PEG and MACRO PEG

古賀 智裕*

2016/05/28†

概 要

PEL が first order の MACRO PEL に真に包含されることが示せたので, ここに書き留めておく. PEG では表現できず, first order の MACRO PEG なら表現できるような言語を具体的に構成する.

1 イントロダクション

MACRO PEG [3] とは, PEG [2] 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. その定義から, 明らかに PEL ⊂ MACRO PEL が成り立つ. ここでの興味はもちろん, PEL ⊊ MACRO PEL が成り立つか否かである. PEG では表現できず, MACRO PEG なら表現できるような言語が具体的に見つかれば, PEL ⊊ MACRO PEL が成り立つ. たとえば, MACRO PEG は「回文」を表現できる [3] が, 回文を PEG で表現するのは不可能であると予想されている. よって, PEL ⊊ MACRO PEL が実際に成り立つことが期待される.
ただし, この方針では, 回文が PEG で表現できないことを証明しなければならず, 現時点では未解決問題である. そこで, もっと簡単に解析できる言語が欲しい. この文書では, そのような言語 L として, PEG の範囲では一種の自己言及を起こしてしまうものを定義する. これにより, その L は PEG では表現できないことが簡単に示される (よくある対角線論法を使って). また, この L は MACRO PEG で表現可能であることも示す. MACRO PEG で gTS をエミュレートすることに成功したので, それを使って, L を MACRO PEG で表現する. 以上により, PEL ⊊ MACRO PEL が成立する. なお, この文書で実際に示すのは

$$\text{PEL} \subsetneq \text{first order MACRO PEL}$$

である. first order MACRO PEL ⊂ MACRO PEL であるから, PEL ⊊ MACRO PEL を示すよりも更に少しだけ良い結果となる.

2 PEG と gTS と MACRO PEG の定義

PEG, gTS, MACRO PEG の定義の仕方には色々な流儀が考えられるが, ここでは [2] の流儀に近いものを採用する.

*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

定義 2.1 正整数全体の集合を $\mathbb{N}$ と置く．空集合でない記号の集合 B(有限集合でなくてもよい) に対して, B^* は B のクリーネ閉包とし，B^+ は B のクリーネプラスとする. 空文字列を ε で表す. $v \in B^*$ に対して, v の文字列長を $|v|$ と書く．$v, w \in B^*$ について, $w = vv'$ を満たす $v' \in B^*$ が取れるとき, かつそのときのみ $v \prec w$ と書く．すなわち, v が w のプレフィックスであるとき, かつそのときのみ $v \prec w$ と書く．$v \in B^*$ に対して, v の先頭の 1 文字を削った残りの文字列のことを v^{+1} と書くことにする．同様にして v^{+n} を定義する. たとえば, $(abcde)^{+1} = bcde$, $(abcde)^{+4} = e$, $\varepsilon^{+1} = \varepsilon$ などが成り立つ. $v \in B^*$ に対して, v を左右反転させた文字列を ${}^t v$ と書くことにする．たとえば ${}^t(abcde) = edcba$ である.
ここから先の文章において, Σ, Σ' は空集合でない記号の集合 (有限集合でなくてもよい) とし，Σ, Σ' は以下の 4 種類の記号を含まないとする.

$$(\quad)\quad]\quad /$$

さらに, V_1, V_2 はそれぞれ記号の可算無限集合とし，$V_1 \cap V_2 = \emptyset$ が成り立っているものとする．さらに, $V_1 \cup V_2$ もまた, 上記の 4 種類の記号を含まないとする．この文書では, PEG における終端記号は V_1 の元から取ってくるようにして，非終端記号は V_2 の元から取ってくるようにする.

定義 2.2 (parsing expression の定義) 次の 2 行を満たす集合 M のうち, 集合の包含関係に関して最小のものが存在することが言える:

- $\forall a \in \Sigma\ [\ a \in M\ ]$,
- $\forall e_1, e_2 \in M\ [\ (e_1 e_2),\ (e_1],\ (e_1/e_2) \in M\ ]$.

そこで, そのような最小の集合のことを $PE(\Sigma)$ と書く. 従って, 次が成り立つ:

- $M = PE(\Sigma)$ と置くと，この M は上の 2 行を満たす.
- 上の 2 行を満たす任意の集合 M に対して，$PE(\Sigma) \subset M$ が成り立つ.

$PE(\Sigma)$ の各元のことを parsing expression と呼ぶ. $\Sigma \subset \Sigma'$ のとき，$PE(\Sigma) \subset PE(\Sigma')$ が成り立つことが証明できる.

補足 $(e_1 e_2)$ は連接のつもりであり, (e_1/e_2) は優先度つき選択のつもりであり，$(e_1]$ は $!e_1$ (NOT predicate) のつもりである. $e*$, $e+$, $e?$, $\&e$ は PEG には必須ではなく, 他の演算子と非終端記号の合わせで実現可能であるから，ここでは定義しない.

定義 2.3 (PEG の定義) X, Y は集合とする. X から Y への写像全体の族を $(X \to Y)$ と書く. さらに,

$$(X \to Y)_0 := \bigcup_{A \subset X,\ B \subset Y} (A \to B)$$

と定義する.

$$PEG \subset \mathfrak{P}(V_1) \times \mathfrak{P}(V_2) \times (V_2 \to PE(V_1 \cup V_2))_0 \times PE(V_1 \cup V_2)$$

を以下のように定義する:

$$(V_N, V_T, R, e_S) \in PEG \ \Leftrightarrow_{def}\ V_T \subset V_1 \text{は空でない有限集合},\ V_N \subset V_2 \text{は有限集合 (空でもよい)},$$
$$R \in (V_N \to PE(V_T \cup V_N)),\ e_S \in PE(V_T \cup V_N).$$

$(V_N, V_T, R, e_S) \in PEG$ に対して, V_T の元のことを「終端記号」と呼び, V_N の元のことを「非終端記号」と呼ぶ. e_S は start expression (開始表現) と呼ばれる. $(A, e) \in R$ のとき, ——同じことだが, $R(A) = e$ のとき,

$$A \underset{R}{\leftarrow} e$$

と書く. 混乱の恐れがないときには, 単に

$$A \leftarrow e$$

と書く. 次に, $B \subset V_1$ は空でない有限集合とする.

$$PEG(B) := \{(V_N, V_T, R, e_S) \in PEG \mid V_T = B\}$$

と定義する.

定義 2.4 (parsing expression の解釈) 空でない有限集合 $V_T \subset V_1$ を任意に取り, 有限集合 $V_N \subset V_2$ を任意に取る (こちらは空でもよい). 記号 $\mathtt{f}$ は V_T^* には含まれないとする. $a \in V_T$ と $v \in V_T^*$ に対して

$$\langle v \rangle_a := v^{+1} \quad (a \prec v \text{ のとき}), \quad \mathtt{f} \quad (\neg(a \prec v) \text{ のとき})$$

と定義する. また, $v \in V_T^*$ と $v' \in V_T^* \cup \{\mathtt{f}\}$ に対して

$$\langle v, v' \rangle := v \quad (v' = \mathtt{f} \text{ のとき}), \quad \mathtt{f} \quad (v' \neq \mathtt{f} \text{ のとき})$$

と定義する. $R : V_N \to PE(V_T \cup V_N)$ を任意に取る. 次を全て満たす集合 M のうち, 集合の包含関係に関して最小のものが存在することが証明できる:

- $\forall a \in V_T,\ \forall v \in V_T^*\ [\ (a, v, \langle v \rangle_a) \in M\ ]$,
- $\forall A \in V_N,\ \forall v \in V_T^*,\ \forall v_f \in V_T^* \cup \{\mathtt{f}\}\ [\ (R(A), v, v_f) \in M \Rightarrow (A, v, v_f) \in M\ ]$,
- $\forall e_1, e_2 \in PE(V_T \cup V_N),\ \forall v, v_1 \in V_T^*,\ \forall v_f \in V_T^* \cup \{\mathtt{f}\}$ *s.t.* 次の 5 行が全て成り立つ :
  - $(e_1, v, v_f) \in M \Rightarrow ((e_1], v, \langle v, v_f \rangle) \in M$,
  - $(e_1, v, v_1),\ (e_2, v_1, v_f) \in M \Rightarrow ((e_1 e_2), v, v_f) \in M$,
  - $(e_1, v, \mathtt{f}) \in M \Rightarrow ((e_1 e_2), v, \mathtt{f}) \in M$,
  - $(e_1, v, v_1) \in M \Rightarrow ((e_1 / e_2), v, v_1) \in M$,
  - $(e_1, v, \mathtt{f}),\ (e_2, v, v_f) \in M \Rightarrow ((e_1 / e_2), v, v_f) \in M$.

そこで, そのような最小の集合のことを $\rho(V_N, V_T, R)$ と置く. 従って, 次が成り立つ:

- $M = \rho(V_N, V_T, R)$ と置くと, この M は上記の条件を満たす.
- 上記の条件を満たす任意の集合 M に対して, $\rho(V_N, V_T, R) \subset M$ が成り立つ.

次に, $(e,v) \in PE(V_T \cup V_N) \times V_T^*$ を任意に取る.

- ある $v' \in V_T^*$が存在して $(e,v,v') \in \rho(V_N,V_T,R)$ が成り立つとき, $(e,v) \xrightarrow[V_N,V_T,R]{} \mathtt{t}$ と書く.
  このとき,「v を e に適用すると成功する」と呼ぶ.
- $(e,v,\mathtt{f}) \in \rho(V_N,V_T,R)$ が成り立つとき, $(e,v) \xrightarrow[V_N,V_T,R]{} \mathtt{f}$ と書く.
  このとき,「v を e に適用すると失敗する」と呼ぶ.
- $(e,v,v_f) \in \rho(V_N,V_T,R)$ となる $v_f \in V_T^* \cup \{\mathtt{f}\}$ が存在しないとき, $(e,v) \xrightarrow[V_N,V_T,R]{} \infty$ と書く.
  このとき,「v を e に適用すると無限ループに陥る」と呼ぶ.

