書いたので, ここでは概略だけ書くことにする. また, ここでは $d = 1$ の場合だけを書く.

まず, 普通の foMPEG の状態では, 計算経路が複雑になりすぎて記述するのが困難なので, foMPEG を gTS に変換することにする. 従って, gTS の MACRO 版を考案しなければならない. ここでは, 1 変数の first order MACRO gTS を私が勝手に考案して定義する.

$G = (V_T, V_N, R, e_S) \in \text{foMPEG}_1$ は次を満たすとする:

$\forall A \in V_N, \exists a \in V_T, \exists B, C, D, B_1, C_1, D_1 \in V_N$ *s.t.* 次の 5 つのうちいずれかが成り立つ.

- $R(A) = a$.
- $R(A) = x_1$.
- $R(A) = ((a]/((a]])$.
- $R(A) = ((a]a)$.
- $R(A) = (\ (\ B[B_1[x_1]]\ C[C_1[x_1]]\ )\ /\ (\ (\ B[B_1[x_1]\ ]\ ]\ D[D_1[x_1]]\ )\ )$.

さらに, ある $a \in V_T$ とある $Z, Z_1 \in V_N$ に対して

$$e_S = Z[Z_1[((a]/((a]])]]$$

という形であるとする[1]. このとき, G は first order の MACRO gTS であると呼ぶ. このような G 全体の集合を foMgTS_1 と置く. このとき, 任意の $G = (V_T, V_N, R, e_S) \in \text{foMPEG}_1$ に対して, ある $G' = (V_T, V'_N, R', e'_S) \in \text{foMgTS}_1$ が存在して, 次が成り立つことが証明できる:

- $\forall v \in V_T^*$ [v を G, G'に適用すると, その結果は両者で一致する (特に $L(G) = L(G')$ である)].
- $\forall v \in V_T^*$ [v を G に適用すると成功または失敗 $\Rightarrow$ G でのステップ数 $\leq G'$でのステップ数].
- $|V'_N| \leq 6 \sum_{A \in V_N} |R(A)| + 6|e_S| + |V_N| + 2$.

この G' は, 計算経路が簡単に記述できるので, 何とか解析することができて, 次が成り立つことが証明できる:

$$\forall v \in V_T^* \left[v \text{ を } e'_S \text{に適用すると成功または失敗} \Rightarrow \text{そのときのステップ数は } 5^{2^{|V'_N|(|v|+4)!}} \text{未満} \right]. \tag{1}$$

[1] $((a]/((a]])$ は true の代用品であり, $((a]a)$ は false の代用品である.

これらを組み合わせると, $d = 1$ の場合の定理 1.1 が証明できる. そこで, 以下では, (1) の証明の概略を書くことにする. $G' \in \mathrm{foMgTS}_1$ の定義により, ある $a \in V_T$ とある $Z, Z_1 \in V'_N$ に対して

$$e'_S = Z[Z_1[((a]/((a]])]]$$

という形である. 簡単のため, $((a]/((a]])$ のことを t_0 と書くことにする. よって, $e'_S = Z[Z_1[t_0]]$ という形である. ここから出発して, v を適用して計算を進めていくと, 計算経路が「木」として表現できる. 木を根から葉に向かって辿っていくと, 様々な計算経路が出現する. そのような経路のうち, あまりにも長い経路が 1 つでも存在するなら, その経路で無限ループが発生することが示せて,

$$v \text{ を } e'_S \text{に適用すると成功または失敗}$$

という仮定に矛盾する. 従って, どの計算経路も, あまりにも長いものを取ることはできない. このことから, (1) が証明できる. 以下では, あまりにも長い経路がどのような状況を生み出すのかを見ていく. $e'_S = Z[Z_1[t_0]]$ から出発するので,

- $R(Z) = a$,
- $R(Z) = x_1$,
- $R(Z) = ((a]/((a]])$,
- $R(Z) = ((a]a)$,
- $R(Z) = (\ (\ B[B_1[x_1]]\ C[C_1[x_1]]\ )\ /\ (\ (\ B[B_1[x_1]\ ]\ ]\ D[D_1[x_1]]\ )\ )$

