Haskell Modules for Sticker Systems (Ver.1)

Yoshihiro MIZOGUCHI
Institute of Mathematics for Industry, Kyushu University, JAPAN.

2007/03/10 (Ver.0)
2013/06/20 (Ver.1)*

Contents

1 Introduction

The sticker system is a formal DNA computing model introduced by [2, 4]. The sticker operation used in the sticker system is the most basic and essential operation based on using Watson-Click complementarity and sticky ends.

In this paper, we introduce Haskell program modules for realize and analyse features related to the sticker system. Most of realized operations and constructions are based on the facts in [4], but we modify the expression of domino and sticker operations for realizing Haskell functions.

We modify the definition of 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 $\left(\begin{array}{c}\lambda \\ C\end{array}\right)\left[\begin{array}{c}AT \\ TA\end{array}\right]\left(\begin{array}{c}GC \\ \lambda\end{array}\right)$ is represented by

*https://github.com/ymizoguchi/HaskellStickerModules.git

$(ATGC, CTA, -1)$. According to this modification, the sticker operation $\mu : D \times D \to D \cup \{\perp\}$ is reformed to one equation $\mu((l_1, r_1, x_1), (l_2, r_2, x_2)) = (l_1 l_2, r_1 r_2, x_1)$, if $(l_1 l_2, r_1 r_2, x) \in D$ and $x_1 + length(r_1) - length(l_1) = x_2$. Using this simple representation of domino and sticker operations, we implement them in Haskell.

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

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

In Haskell notations, we have following function definitions.

```
aA2::Automaton->[Domino]
aA2 m@(q,s,d,q0,f) = concat [(aA2' m i)| i<- [1..(k+1)]] where k = (length q)-1
aA2'::Automaton->Int->[Domino]
aA2' m@(q,s,d,q0,f) i =  [(x++u,x,0)| (x,u) <- xupair, (dstar d 0 (x++u)) ==(i-1)]
       where xupair = [(x,u)|x<-(sigmann s (k+2-i)), u<-(sigman s i)]
             k = (length q)-1
```

The modules is composed of 4 parts, **Automaton** module (string, automaton and their languages), **Sticker** module (domino, sticker system and their languages), **Grammar** module (context-free grammar, linear-grammar and their languages), and **LaTeX** output module (pretty printing for domino). 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 introduced in [4].

In this paper, we introduce implemented module functions and examples of sticker systems with equivalent power of concrete finite automata and linear grammars.

2 Automaton Module

2.1 Basic Definitions

Let Σ be an alphabet set. Σ^* is the set of all strings over Σ including the empty string ε. For a natural number n, $\Sigma^n = \{w \in \Sigma^* | |w| = n\}$ and $\Sigma^{\leq n} = \{w \in \Sigma^* | 1 \leq |w| \leq n\}$.

Haskell function `(sstar s)` computes the infinite elements set Σ^* over `s`$= \Sigma$. We can also compute Σ^n and $\Sigma^{\leq n}$ by `(sigman n s)` and `(sigmann n s)`, respectively.

We use types `Symbol` for alphabets, `SymbolString` for strings, and `SymbolSet` for sets of alphabets.

```
module Automaton where
import Data.List

type Symbol = Char
type SymbolString = [Symbol]
type SymbolSet    = [Symbol]

sstar :: Eq a => [a] -> [[a]]
sstar [] = []
sstar s  = [[]] ++ (union' [ [[x] ++ w | w <- (sstar s)] | x <- s ])

sigman :: Eq a => [a] -> Int -> [[a]]
sigman s 0 = [[]]
```

```
sigman s n = concat' ss (sigman s (n-1))
             where ss = map (\x -> [x]) s

sigmann :: Eq a => [a] -> Int -> [[a]]
sigmann s 0 = []
sigmann s n = (sigmann s (n-1)) ++ (sigman s n)
```

We call a subset A of Σ^* as a language over Σ. For a set of languages `u`$= S$, the set $\cup\{x|x \in S\}$ is computed by `(union' u)`. For languages `x`$= X$, and `y`$= Y$, the set $X \cdot Y = \{xy|x \in X,\ y \in Y\}$ is computed by `(concat' x y)`. And the set $X^* = \cup\{X^i|i \in \mathbf{N}\}$ is computed by `(star x)`.

```
union':: Eq a =>[[a]]->[a]
union' [] = []
union' (x:xs) | x == []   = union' xs
             | otherwise = [head x]++(union' (xs++[tail x]))

concat'::Eq a => [[a]]->[[a]]->[[a]]
concat' [] y = []
concat' x [] = []
concat' (xh:xs) (yh:ys) =  [xh++yh]
   ++ union' [ (concat' [xh] ys), (concat' xs [yh]), (concat' xs ys)]

star :: Eq a => [[a]] -> [[a]]
star [] = []
star s  = [[]] ++ (union' [ [x ++ w | w <- (star s)] | x <- s ])
```

We note that the functions `union'`, `concat'`, and `star` are applicable for languages which contains infinite elements. Even though the fuction `union` and `concat` in Data.List modules are not applicable for infinite sets.

