Implementation of Haskell modules for automata and Sticker systems

K.K.K.R. Perera and Yoshihiro Mizoguchi

Received on March 11, 2009 / Revised on March 17, 2009

Abstract.
We realized operations appeared in the theory of automata using Haskell languages. Using the benefits of functions of lazy evaluations in Haskell, we can express a language set which contains infinite elements as concrete functional notations like mathematical notations. Our modules can be used not only for analyzing the properties about automata and their application systems but also for self study materials or a tutorial to learn automata, grammar and language theories. We also implemented the modules for sticker systems. Paun and Rozenberg explained a concrete method to transform an automaton to a sticker system in 1998. We modified their definitions and improved their insufficient results. Using our module functions, we can easily define finite automata and linear grammars and construct sticker systems which have the same power of finite automata and linear grammars.

Keywords. Automata, Language, Sticker System, DNA Computing, Haskell

1. Introduction

The sticker system is a formal model based on sticking operations, which is an abstraction of the Watson-Crick complementarity. We use the term domino to represent double stranded DNA sequences with sticky ends. By using the sticking operator, dominoes can be annealed and formed a complete double stranded sequence. Paun and Rozenberg [3] explained a concrete method to transform automata to sticker systems. In this paper we are trying to introduce simple efficient transformation and implement it using Haskell module functions. We also indicate and improve the insufficient results in [3]. We modify the expression of dominoes and the sticking operator for realizing Haskell functions. We change the definition of a domino \( D \) from a string of pairs of alphabet to a triple \( (l, r, x) \) of two string \( l \), \( r \) and an integer \( x \). For example \( \lambda \begin{bmatrix} C \\ T A \\ GC \end{bmatrix} \lambda \) in [3] is represented as \((\text{ATGC}, \text{CTA}, -1)\). According to this modification, the definition of sticking operator has been reformulated.

One of the benefits of using Haskell language is that it has descriptions for infinite set of strings using lazy evaluation schemes. For example, the infinite set \( \{a, b\}^\ast \) is denoted by finite length of expression \( \text{sstar} ['a', 'b'] \). We use the \text{take} function to view contents of an infinite set (e.g. \text{take 5 (sstar ['a', 'b'])} is \["a", "aa", "ab", "aa", "ba"\]). Further using set theoretical notions in Haskell, we can easily realize the definitions of various kinds of set of dominoes. For example, to make a sticker system which generates the equivalent language of a finite automaton, we need an atom set

\[ A_2 = \bigcup_{i=1}^{k+1} \{ (xu, x, 0) \mid x \in \Sigma^*, u \in \Sigma^*, |xu| = k + 2, |u| = i, \delta^*(0, xu) = i - 1 \}. \]

In Haskell notations, we have following function definitions.

\begin{align*}
aA2 :: \text{Automaton} \rightarrow \text{[Domino]} \\
aA2 m (q, s, d, q0, f) &= \text{concat} [(aA2' m i) | i \leftarrow [1..(k+1)]] \\
&\text{where } k = (\text{length } q)-1 \\
aA2' :: \text{Automaton} \rightarrow \text{Int} \rightarrow \text{[Domino]} \\
aA2' m (q, s, d, q0, f) i &= [(x++u, x, 0) \mid (x, u) \leftarrow \text{xupair}, (\text{dstar } d 0 (x++u)) == (i-1)] \\
&\text{where } \text{xupair} = \{(x, u) \mid x <\langle (\text{sigman } s (k+2-i)), u <\langle (\text{sigman } s i) \} \\
&k = (\text{length } q)-1
\end{align*}

The precise definition of the generated sticker system is described in Section 3. We prove that the generated languages are equal by using our formulations.

The Haskell module can be downloaded from our homepage\(^1\).

2. Automaton Module

Let \( \Sigma \) is an alphabet and \( \Sigma^\ast \) is the set of all strings over \( \Sigma \) including the empty string \( \lambda \). For a string \( w \), we denote the length of \( w \) by \( |w| \).

\(^1\)http://haskell.math.kyushu-u.ac.jp/
Definition 1. A finite automaton is a five-tuple of $M = (Q, \Sigma, \delta, q_0, F)$, where $Q$ is the finite set of states, $\Sigma$ is the alphabet, $q_0$ is the initial state, $F$ is the set of final states and $\delta : Q \times \Sigma \rightarrow Q$ is the transition function.