任意の $(e,v) \in PE(V_T \cup V_N) \times V_T^*$ に対して, 上記の 3 つのケースのうちちょうど 1 つだけが成り立つことが証明できる. また, $(e,v) \xrightarrow[V_N,V_T,R]{} \mathtt{t}$ ならば, $(e,v,v') \in \rho(V_N,V_T,R)$ を満たす $v' \in V_T^*$ はただ 1 つであり, しかも v' は v のサフィックスであることが証明できる. よって, $v = v''v'$ という形に表せる. このときの v'' のことを, 「v を e に適用して成功したときに消費される文字列」と呼ぶ. また, 各 V_N, V_T, R ごとに, 次を満たす $\varphi : \rho(V_N,V_T,R) \to \mathbb{N}$ がただ 1 つ存在することが証明できる.

- $\forall a \in V_T,\ \forall v \in V_T^*\ [\ \varphi(a,v,\langle v \rangle_a) = 1\ ]$,
- $\forall A \in V_N,\ \forall v \in V_T^*,\ \forall v_f \in V_T^* \cup \{\mathtt{f}\}\ [\ (R(A),v,v_f) \in \rho \Rightarrow \varphi(A,v,v_f) = 1 + \varphi(R(A),v,v_f)\ ]$,
- $\forall e_1, e_2 \in PE(V_T \cup V_N),\ \forall v, v_1 \in V_T^*,\ \forall v_f \in V_T^* \cup \{\mathtt{f}\}$ *s.t.* 次の 5 行が全て成り立つ :
  - $(e_1,v,v_f) \in \rho \Rightarrow \varphi((e_1],v,\langle v, v_f \rangle) = 1 + \varphi(e_1,v,v_f)$.
  - $(e_1,v,v_1),\ (e_2,v_1,v_f) \in \rho \Rightarrow \varphi((e_1e_2),v,v_f) = 1 + \varphi(e_1,v,v_1) + \varphi(e_2,v_1,v_f)$.
  - $(e_1,v,\mathtt{f}) \in \rho \Rightarrow \varphi((e_1e_2),v,\mathtt{f}) = 1 + \varphi(e_1,v,\mathtt{f})$.
  - $(e_1,v,v_1) \in \rho \Rightarrow \varphi((e_1/e_2),v,v_1) = 1 + \varphi(e_1,v,v_1)$.
  - $(e_1,v,\mathtt{f}),\ (e_2,v,v_f) \in \rho \Rightarrow \varphi((e_1/e_2),v,v_f) = 1 + \varphi(e_1,v,\mathtt{f}) + \varphi(e_2,v,v_f)$.

ただし, 簡単のため $\rho = \rho(V_N,V_T,R)$ と置いた.

定義 2.5 $G = (V_N,V_T,R,e_S) \in PEG$ と $v \in V_T^*$ を任意に取る.
$(e_S,v) \xrightarrow[V_N,V_T,R]{} \mathtt{t}$ が成り立つとき, $(G,v) \to \mathtt{t}$ と書き, 「v を G に適用すると成功する」と呼ぶ.
$(e_S,v) \xrightarrow[V_N,V_T,R]{} \mathtt{f}$ が成り立つとき, $(G,v) \to \mathtt{f}$ と書き, 「v を G に適用すると失敗する」と呼ぶ.
$(e_S,v) \xrightarrow[V_N,V_T,R]{} \infty$ が成り立つとき, $(G,v) \to \infty$ と書き, 「v を G に適用すると無限ループに陥る」と呼ぶ.
$(G,v) \to \mathtt{t}$ のとき, $(e_S,v) \xrightarrow[V_N,V_T,R]{} \mathtt{t}$ が成り立つのだから, 定義 2.4 で既に見たように, $(e_S,v,v') \in \rho(V_N,V_T,R)$ を満たす $v' \in V_T^*$ がただ 1 つ存在し, しかも v' は v のサフィックスである. よって, $v = v''v'$ という形に表せる. このときの v'' のことを, 「v を G に適用して成功したときに消費される文字列」と呼ぶ.

定義 2.6 (PEL の定義) $G = (V_N,V_T,R,e_S) \in PEG$ に対して,

$$L(G) := \{v \in V_T^* \mid (G,v) \to \mathtt{t}\} \subset V_T^*$$

と定義する. さらに, $PEL \subset \mathfrak{P}(V_1^*)$ を以下のように定義する:

$$L \in PEL \ \Leftrightarrow_{def} \ \exists G = (V_N, V_T, R, e_S) \in PEG \ [\ L = L(G)\ ].$$

PEL の各元のことを parsing expression language と呼ぶ.

例 $a, b \in V_1,\ A \in V_2$ が成り立っているとして, $V_T := \{a, b\}$, $V_N := \{A\}$ と置く. また, $R : V_N \to PE(V_T \cup V_N)$ を

$$R(A) := (((aA)a)/b)$$

と定義する. さらに, $e_S := (A((a](b]))$ と置く. このとき, $G := (V_N, V_T, R, e_S)$ と置けば, $G \in PEG$ が成り立つ. また,

$$L(G) = \{a^n b a^n \mid n \geq 0\}$$

が成り立つ (この文書の定義に沿って証明すると大変だが). よって,

$$\{a^n b a^n \mid n \geq 0\} \in PEL$$

が成り立つ.

定義 2.7 (gTS の定義) $G = (V_N, V_T, R, e_S) \in PEG$ は $e_S \in V_N$ を満たすとする (従って, ここでの V_N は空でない). さらに, G は次を満たすとする.

$\forall A \in V_N,\ \exists a \in V_T,\ \exists B, C, D \in V_N \ \ s.t.$ 次の 4 つのうちいずれかが成り立つ.

- $R(A) = a$,
- $R(A) = ((a]/((a]])$,
- $R(A) = ((a]a)$,
- $R(A) = ((BC)/((B]D))$.

このとき, G は gTS であると呼ぶ. gTS 全体の集合を gTS と置く. 次に, $B \subset V_1$ は空でない有限集合とする.

$$gTS(B) := \{(V_N, V_T, R, e_S) \in gTS \mid V_T = B\}$$

と定義する.

補足 gTS の定義において, 4 つある • のうち 2 つ目の $R(A) = ((a]/((a]])$ については,

$$R(A) = true$$

のつもりである. 今回は $true$ を定義しないので, $true$ に相当する parsing expression が個別に必要であり, そのような parsing expression として $((a]/((a]])$ を採用した, ということである. 同じく, $R(A) = ((a]a)$ については,

$$R(A) = false$$

のつもりである.

定義 2.8 (MACRO PEG の定義) MACRO PEG [3] は MACRO GRAMMAR の PEG バージョンのようなものであり，PEG の非終端記号に引数がつけられるものが MACRO PEG である．引数に渡せるのは parsing expression のみである. ただし, ここでの parsing expression は，定義 2.2 で定義したものではなく[1]，MACRO PEG 用に拡張して定義された parsing expression である. 引数に規則そのものを渡すことを許容しない MACRO PEG が first order MACRO PEG という雰囲気であり, よって

$$\text{first order MACRO PEL} \subset \text{MACRO PEL}$$

が成り立つ. なお, 引数の評価戦略によって, 同じ MACRO PEG のコードでも違う言語を生成する可能性がある． この文書では, [3] に合わせて, 引数の評価戦略には call-by-name を用いる. また，引数の書き方は $A(x_1, x_2, \cdots, x_n)$ ではなく, $A[x_1][x_2]\cdots[x_n]$ という書き方を用いる． この書き方だと,

$$A[x_1][x_2]\cdots[x_i] \quad (i < n)$$

という表現もまた意味を持っているかのような雰囲気が漂ってくるが, この文書で使う MACRO PEG では, このような表現は使わない. すなわち，A が n 個の引数を取る非終端記号として定義されているなら，常に $A[x_1][x_2]\cdots[x_n]$ という書き方だけを用いることにする[2]． なお, A が非終端記号のとき, このままでは, A がいくつの引数を取る非終端記号なのか区別がつかない. そこで, この文書の中では，A のかわりに (A, n) という書き方をすることがある． ただし, A が取る引数の個数を n とした.
MACRO PEG をきちんと定義しようとすると面倒くさいので, この程度の言及に留めておく．なお, この文書では, first order の MACRO PEG があれば十分である.

例 $\{a, b\}^*$ における回文 (正確には, 文字列長が偶数である回文) を MACRO PEG で表現することができる [3]． 今回の文書での表記法に合わせて [3] のコードを変換すると, 次のように書ける:

V_T := {a,b}　V_N := { (P, 1), (true, 0), (. , 0) }　e_S := (P[true] (.])

```
.  ← (a / b)
true ← ((a] / ((a]])
P[r] ← (((a P[(a r)]) / (b P[(b r)])) / r)
```

このとき, $G := (V_N, V_T, R, e_S)$ と置けば, $G \in$ MACRO PEG であり,

$$L(G) = \left\{ v\ {}^t v \mid v \in \{a, b\}^* \right\}$$

となる. なお, V_N の元が (A, n) という形になっているが, 前述したとおり, これは

(非終端記号, その記号が取る引数の個数)

という意味のつもりである. よって, $(A, n) \in V_N$ ならば, A が取る引数の個数は n 個であり,

$$A[x_1][x_2]\cdots[x_n]$$

という書き方をして使うことになる. 上の例では, P が 1 変数であり，true と . (ドット) が 0 変数である.

[1] なぜなら, そちらは非終端記号に引数がつけられないから.
[2] 私が個人的に製作した MACRO PEG では, $A[x_1][x_2]\cdots[x_i]$ $(i < n)$ という書き方も認識するようになっている. しかし, 今回の文書では，この機能を使う機会は無い.

3　PEG と gTS の性質

この節では, PEG と gTS の特徴的な性質を列挙する.