の 5 通りが考えられる. 5 行目以外の場合は, 木がほとんど伸びずに葉に到達してしまうので, これは考えなくてよい. 5 行目の場合は,

$$(\ (\ B[B_1[Z_1[t_0]]]\ C[C_1[Z_1[t_0]]]\ )\ /\ (\ (\ B[B_1[Z_1[t_0]]\ ]\ ]\ D[D_1[Z_1[t_0]]]\ )\ )$$

に v を適用することになる. ここでは, v を $B[B_1[Z_1[t_0]]]$ に適用すると成功し, 残りの文字列は v_1 であり, それを $C[C_1[Z_1[t_0]]]$ に適用する場面であると想定してみる. このとき,

$$(Z[Z_1[t_0]], v) \to (C[C_1[Z_1[t_0]]], v_1)$$

という経路が得られる. $R(C)$ は再び 5 行目のようになっていると想定して,

$$R(C) = (\ (\ B'[B'_1[x_1]]\ C'[C'_1[x_1]]\ )\ /\ (\ (\ B'[B'_1[x_1]\ ]\ ]\ D'[D'_1[x_1]]\ )\ )$$

であるとする. よって,

$$(\ (\ B'[B'_1[C_1[Z_1[t_0]]]]\ C'[C'_1[C_1[Z_1[t_0]]]]\ )\ /\ (\ (\ B'[B'_1[C_1[Z_1[t_0]]]\ ]\ ]\ D'[D'_1[C_1[Z_1[t_0]]]]\ )\ )$$

に v_1 を適用することになる. ここでは, v_1 を $B'[B'_1[C_1[Z_1[t_0]]]]$ に適用する場面であるとする. よって,

$$(C[C_1[Z_1[t_0]]], v_1) \to (B'[B'_1[C_1[Z_1[t_0]]]], v_1)$$

という経路が得られる. ここで, $R(B') = x_1$ であるとする. よって,

$$(B'[B'_1[C_1[Z_1[t_0]]]], v_1) \to (B'_1[C_1[Z_1[t_0]]], v_1)$$

という経路が得られる. 今の段階での経路を最初から書き並べると,

$$(Z[Z_1[t_0]], v) \to (C[C_1[Z_1[t_0]]], v_1) \to (B'[B'_1[C_1[Z_1[t_0]]]], v_1) \to (B'_1[C_1[Z_1[t_0]]], v_1)$$

となる. 対 $(*, *)$ の第二成分は, v から出発して, 一般的には先頭の何文字かが削られていくことになる. よって, 経路があまりにも長い場合, 対 $(*, *)$ の第二成分がずっと同じ w である部分経路が取れて, しかもその部分経路自体があまりにも長いようにできる $\cdots$(2) また, 対 $(*, *)$ の第一成分は,

$$Z[Z_1[t_0]] \to C[C_1[Z_1[t_0]]] \to B'[B'_1[C_1[Z_1[t_0]]]] \to B'_1[C_1[Z_1[t_0]]]$$

の流れを見ても分かるように,

(3) 先頭の非終端記号が削られる,

(4) 先頭の非終端記号が削られて, かわりに "$B[B_1[$" という形状の非終端記号が 2 つ加わる

の 2 種類の動作のみが起こる. それ以外の動作も実際には起こるが, それらは

- $R(X) = a$,
- $R(X) = ((a]/((a]])$,
- $R(X) = ((a]a)$,

であるから, そこで葉に到達して経路が止まってしまうので, 考える必要がない. さて, (2) の部分経路に (3),(4) を合わせれば, v のあるサフィックス w が存在して,

$$(e_1, w) \to (e_2, w) \to (e_3, w) \to \cdots \to (e_m, w) \tag{5}$$

という部分経路が得られて, 各 e_i は

$$e_i = X_{i1}[X_{i2}[\cdots[X_{ik_i}[t_0]]\cdots]],\ k_i \geq 1$$

という形状をしていて, しかも e_i から (3),(4) のうちどちらかを 1 回だけ使って e_{i+1} に移れるという状況になっている. この状況下で, あまりにも m の値が大きいと, (5) の経路が無限ループを生成することを示したい. まず,

$$e_i = X_{i1}[X_{i2}[\cdots[X_{ik_i}[t_0]]\cdots]] = X_{i1}[e'_i],\ e'_i := X_{i2}[\cdots[X_{ik_i}[t_0]]\cdots]$$

