BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2017
http://excel.fit.vutbr.cz

How to Define any Recursively Enumerable Language as the Input Language of a Translation Grammar

Radek Vít*

Abstract
This paper discusses the descriptional complexity of translation grammars. It shows a way to define any recursively enumerable language by translation grammars. Queue grammars characterize the family of recursively enumerable languages (see page 20 of [1]). This paper shows a way to simulate any queue grammar by a translation grammar. Any recursively enumerable language can then be defined by translation grammars by simulating queue grammars. This means that any recursively enumerable language can be accepted by pushdown automata with two stacks.

Keywords: Formal Lanuguages — Translation Grammars — Recursively Enumerable Languages

Supplementary Material: *N/A*

*xvitra00@fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Translation grammars are traditionally used to define translations (see pages 49 through 57 of [2] and [3]). Their input and output languages define, on their own, two context-free languages. By adding constraints to their output language, however, translation grammars can be used to define languages which are not context-free. This paper aims to prove that translation grammars can be used to define all recursively enumerable languages.

There are several formal systems that define recursively enumerable languages. Extended post correspondences, queue grammars and scattered context grammars all define the family of recursively enumerable languages (see pages 20 through 21 and page 42 of [1]). By fixing a translation grammar's output to a member of Dyck's language, it can be constructed so that it simulates any queue grammar. This paper shows a way to define any recursively enumerable language by translation grammars by simulating queue grammars.

2. Definitions

This section provides some definitions for the reader to understand the following sections. It is assumed that the reader is already familiar with the language theory. The following definitions are quoted essentially verbatim from [1].

Definition 2.1. A *queue grammar* is a sextuple

$$Q = (V, T, W, F, R, g)$$

where

- V is an alphabet of nonterminals and terminals
- $T \subset V$ is an alphabet of terminals
- W is an alphabet of states. $V \cap W = \varnothing$
- $F \subset W$ is a set of final states
- $R \subseteq (V \times (W - F)) \times V^* \times W)$ is a finite relation
- $g \in (V - T)(W - F)$ is the starting pair of symbols

If $u = arb$, $v = rxc$, and $(a, b, x, c) \in R$, where $r, x \in V^*$, $a \in V$, and $b, c \in W$, then Q makes a derivation step from u to v according to (a, b, x, c).
The language generated by a queue grammar Q, denoted by $L(Q)$, is defined as

$$L(Q) = \{x : x \in T^*, g \Rightarrow_Q^* xf, f \in F\}$$

Queue grammars characterize the family of recursively enumerable languages (see page 20 of [1]).

Definition 2.2. A *translation grammar* is a quintuple

$$G = (N, T_I, T_O, P, S)$$

where

- N is an alphabet of nonterminals;
- T_I is an input alphabet such that $T_I \cap N = \varnothing$
- T_O is an input alphabet such that $T_O \cap N = \varnothing$
- P is a finite set of productions in the form

  $$(A, A) \Rightarrow (u_0 B_1 u_1 \dots B_n u_n, v_0 B_1 v_1 \dots B_n v_n)$$

  for $j = 1, \dots, n$, $B_j \in N$, for $i = 0, \dots, n$, $u_i \in T_I^*$ and $v_i \in T_O^*$ ($n = 0$ implies $x = u_0$ and $y = v_0$)
- $S \in N$ is the start symbol.

The translation defined by G is denoted by $T(G)$ and defined as

$$T(G) = \{(i, o) : i \in T_I^*, o \in T_O^*, (S, S) \Rightarrow_G^* (i, o)\}$$

The input language is defined as

$$L_I(G) = \{i : (i, o) \in T(G)\}$$

The output language is defined as

$$L_O(G) = \{o : (i, o) \in T(G)\}$$

Definition 2.3. A *linear translation grammar* is a translation grammar as described in Definition 2.2 with productions in P in the forms

$$(A, A) \Rightarrow (u, x)$$

and

$$(A, A) \Rightarrow (uBv, xBy)$$

where $A, B \in N$, $u, v \in T_I^*$ and $x, y \in T_O^*$

Definition 2.4. Throughout this paper, D denotes Dyck's language defined as

$$D = \{vw : v \in \{0,1\}^*, w = reversal(v)\}$$

3. Generating Recursively Enumerable Languages

This section demonstrates that translation grammars can be used to define any recursively enumerable language by simulating queue grammars.