定義 3.1

$$gTS_0 := \{G \in gTS \mid \forall v \in V_T^* \ [\ (G, v) \to \mathtt{t} \text{ または } (G, v) \to \mathtt{f}\ ]\ \}$$

と定義する. 次に, $B \subset V_1$ は空でない有限集合とする.

$$gTS_0(B) := \{(V_N, V_T, R, e_S) \in gTS_0 \mid V_T = B\}$$

と定義する. 要するに, 無限ループが決して起きない gTS だけを集めた集合が gTS_0, $gTS_0(B)$ である.

定理 3.2 $B \subset V_1$ は空でない有限集合とする. このとき, 次が成り立つ:

$$\forall G \in PEG(B),\ \exists G_1 \in gTS(B)\ [\ L(G) = L(G_1)\ ].$$

証明　[2] に G から G_1 への変換手順が書いてある.

定理 3.3 $B \subset V_1$ は空でない有限集合とする. このとき, 次が成り立つ:

$$\forall G_1 \in gTS(B),\ \exists G_2 \in gTS_0(B)\ [\ L(G_1) = L(G_2)\ ].$$

証明　[1] に証明が書いてある. 大変に長い証明である. 実は, [1] の証明での本質と思われる部分だけを抜き出すことで, [1] よりも直感的かつ直接的に定理 3.3 を証明することができるのだが, それはそれで結局は大量の準備が必要であり, 長くなるので, 省略する.

定理 3.4 $L \in PEL$ に対して,

$$B_L := \{a \in V_1 \mid \exists v \in L\ [\ \text{文字列 } v \text{ の中に } a \text{ が出現する}\ ]\ \}$$

と置く. このとき, 次が成り立つ:

(1) $\forall L \in PEL,\ \forall B \subset V_1\ [\ L \subset B^* \Rightarrow B_L \subset B\ ]$.
(2) $\forall L \in PEL\ [\ B_L$は有限集合, $L \subset B_L^*\ ]$.
(3) $B_L = \emptyset \Leftrightarrow L = \emptyset, \{\varepsilon\}$.

証明　(1) B_L の定義から簡単に示せる.
(2) $L \in PEL$ を任意に取る. $L = L(G)$ を満たす $G = (V_N, V_T, R, e_S) \in PEG$ が少なくとも 1 つは取れる. $L(G) \subset V_T^*$ であるから, (1) より $B_L \subset V_T$ である. V_T は有限集合だから, B_L も有限集合である. また, B_L の定義から明らかに $L \subset B_L^*$ である.
(3) $L = \emptyset$ または $L = \{\varepsilon\}$ のときは, 明らかに $B_L = \emptyset$ である. 逆に, $B_L = \emptyset$ とする. $L = \emptyset, \{\varepsilon\}$ を示したい. $L \subset V_1^*$ だから, $L \cap V_1^+ = \emptyset$ が示せれば十分である. もし $L \cap V_1^+ \neq \emptyset$ ならば, B_L の定義から明らかに $B_L \neq \emptyset$ となって矛盾する. よって, (3) が成り立つ.

例　$a,b,c \in V_1$ が成り立つものとする．$L := \{a^n b a^n \mid n \geq 0\}$ と置くと, 既に見たように $L \in PEL$ であり, かつ $B_L = \{a,b\}$ である．$L := \{c^n \mid n \geq 0\}$ と置くと, $L \in PEL$ であることが証明できて, かつ $B_L = \{c\}$ である.

定理 3.5　$L \in PEL - \{\emptyset, \{\varepsilon\}\}$ を任意に取る. 定理 3.4 の B_L について，ある $= (B_L, V_N, R, e_S) \in PEG$ が存在して, $L = L(G)$ が成り立つ.

証明　定理 3.4 の (3) より, $B_L \neq \emptyset$ である. そこで, $a \in B_L$ を 1 つ固定する．$L \in PEL$ の定義により, $L = L(G)$ を満たす $G = (V_N, V_T, R, e_S) \in PEG$ が少なくとも 1 つは取れる．定理 3.4 の (1) より, $B_L \subset V_T$ である. 特に

$$L = \{v \in B_L^* \mid (G,v) \to \mathtt{t}\} \tag{1}$$

が成り立つ. なぜなら, $v \in$ (左辺) とすると, $v \in L = L(G)$ だから，$(G,v) \to \mathtt{t}$ である. また, $v \in L \subset B_L^*$ である. よって, $v \in$ (右辺) となる．逆に, $v \in$ (右辺) とすると, $v \in B_L^* \subset V_T^*$ かつ $(G,v) \to \mathtt{t}$ だから，$v \in L(G)$ であり, よって $v \in L$ となり, $v \in$ (左辺) となる. よって，(1) が成り立つ. さて，$F : PE(V_T \cup V_N) \to PE(B_L \cup V_N)$ を次のように定義する: $e \in PE(V_T \cup V_N)$ を任意に取る. e の中に出現する終端記号のうち，$V_T - B_L$ の元を全て検出し, それらの終端記号を全て $((a]a)$ に置換する. 置換後の文字列を e' と書くとき，$e' \in PE(B_L \cup V_N)$ が成り立つことが証明できる. そこで，$F(e) := e'$ と定義する. このとき, 次が成り立つことが証明できる:

- $\forall b \in B_L \cup V_N \ [\ F(b) = b\ ]$.
- $\forall b \in V_T - V_L \ [\ F(b) = ((a]a)\ ]$.
- $\forall e_1, e_2 \in PE(V_T \cup V_N)$
  $[\ F((e_1 e_2)) = (F(e_1)F(e_2)),\ \ F((e_1]) = (F(e_1)],\ \ F((e_1/e_2)) = (F(e_1)/F(e_2))\ ]$.

次に, $V_N' := V_N$ と置く. また, $R' : V_N' \to PE(B_L \cup V_N')$ を

$$R'(A) := F(R(A)) \ \ (A \in V_N')$$

と定義する. このとき, 次が成り立つことが証明できる[3]:

$$\begin{aligned} &\forall (e, v, v_f) \in PE(V_T \cup V_N) \times B_L^* \times (B_L^* \cup \{\mathtt{f}\}) \\ &\quad [\ (e,v,v_f) \in \rho(V_N, V_T, R) \Leftrightarrow (F(e), v, v_f) \in \rho(V_N', B_L, R')\ ]. \end{aligned} \tag{2}$$

ここで, $e_S' := F(e_S) \in PE(B_L \cup V_N')$ と置き, $G' := (V_N', B_L, R', e_S')$ と置く．このとき, $G' \in PEG$

[3] $(\Rightarrow)$ は $\varphi(e,v,v_f)$ に関する数学的帰納法で証明できて，$(\Leftarrow)$ は $\varphi(F(e),v,v_f)$ に関する数学的帰納法で証明できる.

である. また,

$$\begin{aligned}
L &= \{v \in B_L^* \mid (G, v) \to \mathtt{t}\} \\
&= \left\{ v \in B_L^* \mid (e_S, v) \xrightarrow[V_N, V_T, R]{} \mathtt{t} \right\} \\
&= \{v \in B_L^* \mid \exists v' \in V_T^* \ [\ (e_S, v, v') \in \rho(V_N, V_T, R)\ ]\ \} \\
&= \{v \in B_L^* \mid \exists v' \in B_L^* \ [\ (e_S, v, v') \in \rho(V_N, V_T, R)\ ]\ \} \\
&= \{v \in B_L^* \mid \exists v' \in B_L^* \ [\ (F(e_S), v, v') \in \rho(V'_N, B_L, R')\ ]\ \} \\
&= \{v \in B_L^* \mid \exists v' \in B_L^* \ [\ (e'_S, v, v') \in \rho(V'_N, B_L, R')\ ]\ \} \\
&= \left\{ v \in B_L^* \mid (e'_S, v) \xrightarrow[V'_N, B_L, R']{} \mathtt{t} \right\} \\
&= \{v \in B_L^* \mid (G', v) \to \mathtt{t}\} \\
&= L(G')
\end{aligned}$$

となる. ただし, 2 つ目の等号では (1) を使った. また, 4 つ目の等号では, v' が v のサフィックスであることと $v \in B_L^*$ を使った. また, 5 つ目の等号では (2) を使った.

系 3.6 $L \in PEL$ を任意に取る.
(1) $L \subset B^*$ を満たす, 空でない有限集合 $B \subset V_1$ が必ず取れる.
(2) (1) を満たす任意の B に対して, ある $G = (V_N, B, R, e_S) \in PEG$ が存在して, $L = L(G)$ が成り立つ.

証明 (1) $L = \emptyset$ または $L = \{\varepsilon\}$ のときは, $B \subset V_1$ として 1 元集合を 1 つ取れば, 明らかに $L \subset B^*$ である. $L \in PEL - \{\emptyset, \{\varepsilon\}\}$ のときは, $B = B_L$ と置けば, 定理 3.4 より, B は空でない有限集合であり, しかも $L \subset B^*$ である.
(2) $L \subset B^*$ を満たす, 空でない有限集合 $B \subset V_1$ を任意に取る. $L = \emptyset$ のときは, $a \in B$ を 1 つ取って $V_N := \emptyset$, $R := \emptyset$, $e_S := ((a]a)$ と置けば, $G := (V_N, B, R, e_S)$ が求める G である. 次に, $L = \{\varepsilon\}$ のときは, B の元を全て列挙して $B = \{a_i \mid 1 \le i \le n\}$ と表し, $_N := \emptyset$, $R := \emptyset$, $e_S := (((\cdots((a_1](a_2])\cdots(a_n])$ と置けば[4], $G := (V_N, B, R, e_S)$ が求める G である.
以下では, $L \in PEL - \{\emptyset, \{\varepsilon\}\}$ としてよい. $L \subset B^*$ だったから, $B_L \subset B$ である. そこで, 定理 3.5 の $G = (V_N, B_L, R, e_S) \in PEG$ をまず取ってくる (よって, $L = L(G)$ である). $V'_N := V_N$ と置くと, $B_L \subset B$ に注意して

$$PE(B_L \cup V_N) \subset PE(B \cup V_N) = PE(B \cup V'_N)$$

である. よって, $e'_S := e_S$ と置けば, $e'_S \in PE(B \cup V'_N)$ である. さらに, $R' : V'_N \to PE(B \cup V'_N)$ を

$$R'(A) := R(A) \quad (A \in V'_N)$$

として定義可能である. このとき, $G' := (V'_N, B, R', e'_S)$ と置けば, $G' \in PEG$ であり, しかも $L(G) = L(G')$ である. $L = L(G)$ だったから, $L = L(G')$ となる. よって, この G' が求める G' である.