と表記する. 各 e'_i の動作は, 今回の経路上においては,

- $w^{+0} = w$ を e'_iに適用したときの結果,
- w^{+1}を e'_iに適用したときの結果,

$\vdots$

- $w^{+|w|} = \varepsilon$を e'_iに適用したときの結果

の $(|w| + 1)$ 個の結果のみで決定される. ただし, w^{+k} を e'_i に適用したときの結果は

- (成功, w^{+k}のサフィックス) ($|w^{+k}| + 1$ 通りある),
- 失敗 (1 通り),
- 無限ループ (1 通り)

に分かれるので, 各 w^{+k} ごとに $(|w^{+k}|+3)$ 通りある. よって, 全体では

$$\prod_{k=0}^{|w|}(|w^{+k}|+3)=\prod_{k=0}^{|w|}(|w|-k+3)=\prod_{k=3}^{|w|+3}k=(|w|+3)!/2$$

通りとなる. よって, e'_i の動作は, 今回の経路上においては, $(|w|+3)!/2$ 通りに分類されることになる. よって, $m\geq(|w|+3)!/2+1$ ならば, 鳩ノ巣原理から, $e'_1,\cdots,e'_m$ の中のある異なる e'_i と e'_j が同一のタイプに属することになる. このとき, 任意の $A_1,\cdots,A_p\in V'_N$ に対して

$$A_1[A_2[\cdots[A_p[e'_i]]\cdots]],\ A_1[A_2[\cdots[A_p[e'_j]]\cdots]]$$

の 2 つもまた, 同じタイプに属する $\cdots(6)$ ことが証明できる. なお, このときの i と j について,

$$e_i=X_{i1}[e'_i],\ e_j=X_{j1}[e'_j]$$

における X_{i1} と X_{j1} は必ずしも $X_{i1}=X_{j1}$ を満たしていない. しかし, もともとの m が $m\geq|V'_N|\cdot(|w|+3)!/2+1$ を満たすならば, 鳩ノ巣原理を 2 回使うことにより, ある異なる i,j が存在して

- e'_iと e'_jは同じタイプに属する,
- $X_{i1}=X_{j1}$である

が成り立つことが分かる. この状態で, (5) の経路のうち

$$(e_i,w)\to(e_{i+1},w)\to\cdots\to(e_j,w)$$

という部分経路を考えれば, (6) により,

- $e_i=X_{i1}[e'_i]$ に (3) が施されて e_{i+1}に移るなら, $e_j=X_{j1}[e'_j]$ にも (3) が施されて e_{j+1}に移る.
- $e_i=X_{i1}[e'_i]$ に (4) が施されて e_{i+1}に移るなら, $e_j=X_{j1}[e'_j]$ にも (4) が施されて e_{j+1}に移る.
  しかも, 追加される "$B[B_1[$" は e_iと e_jで同じもの.

というシンクロした状況になることが証明できる. 同じく,

- e_{i+1}に (3) が施されて e_{i+2}に移るなら, e_{j+1}にも (3) が施されて e_{j+2}に移る.
- e_{i+1}に (4) が施されて e_{i+2}に移るなら, e_{j+1}にも (4) が施されて e_{j+2}に移る.
  しかも, 追加される "$B[B_1[$" は e_{i+1}と e_{j+1}で同じもの.