Lemma 3.1. Recall Lemma 2.38. in [1]. Let Q' be a queue grammar. Then, there exists a queue grammar

$$Q = (V, T, W' \cup \{\$, f\}, \{f\}, R, g)$$

such that $L(Q') = L(Q)$, where $W' \cap \{\$, f\} = \varnothing$, each $(a, b, x, c) \in R$ satisfies $a \in V - T$ and

- either $b \in W'$, $x \in (V - T)^*$, $c \in W' \cup \{\$, f\}$
- or $b = \$$, $x \in T$ and $c \in \{\$, f\}$

The symbol $\$ \notin W'$ denotes the state where only terminals are generated and the symbol $f \notin W'$ is the new and only final state.

Lemma 3.2. For every queue grammar Q, there exists a linear translation grammar G such that

$$L(Q) = \{x : (x, y) \in T(G), y \in D\}$$

Proof. Without any loss of generality, assume that the queue grammar Q satisfies the conditions described in Lemma 3.1.

$$Q = (V, T, W' \cup \{\$, f\}, \{f\}, R, g)$$

Then we construct a linear translation grammar G in the following way:

$$G = (W' \cup \{\$, f, S\}, T, \{0,1\}, P, S)$$

where $S \cup W' = \varnothing$. Set $n = |V|$.
Introduce the bijective homomorphism α from V to $\{0,1\}^n \cap 0^*10^*$
Expand its domain to V^* so that $\alpha(ab) = \alpha(a)\alpha(b)$, where $a \in V, b \in V^*$. Let ω be the bijective homomorphism defined as $\omega(x) = reversal(\alpha(x)), x \in V^*$
P is constructed as follows:

1. For $g = kl, k \in V, l \in W'$, add $(S, S) \Rightarrow (l, \alpha(k)l)$ to P.
2. For each $(a, b, x, c) \in R$, where $a \in V - T$, $b \in W' \cup \{\$\}$, $x \in (V - T)^* \cup T$ and $c \in W \cup \{f\}$, add $(b, b) \Rightarrow (c, \alpha(x)c\omega(a))$ to P.
3. For each $t \in T$, add $(f, f) \Rightarrow (tf, f\omega(t))$ to P.
4. Finally, add $(f, f) \Rightarrow (\varepsilon, \varepsilon)$ to P.

Basic idea. The constructed translation grammar G simulates the queue grammar Q that satisfies the properties described in Lemma 3.1. The production from 1, applied only once, initialises the derivation. The production 4 terminates the derivation. The productions from 2 simulate the rules applied by Q. Finally, productions from 3 generate the simulated terminals to the input string in the order generated by Q.

Claim A. G can generate every $(x, y) \in T(G)$ where $y \in D$ in this way:

$$\begin{array}{lrrr}
 & (S, & S &) \\
\Rightarrow & (l, & \alpha(k)l &) \\
\Rightarrow^* & (\$, & \alpha(a_0..a_k..a_{k+m})\$ & \omega(a_k..a_0)) \\
\Rightarrow^* & (f, & \alpha(a_0..a_{k+m}x_1..x_m)f & \omega(a_{k+m}..a_0)) \\
\Rightarrow^* & (xf, & \alpha(a_0..a_{k+m}x_1..x_m)f & \omega(x_m..x_1a_{k+m}..a_0)) \\
\Rightarrow & (x, & \alpha(a_0..a_{k+m}x_1..x_m) & \omega(x_m..x_1a_{k+m}..a_0))
\end{array}$$

Where $k,m \geq 1$, $a_i \in V-T$ for $i=0,\ldots,k+m$; $g = kl : k \in V-T, l \in W'$; $a_0 = k$; $x = x_1..x_m$, $x_i \in T^*$ for $i=1,\ldots,m$;

$$y = \alpha(a_0..a_{k+m}x_1..x_m)\omega(x_m..x_1a_{k+m}..a_0)$$

$$y = vw, w = reversal(v)$$

Proof of claim A. Examine the construction of G. Observe that every successful derivation begins with application of production 1, followed by applying productions from 2, followed by applying productions from 3. Every successful derivation ends with a single application of production 4.

Observe that during application of rules in 2 and 3, the output side is as follows: $w_i \in V$ for $i=0,\ldots,k+m$; $k,m \geq 1$; $N \in W' \cup \{\$, f\}$

$$\alpha(w_0..w_k..w_{k+m})N\omega(w_l..w_0)$$

To satisfy $y \in D$, only productions that put $\omega(w_{k+1})$ to the right of the nonterminal are applicable.