[4] この e_S はカッコが多すぎて意味が分かりにくいが, 要するに $e_S := (a_1](a_2]\cdots(a_n]$ と定義しているだけである.

4　PEG で表せない言語

この節では, PEG で表せない言語を具体的に 1 つ作る.

定義 4.1 $C := \{\ 1\ ,\ :\ \}$ と置く. $I \subset C^*$ を以下のように定義する:

$$v \in I \ \Leftrightarrow_{def}\ \exists n \geq 1,\ \exists a,b,c,d,e \in \mathbb{N}^n \ \ s.t.\ \text{次が全て成り立つ :}$$

- $v = v_n v_{n-1} \cdots v_1 \ \ (v_i = 1^{a_i} : 1^{b_i} : 1^{c_i} : 1^{d_i} : 1^{e_i} :\ (1 \leq i \leq n))$,
- $\forall i \in [1,n]\ [\ a_i, c_i, d_i, e_i \in [1,n]\ ]$ (b_iが抜けているが, それでよい),
- $\forall i \in [1,n]\ [\ a_i = i\ ]$,
- $\forall i \in [1,n]\ [\ b_i \in [1,5]\ ]$,
- $\forall i \in [1,n]\ [\ b_i \in [1,4] \Rightarrow c_i = d_i = e_i = 1\ ]$.

例　$v = 1:1:1:1:1:$ と置くと, $v \in I$ である. $v = 11:1:1:1:1:1:1111:1:1:1:$ と置くと, $v \in I$ である. $v = 1:1:1:1:1:1:11:1:1:1:1:$ と置くと, $v \notin I$ である.

定義 4.2 V_2 は可算無限集合だったから, 適当に番号づけて $V_2 = \{A_i \mid i \in \mathbb{N}\}$ と表記しておく. 定義 4.1 の C と I に対して, 写像 $\Psi : I \to gTS(C)$ を以下のように定義する: まず, $v \in I$ を任意に取る. v の定義を満たす $n \geq 1$ を取る. この n に対して, $V_T := C$, $V_N := \{A_i \mid 1 \leq i \leq n\}$, $e_S := A_n$ と置く. さらに, $R : V_N \to PE(V_T \cup V_N)$ を以下のように定義する: $1 \leq i \leq n$ を任意に取る. $R(A_i)$ を定義したい.

$$v_i = 1^{a_i} : 1^{b_i} : 1^{c_i} : 1^{d_i} : 1^{e_i} :$$

であったが, もし $b_i = 1$ ならば, $R(A_i) :=\ 1\ $ と定義する. もし $b_i = 2$ ならば, $R(A_i) :=\ :\ $ と定義する. もし $b_i = 3$ ならば, $R(A_i) := ((1]/((1]])$ と定義する. もし $b_i = 4$ ならば, $R(A_i) := ((1]1)$ と定義する. もし $b_i = 5$ ならば, $R(A_i) := ((A_{c_i} A_{d_i})/((A_{c_i}]A_{e_i}))$ と定義する. こうして R を定義すると, $(V_N, V_T, R, e_S) \in gTS(C)$ が成り立つ. そこで, $\Psi(v) := (V_N, V_T, R, e_S)$ と定義する.

例　次の文字列を考える. 見易さのため 7 行に区切っているが, 実際には 1 行の長い文字列である.

1111111 : 11111 : 111111 : 11111 : 1111 :
111111 : 11111 : 1 : 1111 : 111 :
11111 : 11111 : 11 : 1111 : 111 :
1111 : 1111 : 1 : 1 : 1 :
111 : 111 : 1 : 1 : 1 :
11 : 11 : 1 : 1 : 1 :
1 : 1 : 1 : 1 : 1 :

この文字列を v と置くと, $v \in I$ が成り立つ. $\Psi(v) = (V_N, V_T, R, e_S)$ と表すと, 次のようになる:

$$
\begin{aligned}
&V_T = C,\ \ V_N = \{A_i \mid 1 \leq i \leq 7\},\ \ e_S = A_7,\\
&R(A_7) = ((A_6A_5)/((A_6]A_4))\\
&R(A_6) = ((A_1A_4)/((A_1]A_3))\\
&R(A_5) = ((A_2A_4)/((A_2]A_3))\\
&R(A_4) = ((1]1)\\
&R(A_3) = ((1]/((1]])\\
&R(A_2) = \ :\\
&R(A_1) = \ 1
\end{aligned}
$$

なお, この $\Psi(v) = (V_N, V_T, R, e_S) \in gTS$ が生成する言語は $\{\varepsilon\}$ である.
もう 1 つ例を挙げておく. 次の文字列を考える.

11111111111 : 11111 : 11111111 : 11111 : 1111 :

1111111111 : 11111 : 11111111 : 1 : 1111 :

111111111 : 11111 : 1 : 1111111111 : 1111 :

11111111 : 11111 : 111111111 : 111 : 11 :

1111111 : 11111 : 11 : 1111 : 111 :

111111 : 11111 : 1 : 1111 : 111 :

11111 : 11111 : 111111 : 1111111 : 1111 :

1111 : 1111 : 1 : 1 : 1 :

111 : 111 : 1 : 1 : 1 :

11 : 11 : 1 : 1 : 1 :

1 : 1 : 1 : 1 : 1 :

この文字列を v と置くと, $v \in I$ が成り立つ. $\Psi(v) = (V_N, V_T, R, e_S)$ と表すと, 次のようになる:

$$
\begin{aligned}
&V_T = C,\ \ V_N = \{A_i \mid 1 \leq i \leq 11\},\ \ e_S = A_{11},\\
&R(A_{11}) = ((A_8A_5)/((A_8]A_4))\\
&R(A_{10}) = ((A_8A_1)/((A_8]A_4))\\
&R(A_9) = ((A_1A_{10})/((A_1]A_4))\\
&R(A_8) = ((A_9A_3)/((A_9]A_2))\\
&R(A_7) = ((A_2A_4)/((A_2]A_3))\\
&R(A_6) = ((A_1A_4)/((A_1]A_3))\\
&R(A_5) = ((A_6A_7)/((A_6]A_4))\\
&R(A_4) = ((1]1)\\
&R(A_3) = ((1]/((1]])\\
&R(A_2) = \ :\\
&R(A_1) = \ 1
\end{aligned}
$$

なお, この $\Psi(v) = (V_N, V_T, R, e_S) \in gTS$ が生成する言語は $\{\ 1^n : 1^n \mid n \geq 0\}$ である.

定理 4.3 定義 4.1 の C に対して, 次が成り立つ:

$$\forall G \in gTS_0(C),\ \ \exists v \in I\ \ [\ \Psi(v) \in gTS_0(C),\ \ L(G) = L(\Psi(v))\ ]\,.$$

証明 $G = (V_N, C, R, e_S) \in gTS_0(C)$ を任意に取る. G の定義に出現する全ての非終端記号[5]の名前を $A_1, \cdots, A_n$ に置換したものを $G' = (V'_N, C, R', e'_S)$ と置く. ただし, $e'_S = A_n$ となるようにする (これは明らかに可能である). このとき, $G' \in gTS_0(C)$ かつ $L(G) = L(G')$ が成り立つ. また, $\Psi(v) = G'$ が成り立つように $v \in I$ を構成することが明らかに可能である. よって, この v が求める v である.

定理 4.4 定義 4.1 の C に対して, $L \subset C^*$ を以下のように定義する:

$$L := \{v \in I \mid (\Psi(v), v) \to \mathtt{f}\}\,.$$

このとき, $L \notin PEL$ が成り立つ.

証明 明らかに $L \subset \{\ 1\ ,\ :\ \}^* = C^*$ である $\cdots$(*) さて, $L \notin PEL$ が成り立つことを背理法 (よくある対角線論法) で示す. $L \in PEL$ とする. (*) により, 系 3.6 が使えて, ある $G = (C, V_N, R, e_S) \in PEG(C)$ が存在して $L = L(G)$ が成り立つ. 定理 3.2,3.3 より, ある $G' = (C, V'_N, R', e'_S) \in gTS_0(C)$ が存在して $L(G) = L(G')$ が成り立つ. さらに, 定理 4.3 より, ある $v \in I$ が存在して, $\Psi(v) \in gTS_0(C)$ かつ $L(G') = L(\Psi(v))$ が成り立つ. 特に $L = L(\Psi(v))$ が成り立つ. さて, $\Psi(v) \in gTS_0(C)$ であるから, $(\Psi(v), v) \to \mathtt{t}$ もしくは $(\Psi(v), v) \to \mathtt{f}$ のいずれかが成り立つ. 前者の場合は, $L(\Psi(v))$ の定義により, $v \in L(\Psi(v))$ となる. また, L の定義により $v \notin L$ となる. $L = L(\Psi(v))$ だったから, $v \in L$ かつ $v \notin L$ となって矛盾する. 後者の場合は, $L(\Psi(v))$ の定義により, $v \notin L(\Psi(v))$ となる. また, L の定義により $v \in L$ となる. $L = L(\Psi(v))$ だったから, $v \in L$ かつ $v \notin L$ となって矛盾する. 以上より, いずれの場合も矛盾する. よって, $L \notin PEL$ が成り立つ.

5 MPEG で表せる言語

この節では, 前節の言語 L を first order の MACRO PEG で表現する
(ここから PEL $\subsetneq$ first order MACRO PEL が示せる).