というシンクロした状況になることもまた期待される (ただし, 今のままでは欠陥があり, ここは必ずしも成り立たない). このシンクロする性質が $e_i\to e_{i+1}\to\cdots\to e_j$ まで続いたならば, (3),(4) による非終端記号の増減がそれ以降も周期的に繰り返されることが期待されて, (5) は無限ループを生成することが期待される. よって, 当面の問題は, このシンクロする性質がきちんと $e_i\to e_{i+1}\to\cdots\to e_j$ まで続くのかということである. もし (4) のみが適用されるのであれば, 明らかにシンクロする性質が続く. 問題は, 途中で (3) が適用されたときである. 特に, しょっぱなの $e_i=X_{i1}[e'_i]$ に (3) が適用されて e'_i になったときは問題が起こる. なぜなら, このときは $e_{i+1}=e'_i$ となる. 同じく, $e_j=X_{j1}[e'_j]$ にも (3) が施されて e'_j となるので, $e_{j+1}=e'_j$ となる. e'_i と e'_j は同一のタイプに属するのだったが, あくまでもタイプが同じだけである.

$$e'_i=X_{i2}[X_{i3}[\cdots[X_{ik_i}[t_0]]\cdots]],$$
$$e'_j=X_{j2}[X_{j3}[\cdots[X_{jk_j}[t_0]]\cdots]]$$

であるから, 必ずしも $X_{i2}=X_{j2}$ であるとは限らない. もし $X_{i2}\neq X_{j2}$ ならば, もはやシンクロするとは限らず, 失敗する. また, もし $X_{i2}=X_{j2}$ であったとしても,

$$e'_i=X_{i2}[e''_i],\ e''_i:=X_{i3}[\cdots[X_{ik_i}[t_0]]\cdots],$$
$$e'_j=X_{j2}[e''_j],\ e''_ij=X_{j3}[\cdots[X_{jk_j}[t_0]]\cdots]$$

と置くときに, e''_i と e''_j が同じタイプになる保証はない. この場合, $e_{i+1}=e'_i$ に (3) または (4) を施して e_{i+2} になったとしても, $e_{j+1}=e'_j$ には (3),(4) のうち別の方が適用される可能性があり, もはやシンクロしなくなってしまい, 失敗する. これらの失敗の原因をよく考えると, 要するに, $e_{i+1}=e'_i$ において e'_i がむき出しの状態になっていることが失敗の原因なのである. 同様に, $e_i\to e_{i+1}\to\cdots\to e_j$ という推移の中のいずれかで e'_i がむき出しになってしまったら, そこで失敗する. 逆に, $A_1[A_2[A_3[e'_i]]]$ のように, いつまでも手前に非終端記号がついていて, e'_i がむき出しの状態にならないならば, このような失敗は起きず, 上の戦略が成功して, (5) は無限ループを生成することが証明できる. すなわち, ある異なる i,j に対して

(7) e'_iと e'_jは同じタイプに属する.

(8) $X_{i1}=X_{j1}$である.

(9) $e_i\to\cdots\to e_j$という経路において,e'_iはむき出しにならない

が成り立つのであれば, (5) は無限ループを生成することが証明できる. しかし, (7),(8),(9) はあまりにも都合の良い性質であるため, いつでもそのような状況が起こせるとは考えにくい. ここが, 今回の議論における最大の壁となる. そこで, ラムゼー型の定理を自作し, それを証明することで, この壁を回避する. 実は, m の値が十分大きければ, あまりにも都合が良いと思われた (7),(8),(9) が, 100%発生するのである. 以下では, ラムゼー型の定理に落とし込むために, いくつかの準備をする. $d\geq 1$ とする. 任意の $a\in\cup_{n\in\mathbb{N}}\mathbb{R}^n$ に対して, $a\in\mathbb{R}^n$ を満たす $n\geq 1$ がただ 1 つ存在する. この n のことを $|a|$ と表記する. 次に, $a\in\cup_{n\in\mathbb{N}}\mathbb{R}^n$ を任意に取る. $(i_j)_{j=1}^d\in[1,|a|]^d$ は狭義単調増加とする.