A transition function $\delta : Q \times \Sigma \rightarrow Q$ is generally extended to a function $\delta^* : Q \times \Sigma^* \rightarrow Q$ by $\delta^*(q, \lambda) = q$ and $\delta^*(q, ws) = \delta^*(\delta(q, w), s)$ for $q \in Q$, $x \in \Sigma$ and $w \in \Sigma^*$.

Definition 2. For a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$, we define the language $L(M)$ accepted by $M$ by $L(M) = \{w \in \Sigma^* | \delta^*(q_0, w) \in F\}$.

Example 1. Automata $M_1$ and $M_2$ is defined as follows.

$M_1 = ([0, 1], \{a, b\}, \delta_1, 0, \{1\})$, where $\delta_1(0, a) = 0$, $\delta_1(0, b) = 1$, $\delta_1(1, a) = 1$, $\delta_1(1, b) = 0$, $\delta_2(0, a) = 1$, $\delta_2(0, b) = 2$, $\delta_2(1, a) = 2$, $\delta_2(1, b) = 0$, $\delta_2(2, a) = 2$, $\delta_2(2, b) = 2$. Figure 1 is the transition diagram for $M_1$ and $M_2$. The examples are expressed as follows using our Haskell Modules.

\[
\begin{align*}
m1 &::= \text{Automaton} \\
m1 &= ([0, 1], \{a, b\}, d1, 0, \{1\}) \text{ where } d1 \cdot 'a' &= 0 \\
&\quad \text{where } d1 \cdot 'b' &= 1 \\
&\quad d1 \cdot 'a' &= 1 \\
&\quad d1 \cdot 'b' &= 0 \\
m2 &::= \text{Automaton} \\
m2 &= ([0, 1, 2], \{a, b\}, d2, 0, \{1\}) \text{ where } d2 \cdot 'a' &= 1 \\
&\quad d2 \cdot 'b' &= 2 \\
&\quad d2 \cdot 'a' &= 2 \\
&\quad d2 \cdot 'b' &= 0 \\
&\quad d2 \cdot 'a' &= 2 \\
&\quad d2 \cdot 'b' &= 2 \\
\end{align*}
\]

![Diagram](Figure 1: Example of finite automata)

We note that $L(M_1) = \{ w \in \Sigma^* | |w|_a = 1 \mod 2 \}$, and $L(M_2) = \{ a(ba)^n | n = 0, 1, ... \}$. In our module the function `Automaton.language` returns the accepted language. To compute the accepted language generated by $M_1$, we use `Automaton.language m1`, where `m1` is the automaton described in Haskell.

Following is a code for finding accepted language and their executions.

\[
\begin{align*}
\text{accepts} &::= \text{Automaton} \rightarrow [\text{String}] \rightarrow [\text{String}] \\
\text{accepts m ss} &= [w | w < ss, (\text{asterd d s0 w}) \ \text{elem} \ f] \text{ where } (q, s, d, s0, f) = m \\
\end{align*}
\]

3. Sticker Module

Definition 3. Let $\Sigma$ be a set of alphabet and $Z$ a set of integers and $\rho \subseteq \Sigma \times Z$. An element $(i, r, u, v)$ of $\Sigma^* \times \Sigma^* \times Z$ is a **domino** over $(\Sigma, \rho)$, if the following conditions holds:

- If $n \geq 0$ then $(l[i + n], r[i]) \in \rho$, for $1 \leq i \leq \min(|l| - n, |r|)$
- If $n < 0$ then $(l[i], r[i - n]) \in \rho$, for $1 \leq i \leq \min(|r| + n, |l|)$

We denote the set of all dominoes over $(\Sigma, \rho)$ by $D$.

The possible shapes of the dominoes are illustrated as follows:

\[
\begin{pmatrix}
x & x & x & x & x \\
\mid & \mid & \mid & \mid & \mid \\
x & v & x & v' & x' \\
\end{pmatrix}
\]

where $x = x_1 x_2 \cdots x_n, x' = x'_1 x'_2 \cdots x'_n, u, v, \rho \subseteq \Sigma^*$ and $(x, x') \in \rho$ for $1 \leq i \leq n$. Sticky ends can be placed in the upper strand or lower strand.