準備 5.1 まず, gTS の動作を MACRO PEG でエミュレートさせることを考える. きちんとやるとフォーマットのチェックから出発しなければならず面倒くさいので, ここでは本質的な部分だけを抜き出して, やや手抜きしつつ書くことにする. なお, ここで使う parsing expression は, 1 節で定義した形のものだとカッコが多くて冗長なので, カッコは適宜省略することにする. また, 1 節で既に触れたように, 引数の書き方は $A(x_1, x_2, \cdots, x_n)$ ではなく, $A[x_1][x_2]\cdots[x_n]$ という書き方を用いる. さらに, ここから先は, $A[x_1][x_2]\cdots[x_n] \leftarrow e$ という書き方のかわりに, $A[x_1][x_2]\cdots[x_n] = e;$ という書き方を用いることにする.
最初の雛形は, 次のページにある MACRO PEG である.

[5] V_N には当然ながら非終端記号が出現するが, それ以外にも, $(A, e) \in R$ における $e \in PE(V_T \cup V_N)$ にも非終端記号が出現することがある. それらも全て置換対象となる.

```
.  = 1 / :  ;
true = (1] / ((1]];

skip1[F][p] = ((p 1]] skip1[F][ p.  ]  / ((p :]]  F[ p.  ];
skip2[F][p][a] = ((p 1]] skip2[F][ p.  ][a] / ((p :]]  F[ p.  ][a];
skipord [F][p][a][b][c][d] = ((p 1]] skipord [F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord1[F][ p.  ][a][b][c][d];
skipord1[F][p][a][b][c][d] = ((p 1]] skipord1[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord2[F][ p.  ][a][b][c][d];
skipord2[F][p][a][b][c][d] = ((p 1]] skipord2[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord3[F][ p.  ][a][b][c][d];
skipord3[F][p][a][b][c][d] = ((p 1]] skipord3[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord4[F][ p.  ][a][b][c][d];
skipord4[F][p][a][b][c][d] = ((p 1]] skipord4[F][ p.  ][a][b][c][d] /
                             ((p :]]  F[ p.  ][a][b][c][d];

gTS[p][it] = skip2[gTS1][p][it];
gTS1[p][it] = ((p 1:]]  it 1 / ((p 11:]]  it :  / ((p 111:]]  it /
              ((p 11111:]]  Vx[ p ......  ][it][true];
Vx[p][it][x] = ((p x 1]] Vx[p][it][x 1] /
               ((p x :]]  Vy[p][it][x :][true];
Vy[p][it][x][y] = ((p x y 1]] Vy[p][it][x][y 1] /
                  ((p x y :]]  Vz[p][it][x][y :][true];
Vz[p][it][x][y][z] = ((p x y z 1]] Vz[p][it][x][y][z 1] /
                     ((p x y z :]]  Wx[true][it][x][y][z :];
Wx[p][it][ x][ y][z] = ((p x]] Wy[true][it][ p][y][z] /
                       (p x] skipord[Wx][p][it][ x][y][z];
Wy[p][it][px][ y][z] = ((p y]] Wz[true][it][px][p][z] /
                       (p y] skipord[Wy][p][it][px][y][z];
Wz[p][it][px][py][z] = ((p z]] X [it][px][py][p] /
                       (p z] skipord[Wz][p][it][px][py][z];
X[it][px][py][pz] = (( gTS[px][it] ]] Y[true][it][px][py] /
                    ( gTS[px][it] ] gTS[pz][it];
Y[r][it][px][py] = (( gTS[px][it] r.  ]]  Y[ r.  ][it][px][py] /
                   ( gTS[px][it] r.  ]  Z[r][true][py];
Z[r][m][py] = (( r m.  ]]  Z[r][ m.  ][py] / ( r m.  ]  gTS[py][m];

S = gTS[true][true];
```

上記のコードについて, 開始表現 e_S は S に設定するものとする ($e_S := S$).
さて, S に食わせる入力文字列は

$$vw \quad (v \in I,\ w \in C^*)$$

という形のものに限定する. 「雛形」と書いたように, vw をそのまま S に食わせても正常に動作しない. 上の実装では, v と w を取るたびに, S の部分を手動で

```
S = gTS[true][.....(ちょうど|v|個のドットが並ぶ).....];
```

というものに変更しなければならない (実装上の単なる手抜きである).
たとえば, $v = 1:1:1:1:1:$, $w = 1$ としたときの vw を S に食わせるときには,

```
S = gTS[true][.....];
```

としてから食わせなければならない. このとき, S は v を読み取って $\Psi(v) \in gTS$ をエミュレートし, w の部分を $\Psi(v)$ に食わせたときの動作を示す. 今の場合, $v = 1:1:1:1:1:$ だったから, $\Psi(v)$ は「先頭の 1 文字が 1 なら, それを消費して成功」という gTS になっている. $w = 1$ だったから, 結局, vw を S に食わせると「成功する」ことになる. 今度は $w =:$ としてみると, $\Psi(v)$ は同じ gTS であるが, w の先頭の 1 文字が 1 ではないので, vw を S に食わせると「失敗する」ことになる.

次に, `gTS[p][it]` の具体的な仕組みを説明する. まずは引数の役割を説明する. `gTS` の引数にある `p` と `it` には, ドットを 0 個以上並べたものしか代入しない[6]. `p` は, ざっくり言えば

> $\Psi(v)$ において次に適用すべき関数記号 (正確には非終端記号と言うべきだが) の,
> v 側におけるエントリーポイント (1)

を指すように用意されたポインタのような引数である. 従って, `p` が指す場所として許されるのは, $v = v_n v_{n-1} \cdots v_1$ ($v_i = 1^{a_i} : 1^{b_i} : 1^{c_i} : 1^{d_i} : 1^{e_i} :$) と表したときの

$$v_n\text{の先頭の 1 文字},\quad v_{n-1}\text{の先頭の 1 文字},\quad \cdots,\quad v_1\text{の先頭の 1 文字}$$

の n 箇所のみである. また, S における `p` の初期値は常に「v_n の先頭の 1 文字」である. すなわち, `p = true` である. $v \in I$ と $\Psi(v) \in gTS$ の意味を考えると, $v_n = 1^{a_n} : 1^{b_n} : 1^{c_n} : 1^{d_n} : 1^{e_n} :$ は $\Psi(v)$ における開始表現を意味するので, S における `p` の初期値が `p = true` であるのは当たり前の話である. 一方で, `it` は文字列 vw 内の何らかの文字を指すポインタのような引数である. `it = true` ならば, `it` は文字列 vw の先頭の 1 文字を指していると解釈し, `it = .` ならば, `it` は文字列 vw の 2 文字を指していると解釈し, $\cdots$ といった具合である. なお, S における `it` の初期値は,

```
it = (vw における w の先頭の 1 文字)
```

とするのが普通である. これを実現するには,

```
it = ...(ドットが|v|個)...
```

とすればよい. ただし, これ以外の初期値も可能である. たとえば

```
it = (vw における w の先頭の 1 文字のさらに 1 文字先) = ...(ドットが (|v|+1) 個)...
```

[6]ただし, ドットが 0 個のときは `true` が代入されるものとする.

とすると, vw を S に食わせたら, w^{+1} を $\Psi(v)$ に適用したときの動作を示すようになる．もしくは,

`it = (vw` における `w` の先頭の `1` 文字の `1` 文字手前`) = ...(`ドットが `(|v|-1)` 個`)...`

とすると, vw を S に食わせたら, $v^{+(|v|-1)}w$ を $\Psi(v)$ に適用したときの動作を示すようになる．すなわち, v の末尾の 1 文字を c とするとき, cw を $\Psi(v)$ に適用したときの動作を示すようになる.

次に, `gTS[p][it]` が成功したときの, vw において消費される文字列について説明する．ただし, ここでの `p` は (1) の用途で使われているものとし，v 内のある関数のエントリーポイントを正常に指しているものとする．さらに, `it` は文字列 vw 内のどこかを指しているものとする. その指している地点で vw を分割して $vw = x_1x_2$ と書く. 従って, `it` は vw における x_2 の先頭の 1 文字を指していることになる．さて, `p` が指すエントリーポイントに対応する, $\Psi(v)$ 側の関数記号を A としておく．このとき, `gTS[p][it]` がもし成功したならば, vw において消費される文字列は

`it` が指す文字より手前の文字列全体 (すなわち x_1)

及び

A に x_2を適用したときに消費される文字列 (s とする)

である. このとき, $x_2 = sx_2'$ と表せるので, 結局, vw において消費される文字列は x_1s であり，残った文字列は x_2' となる.
以下で具体例を挙げる. 既に見た $v = 1:1:1:1:1:$, $w = 1$ の場合は, もし

```
S = gTS[true][..........];
```

ならば, vw を S に適用すれば vw 全てが消費される. $w = 11:1$ などとすると，vw において消費される文字列は $v1$ であり，残った文字列は $1:1$ となる. $w =:$ のときは, そもそも vw は S において成功しないので, 消費量の話にはならない．次に, 再び $v = 1:1:1:1:1:$, $w = 1$ に戻って, S を

```
S = gTS[true][...........];
```
(ドットがさっきより `1` つ多い)

としてみる. このとき, vw を S に適用すれば, $\Psi(v)$ に空文字を適用することになるので, 結果は「失敗する」ことになる．$w = 11:1$ とすると, $\Psi(v)$ に $1:1$ を適用することになるので，結果は成功であり, vw において消費される文字列は $v11$ であり，残った文字列は $:1$ となる. もう 1 つ例を挙げる. $v = 1:11:1:1:1:$, $w = 1$ として,

```
S = gTS[true][..........];
```

とすると, $\Psi(v)$ は「先頭の 1 文字が $:$ なら, それを消費して成功」という gTS になり，しかも $\Psi(v)$ に食わせる文字列は $:w$ となる (v の末尾の 1 文字から始まっている)．よって, vw を S に食わせると成功し, 消費される文字列は v であり, 残った文字列は w となる.