$$\forall j\in[1,d],\forall k\in[i_j,i_d]\ \left[\ a_{i_j}\leq a_k\ \right]$$

が成り立つとする. このときの $(a_{i_j})_{j=1}^d\in\mathbb{R}^d$ を, a 内の正規 d 列と呼ぶことにする. 次に, $s,n\in\mathbb{N}$ とする.

$$M_n(s):=\{a\in\cup_{n\in\mathbb{N}}\mathbb{N}^n\mid |a|\geq n, a_1\in[1,s], a_{i+1}-a_i\in[-1,1]\ (1\leq i<|a|)\}$$

と置く. このとき, 次が成り立つことが証明できる:

定理 2.1 (正規 d 列に関するラムゼー型の定理)

$$\forall s\geq 1,\forall d\geq 1,\forall a\in M_{2^{d-1}s}(s)\ \left[\ a\text{ 内には正規 }d\text{ 列が存在する }\right].$$

さて, (3),(4) で非終端記号の個数の変動が ±1 になっていることに注意して,

$$a_i:=「e_i=X_{i1}[X_{i2}[\cdots[X_{ik_i}[t_0]]\cdots]]\text{ における }X_{i*}\text{の個数」}=k_i\in\mathbb{N}$$

とおくことで, 上記のラムゼー型の定理が適用できて, そこからさらに計算を再帰的に重ねて, (7),(8),(9) を満たす異なる i,j が存在することが実際に証明できる. 以上の議論により, 計算経路があまりにも長いと, (5) は無限ループを生成することが証明できる. 具体的にどのくらい長ければ無限ループが発生するのかを詳しく見ると, 定理 1.1 が得られる. 以上で, 定理 1.1 の解説を終える. 冒頭でも書いたが, 正式な証明は [3] にあるので, そちらを参照のこと.

補足　ラムゼー型の定理が必要になる理由は, PEG からの類推で考えると, 次のようになる: まず, PEG の場合は, gTS の計算経路の上に出現する非終端記号が有限種類しかなく, その非終端記号に適用する入力文字列も計算経路が進むごとに削られていって減少するので, 計算経路があまりにも長い場合, 単純な鳩ノ巣原理を使うだけで, 無限ループの発生が簡単に証明できる. このことから, PEG の計算経路は「あまり長くできない」ことになり, ステップ数の上界が具体的に求まる (指数オーダー程度になる).
一方で, MACRO PEG の場合は, 非終端記号が引数を取れるので, 非終端記号の種類は実質的には無限個となってしまい, 単純な鳩ノ巣原理では, 無限ループの発生が証明できない. しかし, 無限個あると思われた非終端記号には一種の法則性があるので, そこを詳しく解析すれば何とかなりそうに見える. この解析は, 単純な鳩ノ巣原理では歯が立たず, もう少し高度なツールが必要になる. 鳩ノ巣原理を高度に昇華したものはラムゼー理論と呼ばれるので, これが, ラムゼー型の定理が必要になる理由だと考えられる.

定理 1.2 の証明　一般に, 任意の first order MACRO PEG $G = (V_T, V_N, R, e_S)$ に対して, G 内の規則で使われる引数の個数の最大値を d とするとき, ある $G' = (V_T, V'_N, R', e'_S) \in \text{foMPEG}_d$ が存在して,

- $\forall v \in V_T^*$ [v を G, G'に適用すると, その結果は両者で一致する (特に $L(G) = L(G')$ である)],
- $\forall v \in V_T^*$ [v を G に適用すると成功または失敗 $\Rightarrow$ G でのステップ数 $\leq$ G'でのステップ数]

が成り立つことが言える. たとえば,

$$
\begin{aligned}
&V_T := \{a, b\}\,,\ V_N = \{A, B, C\}\,,\\
&A[z][y][x] = axyz;\quad B[s][t] = sA[t][t][t]bC;\quad C = a;\\
&e_S := B[b][b]
\end{aligned}
$$