Example 2. We can represent a double stranded sequence $\begin{pmatrix}
\lambda & A \\
\lambda & C
\end{pmatrix}$ $\begin{pmatrix}
AT \\
TG
\end{pmatrix}$ in our module. Similarly, $\begin{pmatrix}
G & A \\
\lambda & TA
\end{pmatrix}$ can be represented by $\begin{pmatrix}
G & A \\
\lambda & TA
\end{pmatrix}$.

Definition 4. The sticking operator $\mu : D \times D \rightarrow D \cup \{\perp\}$ is defined as follows:

\[
\mu((l_1, r_1, n_1), (l_2, r_2, n_2)) = \begin{cases} 
(l_1 l_2, r_1 r_2, n_1) & (\text{if } \ast) \\
\perp & (\text{otherwise})
\end{cases}
\]

$\ast$ $(l_1 l_2, r_1 r_2, n_1) \in D$ and $n_1 + |r_1| - |l_1| = n_2$

Definition 5. Sticker System $\gamma$ is a four tuple $\gamma = (\Sigma, \rho, A, R)$ of an alphabet set $\Sigma$, $\rho \subseteq \Sigma \times \Sigma$, a finite set of axioms $A(\subseteq D)$ and a finite set of pairs of dominoes $R \subseteq D \times D$.

Let $Q = \{q_0, q_1, ..., q_k\}$ be a finite set, which consists of $k+1$ elements. For a finite automaton $M = (Q, \Sigma, \delta, q_0, F_M)$, the sticker system $\gamma_M = (\Sigma, \rho, A, R)$ is defined as follows:

\[
\rho = \{(a, a)|a \in \Sigma\} \\
A = A_1 \cup A_2 \\
A_1 = \{(x, x, 0) | x \in L(M), |x| \leq k + 2\}
\]
The language \( \leq \mid u \mid \) is defined by \( \mu(\cdot) \). For any \( (\lambda, \lambda, 0) \), \((x, u, x', -|u|)\) is \( L(\gamma) \). Then \((l', r', \delta')\) is \((x, u'x', -|u'|)\) for some \( x' \in \Sigma^* \) and \( 1 \leq \delta' \). So there exist \((\lambda, \lambda, 0), (x' u' x', -|u'|)\) in \( F \) and \( a \Rightarrow_\gamma^* (x, u, 0) \Rightarrow (l, r, 0) \). 

(vii) \( F = \bigcup_{i=1}^{k+1} Z_i \) 

(Proof) (i),(iii),(iv) and (vii) are trivial.

(ii) Let \((l, 0)\) be a domino and \((\lambda, \lambda, 0), (x, u, x, -|v|)\) in \( R_1 \). If \( \mu(l, r, 0, (x, u, x, -|v|)) \neq \bot \) then \( \mu(l, l, 0, 0) \notin L(\gamma) \).

Let \((x, u, x, 0)\) be a domino and \( x, u \in \Sigma^* \) and \( 1 \leq \delta' \). So there exist \((\lambda, \lambda, 0), (x' u', x', -|u'|)\) in \( R_1 \) such that \( \mu((x' u', x', 0), (x', u', -|u'|)) = (x' u' x', u' x', 0) = (x, u, x, 0) \). So we have \( x = x' u' x', |x| = k + 2 \) and \( 1 \leq |u'| \leq k + 1 \).

(vi) \((\lambda, \lambda, 0), (x, u, x, 0) \Rightarrow (l, r, 0) \). Since \((x', u', -|u'|) = (q_{i-1}, 1) \) such that \( \delta^*(q_{i-1}, 1) = (l, 0) \) and \( (x, u, x, 0) \) in \( F \), we have \( x = x' u' x', |x| = k + 2 \) and \( 1 \leq |u'| \leq k + 1 \).

(\( \Rightarrow \)) Let \((x, u, x, 0) \in Y_i \). If \( |x| \leq k + 2 \) then \( \delta^*(q_0, q_0) = q_{i-1} \) by (iii) and (iv). If \( |x| > k + 2 \) then there exists a domino \((x' u', x', 0) \in Y_i \) and \((\lambda, \lambda, 0), (x' u', u' x', -|u'|)\) in \( R_1 \) such that \( x = x' u' x' \). Since \((x' u', x', 0) \in Y_i \), we have \( \delta^*(q_0, x') = q_{i-1} \). Since \((\lambda, \lambda, 0), (x' u', x', -|u'|)\) in \( R_1 \), we have \( \delta^*(q_{i-1}, x') = q_{i-1} \). So we have \( \delta^*(q_0, x') = \delta^*(q_{i-1}, x') = q_{i-1} \).