さて, ここから先は, `gTS[p][it]` の内部的な動作を見ていきたいのだが，その前に, `gTS[p][it]` が満たすべき仕様をきちんと書き下しておく．上で具体例を交えて `gTS[p][it]` の仕様をふわっと解説したばかりだが，もう少し丁寧に説明する.

`gTS[p][it]` が満たすべき仕様: まず, $v \in I$ と $w \in C^*$ を任意に取る．`p` は vw 内における `v` のある

関数記号のエントリーポイントを指しているとする．it は vw のある文字を指しているとする. その場所で vw を分割して $vw = x_1x_2$ と置く． p が指すエントリーポイントに対応する, $\Psi(v)$ 側の関数記号を A としておく． このとき,

(1) vw を gTS[p][it] に適用すると成功する $\Leftrightarrow$ x_2を A に適用すると成功する,

(2) vw を gTS[p][it] に適用すると失敗する $\Leftrightarrow$ x_2を A に適用すると失敗する

が成り立つ. よって,

(3) vw を gTS[p][it] に適用すると無限ループする $\Leftrightarrow$ x_2を A に適用すると無限ループする

も成り立つ. また, (1) が成り立つとき, vw が gTS[p][it] において消費される文字列を z として, x_2 が A において消費される文字列を s とするとき, $z = x_1s$ が成り立つ $\cdots$(4)

今回実装した gTS[p][it] は, 上記の仕様を実際に満たしている. このことを確認する.
まず, gTS[p][it] の内部的な動作を見ていく． 前節では $V_2 = \{A_i \mid i \in \mathbb{N}\}$ という表現を使っていたが，ここでもこの表現を使うことにする. さっきと同じく，ここでの p は (1) の用途で使われているものとし，v 内のある関数のエントリーポイントを正常に指しているものとする． 従って, ある $i \in [1, n]$ に対して, p は v_i の先頭の 1 文字を指している． これに対応する, $\Psi(v)$ 側の関数記号は A_i である. さて,

```
gTS [p][it] = skip2[gTS1][p][it];
```

によって, p は $v_i = 1^{a_i} : 1^{b_i} : 1^{c_i} : 1^{d_i} : 1^{e_i} :$ のうちの $1^{a_i} :$ の部分をすっ飛ばし, $1^{b_i} : 1^{c_i} : 1^{d_i} : 1^{e_i} :$ における先頭の 1 文字を指すようになる[7]． 以下では, b_i の値に応じて処理が分岐する. これを記述しているのが

```
gTS1[p][it] = ((p 1:]]  it 1 / ((p 11:]]  it :  / ((p 111:]]  it /
                  ((p 11111:]]  Vx[ p ......  ][it][true];
```

の部分である.
$b_i = 1$ のときは, 最初の ((p 1:]] it 1 に引っかかるので，ここが実行される. $b_i = 1$ は「先頭の 1 文字が 1 なら, それを消費しつつ成功」という意味だから，((p 1:]] it 1 によって, きちんと消費される. なお, この部分は ((p 1:]] 1 ではなく ((p 1:]] it 1 であることに注意する. it のところも消費しなければ，既に説明した gTS[p][it] の動作にならないからである.
$b_i = 2$ のときは, ((p 11:]] it : に引っかかり, ここが実行される． $b_i = 2$ は「先頭の 1 文字が:なら, それを消費しつつ成功」という意味だから，((p 11:]] it : によって, きちんと消費される.
$b_i = 3$ のときは, ((p 111:]] it に引っかかるので，ここが実行される. $b_i = 3$ は「無条件で成功し, 文字列の消費は無い」という意味だから，((p 111:]] it というコードになる. なお, この部分は ((p 111:]] ではなく ((p 111:]] it であることに注意する. it のところを消費しなければ，既に説明した gTS[p][it] の動作にならないからである.
$b_i = 4$ のときは「無条件で失敗する」という意味なので, MACRO PEG 側でも失敗するべきである． 当然ながら, 文字列 vw は MACRO PEG 側で 1 文字も消費されないようにしたい. よっ

[7] p が指している文字列が $1^s : 1^t : 1^u : \cdots$ というものであるとき, skip2 を通過することで, p は $1^t : 1^u : \cdots$ を指すようになる. すなわち, p が指す文字列の先頭部分にある一塊の「1^* :」をスキップさせる処理が skip2 である.

て，$b_i = 4$ のケースは単に無視すればよく, MACRO PEG 側で捕捉する必要はない.
あとは, $b_i = 5$ のときが残っている. このときは, $\Psi(v)$ 側での処理が

$$A_i \leftarrow ((A_{c_i} A_{d_i})/((A_{c_i}]A_{e_i}))$$

であるため, この処理をすることになる. よって, まずは関数番号 c_i, d_i, e_i を取得しなければならない． これを行うためのスタート地点が `((p 11111:]] Vx[ p ..... ][it][true];` である. 以下では, `Vx`,`Vy`,`Vz` によって, c_i, d_i, e_i の値が引数 `x`,`y`,`z` に取得される． 具体的には, `x` は 1^{c_i} : となり， `y` は 1^{d_i} : となり， `z` は 1^{e_i} : となる. 次に, この `x`,`y`,`z` を手がかりにして， $A_{c_i}, A_{d_i}, A_{e_i}$ に対応する v 内のエントリーポイントを探し出す． これを行っているのが `Wx`,`Wy`,`Wz` である. その結果， 引数 `px`,`py`,`pz` に該当のエントリーポイントが格納される． `px` には A_{c_i} のエントリーポイントが格納され， `py` には A_{d_i} のエントリーポイントが格納され， `pz` には A_{e_i} のエントリーポイントが格納される． このあとは, いよいよ

$$((A_{c_i} A_{d_i})/((A_{c_i}]A_{e_i}))$$

を実行しなければならない. これを行っているのが

```
X[it][px][py][pz] = (( gTS[px][it] ]] Y[true][it][px][py] /
                    ( gTS[px][it] ] gTS[pz][it];
Y[r][it][px][py] = (( gTS[px][it] r. ]] Y[ r. ][it][px][py] /
                   ( gTS[px][it] r. ] Z[r][true][py];
Z[r][m][py] = (( r m. ]] Z[r][ m. ][py] / ( r m. ] gTS[py][m];
```

の部分である. 以下では, 簡単のため, 文字列 vw 内において `it` が指している場所で vw を分割し， $vw = x_1 x_2$ と書くことにする. 従って, `it` は vw における x_2 の先頭の 1 文字を指していることになる． まず, A_{c_i} が成功するかどうかをチェックする. そのためには, 単に `gTS[px][it]` を実行すればよい． ただし, 成功するかどうかの結果だけが欲しいので, 仮に成功しても, vw 内での文字列の消費は無いようにしたい． そこで, 実際には `(( gTS[px][it] ]]` を実行することになる. もしこれが「失敗」したならば, 実行すべきは A_{e_i} であるから， `gTS[pz][it]` を実行すればよい. このケースでは, `gTS[pz][it]` が成功した暁にはきちんと vw 内で文字列が消費されるべきであるから， `(( gTS[pz][it] ]]` とはせず, 普通に `gTS[pz][it]` とする. これを行っているのが `( gTS[px][it] ] gTS[pz][it];` の部分である． あとは, A_{c_i} の結果が成功だった場合が残っている. このときは,

文字列 x_2を A_{c_i}に食わせて成功している

ので, そこで消費された x_2 内の文字列を s としておく. よって, $x_2 = s x_2'$ と表せる． さて, 次に実行すべきは A_{d_i} なのだが, ここで 1 つ問題が発生する． まず, A_{c_i} については結果だけを確認しただけであるから， A_{c_i} に関する文字列の消費はまだ行っておらず, 文字列 vw は vw のままである． また, A_{d_i} に食わせる文字列は x_2' でなければならない． よって, `it` の位置を, $vw = x_1 x_2 = x_1 s x_2'$ 内における x_2' の先頭の 1 文字まで移動しなければならない． 以上を踏まえて, まず x_2' の位置を算出することを目指す. これを行っているのが `Y`,`Z` である． まず, `Y` において, $|x_2'|$ の文字列長が引数 `r` に格納される． より具体的には, 次のようになる: まず, `(( gTS[px][it] r. ]]` の部分において， `gTS[px][it]` によって文字列 vw は $x_1 s$ のところまでを消費し, x_2' だけが残る． これに `r.` を適用するので, 「`r` に格納されているドットの個数+1」の文字数だけ x_2' が削られる． ここまでが成功する限りは, `r` の値は 1 ドットずつ増えていき, 再帰的に `Y` が繰り返される. この作業は常に

`(( ]]` の中で行われるので, 成功するたびに vw の消費はリセットされ, vw は常に vw のままであり, `r` の値だけが増えていくことになる. そして, これが失敗した段階で, そのときの `r` は $|x_2'|$ 個のドットで構成されている. よって, `Y` を通過した段階で, 引数 `r` には $|x_2'|$ の文字列長が格納されることになる. もちろん, 文字列 vw は vw のままである. 次は `Z` の出番である. ここでは, `r` の値を手がかりにして, $|x_1 s|$ の文字列長を引数 `m` に格納している. 格納が終わったら, そのときの `m` は

$$vw = x_1 x_2 = x_1 s x_2' \text{内における } x_2' \text{の先頭の 1 文字}$$

を指していることになる. さらに, 文字列 vw は vw のままである. よって, この `m` を新しい `it` だと思って `gTS[py][m]` を実行すれば, 意図した動作が実現できる. この場合, `gTS[py][m]` が成功した暁にはきちんと vw 内で文字列が消費されるべきであるから, `(( gTS[py][m] ]]` とはせず, 普通に `gTS[py][m]` とする.