として $G = (V_T, V_N, R, e_S) \in$ foMPEG を定義する[2]と, 各規則ごとに引数の個数が違っている. そこで, まずは, 各規則に使われる引数の個数の最大値を求める. 上の例の場合は「3」である. 次に, 3 つの異なる文字 x_1, x_2, x_3 を取り, 各規則の変数をこれらの文字に置換して次のようにする:

$$
\begin{aligned}
&V_T := \{a, b\}\,,\ V_N = \{A, B, C\}\,,\\
&A[x_1][x_2][x_3] = ax_3x_2x_1;\quad B[x_1][x_2] = x_1A[x_2][x_2][x_2]bC;\quad C = a;\\
&e_S := B[b][b]
\end{aligned}
$$

次に, 各規則の左辺に注目する. 引数が 3 個未満のものは, 人工的に引数の個数を 3 個まで増やす:

$$
\begin{aligned}
&V_T := \{a, b\}\,,\ V_N = \{A, B, C\}\,,\\
&A[x_1][x_2][x_3] = ax_3x_2x_1;\quad B[x_1][x_2][x_3] = x_1A[x_2][x_2][x_2]bC;\quad C[x_1][x_2][x_3] = a;\\
&e_S := B[b][b]
\end{aligned}
$$

次に, 各規則の右辺に注目する. 右辺に出現する非終端記号のうち, 引数が 3 個未満になっているものは, 人工的に引数の個数を 3 個まで増やし, その中身には a を指定する:

$$
\begin{aligned}
&V_T := \{a, b\}\,,\ V_N = \{A, B, C\}\,,\\
&A[x_1][x_2][x_3] = ax_3x_2x_1;\quad B[x_1][x_2][x_3] = x_1A[x_2][x_2][x_2]bC[a][a][a];\quad C[x_1][x_2][x_3] = a;\\
&e_S := B[b][b][a]
\end{aligned}
$$

[2]parsing expression に使われるカッコは適宜省略した (ここでは本質的ではないので).

最後に, V_T, V_N, R, e_S という記号を V'_T, V'_N, R', e'_S とでも変更して

$$V'_T := \{a, b\}\,,\ V'_N = \{A, B, C\}\,,$$
$$A[x_1][x_2][x_3] = ax_3x_2x_1;\quad B[x_1][x_2][x_3] = x_1A[x_2][x_2][x_2]bC[a][a][a];\quad C[x_1][x_2][x_3] = a;$$
$$e'_S := B[b][b][a]$$

と表記する. このとき, $G' := (V'_T, V'_N, R', e'_S) \in \mathrm{foMPEG}_3$ が成り立つ. また, 任意の入力文字列 v に対して, v を e_S と e'_S に適用すると, その結果は両者で一致する. さらに, 結果が成功または失敗のときは, 「e_S でのステップ数」≤「e'_S でのステップ数」が成り立つ (引数の個数の解析で生じるステップ数を無視するなら, ステップ数は一致する). こうして, G から G' を作ったとき,

$$D' := 3(d+1)\sum_{A\in V'_N}|R'(A)| + 3(d+1)|e'_S| + |V'_N| + (d+1)$$

と置くと, 定理 1.2 より, 次が成り立つ:

$$\forall v \in V_T^*\ \left[\ v \text{ を } e'_S \text{に適用すると成功または失敗する} \Rightarrow \text{そのときのステップ数は } 5^{2^{D'(|v|+4)!^d}} \text{未満}\ \right].$$

「e_S でのステップ数」≤「e'_S でのステップ数」が成り立つのだったから, e_S から出発すれば, 次が成り立つ:

$$\forall v \in V_T^*\ \left[\ v \text{ を } e_S \text{に適用すると成功または失敗する} \Rightarrow \text{そのときのステップ数は } 5^{2^{D'(|v|+4)!^d}} \text{未満}\ \right].$$