(\( \Leftarrow \)) Let \((x, u, x, 0) \) be an element of the right-hand set. That is \( \delta^*(q_0, x) = q_{i-1}, |u| = k + 1 \) and \( |x| = k + 2 \). We prove \((x' u', x, 0) \in Y_i \) using induction on \( n \). Assume \((x, u, x, 0) \in Y_i \) for any \( x, u, x, 0 \in \Sigma^* \) such that \( |x| = n + 2 \).

(\( \Rightarrow \)) Let \((x, u, x, 0) \) be a domino and \( |x| = n + 2 \). We put \( x = x' u' x' \) where \( |x'| = k + 2, 1 \leq |u'| \leq k + 1 \) and \( \delta^*(q_0, x' u') = q_{i-1} \). Since \( |x'| = |x| - |x' u'| = n + k + 1 \), we have \( (x', u', x', 0) \in Y_i \) by the hypothesis of the induction. Since \( \delta^*(q_{i-1}, x' u') = \delta^*(q_0, x' u' x') = q_{i-1} \), we have \((\lambda, \lambda, 0), (x' u', u' x', -|u'|)\) in \( R_1 \). So \( \mu((x' u', x', 0), (x' u', u' x', -|u'|)) = (x' u' x', u' x', 0) \) and \((x, u, x, 0) \) in \( Y_i \).

The idea of the proof of the next theorem is originally introduced by Paun and Rozenberg ([3]) in 1998. It lacked several conditions and formal proofs in their paper. We modified and improved them and proved it using our formulations.
Theorem 1. Define the sticker system $\gamma = \gamma_M$ for a finite automaton $M = (Q, \Sigma, \delta, q_0, F_M)$. Then $L(\gamma) = L(M)$.

(Proof) Let $w \in L(\gamma_M)$. Then we have $a \Rightarrow^* (w, w, 0)$ for some $a \in A$. If $(w, w, 0) \in A$ then $w \in L(M)$ by definition.

Since $(w, w, 0) \notin A$ there exist $(xu, x, 0)$ and $((\lambda, \lambda, 0), (x', ux', -|u|)) \in F$ such that $a \Rightarrow^* (xu, x, 0)$ and $\mu((xu, x, 0), (x', ux', -|u|)) = (w, w, 0)$. Since $a \Rightarrow^* (xu, x, 0)$, we have $\delta^*(q_0, xu) = q_{u-1}$ from Lemma 1(iii). Since $((\lambda, \lambda, 0), (x', ux', -|u|)) \in F$, we have $\delta^*(q_{u-1}, x') \in F_M$. Since $\delta^*(q_0, w) = \delta^*(q_{u-1}, x') = \delta^*(q_{u-1}, x') \in F_M$, we have $w \in L(M)$.

(\Rightarrow) Let $w \in L(M)$. If $|w| \leq k + 2$ then $(w, w, 0) \in A$ and $w \in L(\gamma_M)$.

If $k + 2 \leq |w| \leq 2(k + 2)$ then we put $w = w'x'$ where $|w'| = k + 2$. If $\delta^*(q_0, w') = q_{i-1}$ then $(w''u, w''u, 0) \in A$ where $w' = w''u$ and $|u| = i$. Since $\delta^*(q_{i-1}, x') = \delta^*(q_0, w', x') = \delta^*(q_0, w, i)$, we have $((\lambda, \lambda, 0), (x', ux', -i)) \in F$. Since $\mu((w''u, w''u, 0), (x', ux', -i)) = (w''ux', w''ux', 0) = (w, w, 0)$, we have $(w''u, w''u, 0) \Rightarrow (w, w, 0) \in L(\gamma_M)$. If $|w| > 2(k + 2)$, let $w = w'x'$ where $(k + 2) \mid |w'$ and $|x'| \leq k + 2$. If $\delta^*(q_0, w) = q_{i-1}$ then $(w''u, w''u, 0) \in Y_i$ where $\delta^*(q_{i-1}, x') = \delta^*(q_0, w', x') = \delta^*(q_0, w, i)$, we have $((\lambda, \lambda, 0), (x', ux', -i)) \in F$. Since $\mu((w''u, w''u, 0), (x', ux', -i)) = (w''ux', w''ux', 0) = (w, w, 0)$, we have $(w''u, w''u, 0, w, w, 0) \in L(\gamma_M)$.