以上で `gTS[p][it]` の説明を終える. 上記のように定義した `gTS[p][it]` は, (1),(2),(3),(4) を全て満たす. これらのことは厳密には証明すべきであるが, `gTS[p][it]` の実装の仕方から直感的には明らかである. まじめに証明するなら, 証明すべきは (1),(2),(4) のみである ((3) は (1),(2) から従う). これらは 1 つずつ証明するのではなく, (1) ∧ (2) ∧ (4) を同時に証明する方がやりやすい.
(⇒) については, `gTS[p][it]` の内部において再帰的に呼び出された `gTS` の合計回数を $n \geq 0$ とするとき, n に関する数学的帰納法で (1) ∧ (2) ∧ (4) の (⇒) バージョンが証明できる.
(⇐) については, 1 節で定義した φ を使って, $\varphi(A, x_2, v_f)$ に関する数学的帰納法で (1) ∧ (2) ∧ (4) の (⇐) バージョンが証明できる.

準備 5.2 準備 5.1 では, gTS[p][it] の実装のみに焦点を当てており, 入力文字列である vw のうち, $v \in I$ に関するフォーマットのチェックを行っていなかった. そこで, $v \in I$ のフォーマットをチェックする MACRO PEG を実装する. これは次のページのようにして実装できる.

```
.  = 1 / :  ;
true = (1] / ((1]];

skip1[F][p] = ((p 1]] skip1[F][ p.  ]  / ((p :]]  F[ p.  ];
skip2[F][p][a] = ((p 1]] skip2[F][ p.  ][a] / ((p :]]  F[ p.  ][a];
skipord [F][p][a][b][c][d] = ((p 1]] skipord [F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord1[F][ p.  ][a][b][c][d];
skipord1[F][p][a][b][c][d] = ((p 1]] skipord1[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord2[F][ p.  ][a][b][c][d];
skipord2[F][p][a][b][c][d] = ((p 1]] skipord2[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord3[F][ p.  ][a][b][c][d];
skipord3[F][p][a][b][c][d] = ((p 1]] skipord3[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord4[F][ p.  ][a][b][c][d];
skipord4[F][p][a][b][c][d] = ((p 1]] skipord4[F][ p.  ][a][b][c][d] /
                             ((p :]]  F[ p.  ][a][b][c][d];

one = 1 (one / true);
chk1 = (( chk10 (chk1 / (.]  )  ]];
chk10 = one :1:1:1:1:  / one :11:1:1:1:  / one :111:1:1:1:  /
        one :1111:1:1:1:  / one :11111:  one :  one :  one :  ;

geq[p][q] = ((p 1]] ((q 1]] geq[p .][q .]  / ( ((p 1]] ((q 1]] ] (q 1];
chk2 = chk20[true];
chk20[q] = ((q.]]  chk21[q] / (q.];
chk21[q] = geq[true][q] skip1[chk22][q];
chk22[q] = skip1[chk23][q];
chk23[q] = geq[true][q] skip1[chk24][q];
chk24[q] = geq[true][q] skip1[chk25][q];
chk25[q] = geq[true][q] skip1[chk20][q];

A[p][q][t][td] = ((p t :  1:1:1:]]  chk30[ p td .  ......  ][ .q ];
chk3 = chk30[true][true];
chk30[p][q] = (q :]  geq[p][q] geq[q][p] skip2[chk31][p][q] / ((q :]]  ( p.  ];
chk31[p][q] = A[p][q][1][.]  / A[p][q][11][..]  / A[p][q][111][...]  /
              A[p][q][1111][....]  / ((p 11111:]]  chk32[ p ......  ][q];
chk32[p][q] = skip2[chk33][p][q];
chk33[p][q] = skip2[chk34][p][q];
chk34[p][q] = skip2[chk35][p][q];
chk35[p][q] = chk30[p][ q.  ];

S = chk1 chk2 chk3;
```

上記のコードについて, 開始表現 e_S は S に設定するものとする ($e_S := S$).
今回は $v \in I$ に関するチェックだから, S に食わせる文字列は vw ではなく, v のみである. 上記の S に任意の文字列 v を食わせれば, S が成功するとき, かつそのときのみ $v \in I$ が成り立つ. すなわち, 上記のコードで $v \in I$ のフォーマットがチェックできる. このことを確認する.
前ページのとおり, チェック項目は chk1,chk2,chk3 の 3 つに分かれている. まず, chk1 では, $v \in I$ が満たすべき最低限のフォーマットをチェックしている. 具体的にどういうチェックをしているのかは, ここで説明するよりも chk1 の実装を眺めた方が早いと思うので, chk1 の説明は省略する. 次のチェック項目は chk2 であるが, その前にまず, geq[p][q] について説明しなければならない.
geq[p][q] では, p と q は 0 個以上のドットを並べた値になっており, 文字列 v 内の何らかの文字を指していると解釈する. p が指しているところから出発する文字列のうち, 連続して並んでいる 1 の個数を n_p と置く. たとえば, p が指している文字列が 11111 : 11 : というものだったら, $n_p = 5$ である. また, 111 : 1111111 : 1111 : というものだったら, $n_p = 3$ である. また, : 111 : 11 : のようになっている場合は, $n_p = 0$ である. 同様にして n_q を定義する. このとき, geq[p][q] が成功するのは

$$n_p \geq n_q$$

が成り立つとき, かつそのときのみである. また, geq[p][q] が成功しようが失敗しようが, 文字列 v は 1 文字も消費しない. さて, ここからは chk2 の説明に移る. まず, chk1 でのチェックにより, v は

$$v = v_n v_{n-1} \cdots v_1 \quad (v_i = 1^{a_i} : 1^{b_i} : 1^{c_i} : 1^{d_i} : 1^{e_i} : \ (1 \leq i \leq n)),$$
$$\forall i \in [1, n] \ [\ a_i \geq 1,\ b_i \geq 1,\ c_i \geq 1,\ d_i \geq 1,\ e_i \geq 1\ ]$$

という形をしていることが分かる. 今回の chk2 の項目では, 上記の a, b, c, d, e について,

$$\forall i \in [1, n] \ [\ a_n \geq a_i,\ a_n \geq c_i,\ a_n \geq d_i,\ a_n \geq e_i\ ]$$

が成り立っているかどうかをチェックしている (b はチェックしない).
$a_n \geq a_i$ のチェックは chk21 で行っている.
b_i をスルーする処理は chk22 で行っている.
$a_n \geq c_i$ のチェックは chk23 で行っている.
$a_n \geq d_i$ のチェックは chk24 で行っている.
$a_n \geq e_i$ のチェックは chk25 で行っている.

最後に, chk3 について説明する. 自分で作っておいてなんだが, この chk3 は大変に説明しづらいアルゴリズムになっている. この項目では,

$$(1)\ \ \forall i \in [1, n] \ [\ a_i = i\ ],$$
$$(2)\ \ \forall i \in [1, n] \ [\ b_i \in [1, 5]\ ],$$
$$(3)\ \ \forall i \in [1, n] \ [\ b_i \in [1, 4] \Rightarrow c_i = d_i = e_i = 1\ ]$$

が成り立っているかどうかをチェックしている ((1) のアルゴリズムが説明しづらい).
では, 具体的にアルゴリズムを見ていく. chk3 = chk30[true][true]; であるから, chk30 から出発する. chk30[p][q] の引数 p,q は 0 個以上のドットを並べた値になっており, 文字列 v 内の何らかの文字を指していると解釈する. p と q の初期値は「v の先頭の 1 文字」と解釈できる.

よって, 既に定義した n_p, n_q について, 初期状態では $n_p = n_q = a_n$ が成り立っている. chk30 の内部を潜って進んでいくと, いずれは再帰的に新しい chk30 が呼び出されるわけだが, そのたびに q は 1 文字だけ先に進む. また, q の値に変化があるのはそのときだけである. よって, chk30 を 1 回呼び出すごとに, n_q の値は

$$a_n \to a_n - 1 \to a_n - 2 \to \cdots$$

と 1 ずつ減少するように変化していく. よって, i 回目の chk30 を呼び出した[8] 直後の q に関する n_q を $n_q(i)$ と書くことにすると, 基本的には

$$n_q(i) = a_n - (i - 1)$$

が成り立つことになる. 一方で, 新しい chk30 が呼び出されるときには, p は一塊の v_i をスキップした状態になる. 従って, chk30 を 1 回呼び出すごとに, p は

$$v_n\text{の先頭の 1 文字} \to v_{n-1}\text{の先頭の 1 文字} \to v_{n-2}\text{の先頭の 1 文字} \to \cdots$$

という場所を指すように変化する. よって, i 回目の chk30 を呼び出した直後の p に関する n_p を $n_p(i)$ と書くことにすると, (4) により,

$$n_p(i) = a_{n-(i-1)} \tag{5}$$

が成り立つことになる. さて,

```
chk30[p][q] = (q :]  geq[p][q] geq[q][p] skip2[B][p][q] / ((q :]]  ( p.  ];
```

であるから, `(q :]  geq[p][q] geq[q][p] skip2[B][p][q]` が成功する限りは, この再帰的な作業が続く. それ以外の場合は, `(q :]` に失敗するか, もしくは `(q :]` に成功しつつ `geq[p][q] geq[q][p] skip2[B][p][q]` に失敗するかのいずれかである[9]. 後者の場合は, /の右にある `((q :]]  ( p.  ];` が適用されるが, `(q :]` は成功していたのだから, `((q :]]  ( p.  ];` には失敗し, よって `chk30[p][q]` に失敗する. 前者の場合は, やはり / の右にある `((q :]]  ( p.  ];` が適用される. `(q :]` は失敗していたのだから, `((q :]]` の部分は成功し, 残るは `( p.  ];` のみである. この部分まで成功して初めて, `chk30[p][q]` は成功する. さて, `(q :]` に失敗するのはどういうときかと言えば, q が

$$v = 1^{a_n} : 1^{b_n} : 1^{c_n} : 1^{d_n} : 1^{e_n} : 1^{a_{n-1}} : 1^{b_{n-1}} : 1^{c_{n-1}} : 1^{d_{n-1}} : 1^{e_{n-1}} : \cdots : 1^{a_1} : 1^{b_1} : 1^{c_1} : 1^{d_1} : 1^{e_1} :$$

$$\uparrow$$

この ” : ”

を指しているときである. また, `( p.  ];` が成功するのはどういうときかと言えば, p が v 全体を通過してヌル文字を指しているような状態のときである. よって, `chk30[true][true]` が成功するためには, 次が満たされなければならない:

(6) q が 1^{a_n} : のところにある : を指すまでは, chk30 の再帰がずっと続かなければならない.