Claim B. Q generates every $h \in L(Q)$ in this way: $g = a_0q_0$

$$\begin{aligned}
& a_0q_0 && \\
\Rightarrow & a_1y_1q_1 && (a_0,q_0,z_0,q_1)\\
\Rightarrow & a_2y_2q_2 && (a_1,q_1,z_1,q_2)\\
& \ldots && \\
\Rightarrow & a_{k+1}y_{k+1}q_{k+1} && (a_k,q_k,z_k,q_{k+1})\\
\Rightarrow & a_{k+2}y_{k+2}x_1\$ && (a_{k+1},q_{k+1},x_1,\$)\\
& \ldots && \\
\Rightarrow & a_{k+m}x_1\ldots x_{m-1}\$ && (a_{k+m-1},\$,x_{m-1},\$)\\
\Rightarrow & x_1\ldots x_mf && (a_{k+m},\$,x_m,f)
\end{aligned}$$

where $k,m \geq 1$; $a_i \in V-T$ for $i=0,\ldots,k+m$; $y_i \in (V-T)^*$ for $i=1,\ldots,k+m-1$; $x_j \in T^*$ for $j=1,\ldots,m$; $z_0 = a_1y_1$; $y_iz_i = a_{i+1}y_{i+1}$ for $i=1,\ldots,k$; $g = a_0q_0$; $q_i \in W'$ for $i=0,\ldots,k$;

Proof of claim B. Recall that Q satisfies the properties given in Lemma 3.1. These properties imply that Claim B holds.

Claim C. Let G generate $(h,y) \in T(G)$, $y \in D$ in the way described in Claim A; then, $h \in L(Q)$.

Proof of claim C. Let $(h,y) \in T(G)$, $y \in D$. Consider the generation of h as described in Claim A. Examine the construction of P to see that at this point R contains $(a_0,q_0,z_0,q_1),\ldots,(a_k,q_k,z_k,q_{k+1}),(a_{k+1},q_{k+1},x_1,\$),\ldots,(a_{k+m},\$,x_m,f)$, where $z_0,\ldots,z_k \in (V-T)^*$, and $x_1,\ldots,x_m \in T^*$. Then, Q makes the generation of h in the way described in Claim B.

Claim D. Let Q generate $h \in L(Q)$ in the way described in Claim B; then, $(h,y) \in T(G)$, $y \in D$.

Proof of claim D. Let $h \in L(Q)$. Consider the generation of h as described in Claim B. Examine the construction of P to see that at this point P contains $(S,S) \Rightarrow (q_0,\alpha(a_0)q_0),(q_0,q_0) \Rightarrow (q_1,\alpha(z_0)q_1\omega(a_0)),\ldots,(q_k,q_k) \Rightarrow (q_{k+1},\alpha(z_k)q_{k+1}\omega(a_k)),(q_{k+1},q_{k+1}) \Rightarrow (\$,\alpha(x_1)\$\omega(a_k+1)),\ldots,(\$,\$) \Rightarrow (f,\alpha(x_m)f\omega(a_{k+m})),(f,f) \Rightarrow (x_1f,f\omega(x_1)),\ldots,(f,f) \Rightarrow (x_mf,f\omega(x_m))$, where $z_0,\ldots,z_k \in (V-T)^*$, and $x_1,\ldots,x_m \in T^*$. Then, G makes the generation of (h,y) in the way described in Claim A.

Claims A through D imply that $L(Q) = \{x : (x,y) \in T(G), y \in D\}$, so this lemma holds. □

4. Conclusion

Translation grammars can be used to simulate any queue grammar when constructed as described in this paper. Translation grammars constructed in this way can generate any recursively enumerable language when their output is a member od Dyck's language. This subset of the grammar's translations is easily obtainable from the set of all the grammar's translations.

To my knowledge, no attempts have been made to define a single non context-free language using a translation grammar. This paper shows the descriptional complexity of translation grammars when used for defining languages.

Because of translation grammars' computational complexity, other formal systems defining recursively enumerable languages can be simulated by translation grammars. Future research can introduce ways to simulate these systems by translation grammars. Parsers accepting recursively enumerable languages can be implemented using pushdown automata with two stacks based on translation grammars.

Acknowledgements

I would like to thank my supervisor Prof. RNDr. Alexander Meduna, CSc. for introducing this problem and his tremendous help.

References

[1] Alexander Meduna and Jiří Techet. *Scattered context grammars and their applications*. WIT, 2010.

[2] Alexander Meduna. *Automata and Languages: Theory and Applications*. Springer, 2005.

[3] Bořivoj Melichar. Formal translation directed by LR parsing. *Kybernetika*, 28(1):50–61, 1992.

[4] Alexander Meduna. *Formal Languages and Computation: Models and Their Applications*. Auerbach Publications, 2014.