Note: We correct the limit length of $x$ from $k$ to $k + 2$ in [3]. Consider the sticker system $\gamma_M$ generated by the automaton $M_1$ in Example 1 again. Since $(\lambda, \lambda, 0), (aba, baba, -1)) \in F$ and $\mu((\lambda, \lambda, 0), (aba, baba, -1)) = (ababa, ababa, 0)$, we have $ababa \in L(\gamma_M)$. In the definition of $F$, the limit of length $|x|$ for $(x, x, x, -|w|) \in F$ is $k = 1$. Since $|aba| > 1$, we do not have $((\lambda, \lambda, 0), (aba, baba, -1)) \in F$ by the definition in [3]. So even $ababa \in L(\gamma_M)$, $ababa \notin L(\gamma_M)$ according to the definition of sticker system described in [3].

4. GRAMMAR MODULE

Definition 7. A grammar is a four tuple $G = (T, N, R, S)$ of terminal symbols $T$, non-terminal symbols $N$, transformation rules $R$ and a start symbol $S$.

Definition 8. The language $L(G)$ generated by grammar $G = (\Sigma, N, R, S)$ is defined as $L(G) = \{w \in \Sigma^* | \Rightarrow^* \gamma w \}$. For a grammar $g = G$, (Grammar.language g) computes the $L(G)$.

Example 4. The grammars $G_1 = (\{a, b\}, \{S\}, \{S \rightarrow ab\}, S \rightarrow ab)$, $S$ and $G_2 = (\{a, b\}, \{S\}, A, \{S \rightarrow ab\}, A \rightarrow a, A \rightarrow a)$, $S$ and $G_3 = (\{a, b\}, \{S\}, A, \{S \rightarrow ab\}, A \rightarrow a, A \rightarrow a)$, $S$) are expressed as follows using our Haskell Modules.

```
gex1::Grammar  
gex1=(["aa","bb"],["S",["ab","S","b","ab"]],["S"])
gex2::Grammar  
gex2=(["aa","bb"],["S",["ab","S","b","ab"]],["S"])
```

For a string $w = x_1x_2 \cdots x_n$ and $1 \leq i \leq n$, $Left(w, i) = x_1 \cdots x_i$ and $Right(w, i) = x_{i-1+1} \cdots x_n$.

Definition 9. Let $N = \{X_1, X_2, \cdots, X_k\}$ be a finite set of $k$ non-terminal symbols and $S = X_1$. For a linear grammar $G = (\Sigma, T, R, S)$, the sticker system $\gamma_G = (\rho, \rho, A, R)$ is defined similar to [3] as follows.

```
\rho = \{(a, a) | a \in \Sigma\}
X_1 = S\text{if} i = 1\text{then} X_i = S
T(i, k) = \{w \in \Sigma^* | X_i \Rightarrow^* w, |w| = k\}
T(i, l, r) = \{(w_1, x, w_r) \in (\Sigma^* \times N \times \Sigma^*) | X_i \Rightarrow w X_j w_r, |w_1| = l, |w_r| = r\}
X_1 = A_1 \cup A_2 \cup A_3
A_1 = \{(x, x, 0) | x \in T(1, m), m \leq 3k + 2\}
A_2 = \bigcup_{i=1}^{k} \{(w, x, x, |w|) | w \in T(i, m), i + 1 \leq m \leq 3k + 2, x = Right(w, m - i), u = Left(w, i)\}
A_3 = \bigcup_{i=1}^{k} \{(xu, x, 0) | w \in T(i, m), i + 1 \leq m \leq 3k + 2, x = Left(w, m - i), u = Right(w, i)\}
R = R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6
R_1 = \bigcup_{i=1}^{k} \bigcup_{j=0}^{i} \{(w u v, x, z, 0) | w = T(i, k + 1), z = Left(w, i), x = Right(w, i), |v| = j\}
R_2 = \bigcup_{i=1}^{k} \bigcup_{j=0}^{i} \{(x, x, 0) | (z, w, z, 0) \in T(i, k + 1), z = Left(w, k + 1 - i), x = Right(w, i), |v| = j\}
```
We define a set \( R \) to allow the form \( x \Rightarrow y \) using induction on the length of \( x \). Since \( X_1 \Rightarrow \gamma_1 w' \) and \( |w'| \geq k+1 \), there exist \( x' \) and \( u' \) satisfying \( w' = x'u' \) and \( |u'| = j \) such that \( a \Rightarrow (x'u', x', 0) \) for some \( a \in A \). Let \( x = y_{m-1} \cdots x_1 y_1 \) and \( z = x_1 x_2 \cdots x_m \). Since \( X_1 \Rightarrow \gamma_1 z \), \( xu \) and \( |x| = i \), we have \( (x, y, x', u, [w]) \in R_4 \) and \( (x'u', x', 0) \Rightarrow \gamma_1 (x'u'uxu, x'u'uxu, 0) \). Since \( x'u'uxu = w \) and \( |x| = i \), we have \( w \in Y_i \).