(7) q がその : を指した段階で, p はギリギリ v 全体を通過していなければならない.

[8] 初回の呼び出しを「1 回目」とカウントする. よって, i は 1 から始まる (0 ではなく).

[9] chk30 の構造上, 無限ループは起きない.

よって, chk30[true][true] が成功した暁には, (6) により, chk3 での一連の処理における n_q の値が

$$a_n \to a_n - 1 \to a_n - 2 \to \cdots \to 2 \to 1 \tag{8}$$

まで推移することになり, その次の $n_q = 0$ の時点で (q :] に失敗することになる. また, (7) と (5) により, n_p の値は

$$a_n \to a_{n-1} \to a_{n-2} \to \cdots \to a_2 \to a_1 \tag{9}$$

まで推移することになる (その次は $n_q = 0$ のタイミングなので, (q :] に失敗し, geq が行われないので, もはや n_p は参照されない). さて,

(q :] geq[p][q] geq[q][p] skip2[B][p][q]

における geq の部分では,

$$n_p == n_q$$

という条件式をチェックしている. これを $n_p(i)$, $n_q(i)$ の表記で書き直せば

$$n_p(i) == n_q(i)$$

をチェックしていることになる. $n_q = 0$ のときは geq が行われないので, このチェックは $n_q = 1$ のときまで続くことになる. (8) における, $n_q = 1$ のタイミングでの値は, (8) の右端にある「1」である. また, (9) における, $n_q = 1$ のタイミングでの値は, (9) の右端にある「a_1」である. よって, (8),(9) を右から左に向かって比較すれば,

$$a_1 == 1, \quad a_2 == 2, \quad a_3 == 3, \quad \cdots$$

が全て *true* になる. よって, (1) が成り立つ. (2),(3) については, chk31,$\cdots$,chk35 でチェックしている. $b_i = 1, 2, 3, 4$ のときのチェックは chk31 で済んでしまう. $b_i = 5$ のときは, c_i, d_i, e_i に関して何もチェックしなくてよいので, c_i, d_i, e_i の項はスキップさせる. このスキップを順次行っているのが chk33,chk34,chk35 である. 以上で, chk3 の説明を終わる.

以上で chk1,chk2,chk3 の説明を終わる. これらのチェックにより, $v \in I$ のフォーマットが完全にチェックできることが分かる.

定理 5.3 前節の言語 L は first order の MACRO PEG で表現できる.

証明 準備 5.1,5.2 を組み合わせればよい. 具体的には, 次からの 2 ページの MACRO PEG で L が表現できる. なお, skip1,skip2,skipord では引数に規則を渡しているが, これらは可読性のための処置にすぎない. すなわち, skip1,skip2,skipord の実際の適用場面ごとに, これらは「規則を渡さない形に展開可能」である. よって, 実質的には first order の MACRO PEG で記述できていることになる. また, その他の引数の使い方を眺めてみると,

true, ドット, 1, :

の 4 種類のものを, 単に連接しながら引数に渡すことしかしていない.

```
.  = 1 / :  ;
true = (1] / ((1]];

skip1[F][p] = ((p 1]] skip1[F][ p.  ]  / ((p :]]  F[ p.  ];
skip2[F][p][a] = ((p 1]] skip2[F][ p.  ][a] / ((p :]]  F[ p.  ][a];
skipord [F][p][a][b][c][d] = ((p 1]] skipord [F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord1[F][ p.  ][a][b][c][d];
skipord1[F][p][a][b][c][d] = ((p 1]] skipord1[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord2[F][ p.  ][a][b][c][d];
skipord2[F][p][a][b][c][d] = ((p 1]] skipord2[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord3[F][ p.  ][a][b][c][d];
skipord3[F][p][a][b][c][d] = ((p 1]] skipord3[F][ p.  ][a][b][c][d] /
                             ((p :]]  skipord4[F][ p.  ][a][b][c][d];
skipord4[F][p][a][b][c][d] = ((p 1]] skipord4[F][ p.  ][a][b][c][d] /
                             ((p :]]  F[ p.  ][a][b][c][d];

one = 1 (one / true);
chk1 = (( chk10 (chk1 / (.]  )  ]];
chk10 = one :1:1:1:1:  / one :11:1:1:1:  / one :111:1:1:1:  /
        one :1111:1:1:1:  / one :11111:  one :  one :  one :  ;

geq[p][q] = ((p 1]] ((q 1]] geq[p .][q .]  / ( ((p 1]] ((q 1]] ] (q 1];
chk2 = chk20[true];
chk20[q] = ((q.]]  chk21[q] / (q.];
chk21[q] = geq[true][q] skip1[chk22][q];
chk22[q] = skip1[chk23][q];
chk23[q] = geq[true][q] skip1[chk24][q];
chk24[q] = geq[true][q] skip1[chk25][q];
chk25[q] = geq[true][q] skip1[chk20][q];

A[p][q][t][td] = ((p t :  1:1:1:]]  chk30[ p td .  ......  ][ .q ];
chk3 = chk30[true][true];
chk30[p][q] = (q :]  geq[p][q] geq[q][p] skip2[chk31][p][q] / ((q :]]  ( p.  ];
chk31[p][q] = A[p][q][1][.]  / A[p][q][11][..]  / A[p][q][111][...]  /
              A[p][q][1111][....]  / ((p 11111:]]  chk32[ p ......  ][q];
chk32[p][q] = skip2[chk33][p][q];
chk33[p][q] = skip2[chk34][p][q];
chk34[p][q] = skip2[chk35][p][q];
chk35[p][q] = chk30[p][ q.  ];
```

```
gTS[p][it] = skip2[gTS1][p][it];
gTS1[p][it] = ((p 1:]]  it 1 / ((p 11:]]  it :  / ((p 111:]]  it /
              ((p 11111:]]  Vx[ p ......  ][it][true];
Vx[p][it][x] = ((p x 1]] Vx[p][it][x 1] /
               ((p x :]]  Vy[p][it][x :][true];
Vy[p][it][x][y] = ((p x y 1]] Vy[p][it][x][y 1] /
                  ((p x y :]]  Vz[p][it][x][y :][true];
Vz[p][it][x][y][z] = ((p x y z 1]] Vz[p][it][x][y][z 1] /
                     ((p x y z :]]  Wx[true][it][x][y][z :];
Wx[p][it][ x][ y][z] = ((p x]] Wy[true][it][ p][y][z] /
                       (p x] skipord[Wx][p][it][ x][y][z];
Wy[p][it][px][ y][z] = ((p y]] Wz[true][it][px][p][z] /
                       (p y] skipord[Wy][p][it][px][y][z];
Wz[p][it][px][py][z] = ((p z]] X [it][px][py][p] /
                       (p z] skipord[Wz][p][it][px][py][z];
X[it][px][py][pz] = (( gTS[px][it] ]] Y[true][it][px][py] /
                    ( gTS[px][it] ] gTS[pz][it];
Y[r][it][px][py] = (( gTS[px][it] r.  ]]  Y[ r.  ][it][px][py] /
                   ( gTS[px][it] r.  ]  Z[r][true][py];
Z[r][m][py] = (( r m.  ]]  Z[r][ m.  ][py] / ( r m.  ]  gTS[py][m];

consume = .  consume / true;

S = chk1 chk2 chk3 ( gTS[true][true] ] consume;
```

上記のコードについて, 開始表現 e_S は S に設定するものとする ($e_S := S$).
さて, 上記の MACRO PEG で実際に L が表現できていることを確認する. 上記の MACRO PEG を G と置く. $L = L(G)$ を示したい. まず, $L \subset L(G)$ を示す. $v \in L$ を任意に取る. L の定義により,

$$(1)\ \ v \in I,$$
$$(2)\ \ (\Psi(v), v) \to \mathtt{f}$$

が成り立っている. よって, v を

```
S = chk1 chk2 chk3 ( gTS[true][true] ] consume;
```

に適用すると, (1) により, `chk1 chk2 chk3` のところは成功し, 文字列 v は 1 文字も消費しない. よって, 文字列 v は v のままで `gTS[true][true]` に到達する. ここでは, 既に説明した `gTS` の動作により,

$$\Psi(v) \text{ に } v \text{ を適用したときの動作}$$

が実現される. (2) により, `gTS[true][true]` の結果は失敗となる. よって, `( gTS[true][true] ]` 全体では成功となる. あとは `consume;` が残っているが, この部分は単に入力文字列を全て消

費するだけであり, 必ず成功する. よって, v を S に適用すると成功する. よって, $v \in L(G)$ となる. よって, $L \subset L(G)$ となる.
次に, $L(G) \subset L$ を示す. $v \in L(G)$ を任意に取る. このとき, $L(G)$ の定義により, v を S に適用すると成功する. 特に `chk1 chk2 chk3` のところが成功するので, v は (1) を満たすことになる. また, この部分では文字列 v の消費は無いので, 文字列 v は v のままで `( gTS[true][true] ]` に到達する. ここも成功するのだから, `gTS[true][true]` だけを見ると「失敗する」ことになる. 既に説明した `gTS` の動作により, v を `gTS[true][true]` に適用すると

$\Psi(v)$ に v を適用したときの動作

が実現されるので, それが失敗するということは, (2) が成り立つということである. よって, v は (1),(2) を満たすので, L の定義により, $v \in L$ が成り立つ. 以上より, $L(G) \subset L$ である. よって, $L = L(G)$ である. よって, L は first order MACRO PEG で表現できる.

参考文献

[1] Birman, A.: The TMG Recognition Schema, Ph.D. Thesis, Princeton University, 1970.

[2] 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)

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