さて, G' の作り方から, $V'_N = V_N$ となる. また, e_S と e'_S の文字列としての違いは, e_S の中に出現する非終端記号の引数が増設されるか否かの違いしかない. また, 引数を 1 つ増設するごとに, "$[a]$" という形の 3 文字の文字列が 1 つ増えるだけである. よって, e_S の全ての文字が引数 0 の非終端記号で出来ているときが最も変化が大きく, 一般には

$$|e'_S| \le |e_S| + |e_S| \cdot 3d = (3d+1)|e_S|$$

となる. 同様にして, $R(A)$ と $R'(A)$ でも, "$[a]$" という形の 3 文字の文字列が増えるか否かの違いしかない. よって

$$|R'(A)| \le (3d+1)|R(A)|$$

となる. よって,

$$\begin{aligned}
D' &= 3(d+1)\sum_{A\in V'_N}|R'(A)| + 3(d+1)|e'_S| + |V'_N| + (d+1)\\
&= 3(d+1)\sum_{A\in V_N}|R'(A)| + 3(d+1)|e'_S| + |V_N| + (d+1)\\
&\le 3(d+1)\sum_{A\in V_N}(3d+1)|R(A)| + 3(d+1)(3d+1)|e_S| + |V_N| + (d+1)\\
&= D
\end{aligned}$$

となるので,

$$5^{2^{D'(|v|+4)!^d}} \le 5^{2^{D(|v|+4)!^d}}$$

となる. 以上を繋げると, 定理 1.2 が得られる.

3　定理 1.3 の証明

定理 1.3 の証明　以下の first order MACRO PEG を考える (簡単のため, parsing expression に使われるカッコは適宜省略した):

V_T := {a}　V_N := { T, TT, G1, GG1, G2, GG2, G3, GG3, true }

```
true = (a] / ((a]];

T[x] = TT[true][x];
TT[i][x] = (( i a ]] x TT[ i a ][x] / ( i a ];

G1[x] = GG1[true][x];
GG1[i][x] = (( i a ]] GG1[ i a ][T[x]] / ( i a ] x;

G2[x] = GG2[true][x];
GG2[i][x] = (( i a ]] GG2[ i a ][G1[x]] / ( i a ] x;

G3[x] = GG3[true][x];
GG3[i][x] = (( i a ]] GG3[ i a ][G2[x]] / ( i a ] x;
```

e_S := G3[true]

このとき, 次が成り立つことが証明できる:

(1) $\forall v \in V_T^*$ [v を e_Sに適用すると成功し, 文字列の消費は無い].

(2) $\forall v \in V_T^*$ $\left[$ v を e_Sに適用したときのステップ数は少なくとも $|v|^{|v|^3}$ である $\right]$.

実際, 入力文字列の長さを n とするとき, T[x] によって

x x x ⋯ x　(x が n 個)

が実行される. また, G1[x] によって

T[T[T[⋯ T[x] ⋯]]]　(T が n 個)

が実行される. また, G2[x] によって

G1[G1[G1[⋯ G1[x] ⋯]]]　(G1 が n 個)

が実行される. また, G3[x] によって

G2[G2[G2[⋯ G2[x] ⋯]]]　(G2 が n 個)

が実行される. 以上より, G3[x] によって

x x x ⋯ x　(x が n^{n^3} 個)

が実行される. よって, e_S によって

`true true true` $\cdots$ `true`　(`true` が n^{n^3} 個)

が実行されるので, (1),(2) が成り立つ. 以上より, $k = 3$ のときの定理 1.3 が証明できた. 一般の k に対しては, `G1,G2,`$\cdots$`,Gk` の k 個まで作ればよい.

参考文献

[1] Ford, B.: Parsing Expression Grammars: A Recognition-Based Syntactic Foundation. In: POPL 2004: Proceedings of the 31st ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 111-122. ACM, New York (2004)

[2] Kota Mizushima: Macro PEG: PEG with macro-like rules
https://github.com/kmizu/macro_peg

[3] Toshihiro Koga: MACRO PEG のお勉強
https://github.com/T-K-1/peg_and_macro_peg/blob/master/macro_peg_note.pdf