Next we prove \( L(G) \subseteq L(\gamma). \) Let \( w \in L(G) \). If \( |w| \leq 3k + 2 \) then \( (w, u, v) \in A \) and \( w \in L(\gamma) \). Assume \( |w| > 3k + 2 \). According to the limitation of production rules in \( G \), we have \( S \Rightarrow \gamma \) for any \( \gamma \in R \). Since \( x_1 x_2 \cdots x_m y_1 \) and \( x_1 x_2 \cdots x_m y_1 \leq 2k + 2 \), we have \( (x_1 x_2 \cdots x_m) + |y_1 y_2 \cdots y_m| \leq 2k + 2 \). Since \( \mu(x_1 x_2 \cdots x_m, 0), (y_1 y_2 \cdots y_m, 0) \in R_6 \). Since \( \mu(x_1 x_2 \cdots x_m, 0), (y_1 y_2 \cdots y_m, 0) \in R_6 \) we have \( w \in L(\gamma) \).

**Example 5.** Consider the Language generated by linear grammar \( G = \{ \Sigma \}, \{ a, b \}, S \to ab, S \to aSb \). The language generated by \( G \) is \( L(G) = \{ a^ib^n | i \geq 1 \} \).

Now we can induce the domino \( \begin{array}{cccc} a & a & a & b \\ a & a & b & b \end{array} \) by using pair of elements \( \begin{array}{c} a \\ b \\ b \end{array} \) and \( \begin{array}{c} a \\ b \\ b \end{array} \) in \( R_4 \) and \( \begin{array}{c} a \\ b \\ b \end{array} \) in \( A_3 \). All of elements in \( A \) and \( R \) are listed in Appendix.

**5. Conclusion**

We can define the dominos using set theoretical notations in Haskell and simulate sticker systems, finite automata and grammar systems. Using our system, we could find some insufficient conditions to construct the sticker systems written in [3]. One of related work is implementation of HaLex [5]. HaLex is a Haskell library enables us to model and manipulate a regular language. HaLex also provide the facility for defining deterministic and non-deterministic finite automata, regular expressions etc. It does not represent an infinite set as a language. One of the merits of our modules is treating the generated languages as an infinite set using lazy evaluations.

**Acknowledgment**

We would like to thank Ai Omodaka for her contribution to write Haskell codes and the great support made during
the research.

REFERENCES


APPENDIX

In Appendix, we show examples of sticker systems generated from automata and grammar by using our Haskell module functions.

Example 6. For an automaton $M_1 = (\{0,1\}, \Sigma, \delta, 0, \{1\})$ in Example 1, we have the sticker system $\gamma_{M_1}$ as follows.

$$\gamma_{M_1} = (\Sigma, \rho, A, R)$$

$$\rho = \{(a,a),(b,b)\}$$

$$A = A_1 \cup A_2$$

$$b \quad b \quad ab \quad a \quad ba \quad b \quad a \quad ab \quad bb \quad bbb \quad \ldots$$

$$aa \quad aba \quad b \quad ab \quad bb \quad bbb \quad a \quad ab \quad bb \quad bbb \quad \ldots$$

$$R = D \cup F$$

$$R = R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6$$

(1) (1) (1) (1) (1) (1)

(2) (2) (2) (2) (2) (2)

(3) (3) (3) (3) (3) (3)

(4) (4) (4) (4) (4) (4)

(5) (5) (5) (5) (5) (5)

(6) (6) (6) (6) (6) (6)

K.K.K.R. Perera
Graduate School of Mathematics, Kyushu University, Japan.
E-mail: kisani(at)math.kyushu-u.ac.jp

Y. Mizuguchi
Faculty of Mathematics, Kyushu University, Japan.
E-mail: ym(at)math.kyushu-u.ac.jp