2.2 Automaton

We use types `State` for states, `States` for sets of states, and `Automaton` for automata.

We can naturally extend a state transition function $\delta : Q \times \Sigma \to Q$ to $\delta^* : Q \times \Sigma^* \to Q$. For a fuction `d`$= \delta$, the function δ^* is computed by `dstar d`.

```
type State = Int
type States = [State]
type Automaton = (States, [Symbol], State->Symbol->State, State, States)

dstar::(State->Symbol->State)->State->SymbolString->State
dstar d s [] = s
dstar d s (a:w) = dstar d (d s a) w
```

For an automaton `m`$= M = (Q, \Sigma, \delta, q_0, F)$, an accepted language

$$L(M) = \{w \in \Sigma^* \,|\, \delta^*(w) \in\ F\}$$

is computed by `(language m)`.

```
accepts::Automaton->[String]->[String]
accepts (q,s,d,s0,f) ss = [w | w <- ss, (dstar d s0 w) `elem` f]

language::Automaton ->[String]
language m@(q,s,d,s0,f) = accepts m (sstar s)
```

2.3 Translation

In this section, we defines meta-functions for transformations.

A function $f : A \to 2^B$ is naturally extended to a function $\hat{f} : 2^A \to 2^B$ by defining $\hat{f}(X) = \cup\{f(x)|x \in X\}$ for $X \subset A$. The function $\hat{f}$ is computed by `(power f)`.

```
power::(Eq a,Eq b)=>(a->[b])->[a]->[b]
power f ss = union' [(f s)|s<-ss]
```

For a function $f : A \to 2^A$, n-th repeated function is defined by $\hat{f}^n(X) = \cup\{f(x)|x \in f^{n-1}(X)\}$, where $f^0(X) = \phi$. For a predicate $P : A \to \{true, false\}$, we define the function $\hat{f}^n_P(X) = \cup\{f(x) \,|\, (x \in f^{n-1}_P(X)) \wedge P(f(x))\,\}$.

$\hat{f}^n$ and $hatf^n_P$ are computed by `(nstep n f)` and `(nfstep p n f)`, respectively.

```
nstep::Eq a=>Int->(a->[a])->[a]->[a]
nstep 0 _ _  = []
nstep 1 f ss = (power f) ss
nstep n f ss = (power f) $ nstep (n-1) f ss

nfstep::Eq a=>(a->Bool)->Int->(a->[a])->[a]->[a]
nfstep _ 0 _ _  = []
nfstep p 1 f ss = filter p ((power f) ss)
nfstep p n f ss = filter p $ (power f) $ nfstep p (n-1) f ss
```

$f^*(X) = \cup\{f^i(X)|i \in \mathbf{N}\}$ is computed by `(sstep f)`. $f^*_P(X) = \cup\{f^i_P(X)|i \in \mathbf{N}\}$ is computed by `(sfstep p f)`.

```
sstep::Eq a=>(a->[a])->[a]->[a]
sstep f s0 = nub $ s0 ++ (g f s0)
      where g f [] = []
            g f ss = nub $ ss ++ (g f ((power f) ss))

sfstep::Eq a=>(a->Bool)->(a->[a])->[a]->[a]
sfstep p f s0 = nub $ s0' ++ (g f s0')
  where s0' = filter p s0
        g f [] = []
        g f ss = nub $ (filter p ss) ++ (g f ((power f) (filter p ss)))
```

2.4 Examples

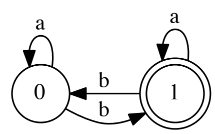
```
module AutomatonEx where
import Data.List
import Automaton
```

Example 1 $L(M_p) = \{w \in \Sigma^* \,|\, |w|_b \textbf{ mod } 2 = 1\,\}$, $L(M_1) = \{a(ba)^n|n = 0, 1, \cdots\}$.

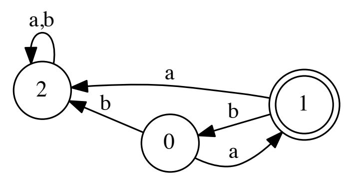
```
mp::Automaton
mp = ([0,1], ['a','b'], dp, 0, [1])
    where dp 0 'a' = 0
          dp 0 'b' = 1
          dp 1 'a' = 1
          dp 1 'b' = 0

m1::Automaton
m1 = ([0,1,2], ['a','b'], d, 0, [1])
     where d 0 'a' = 1
           d 0 'b' = 2
           d 1 'a' = 2
```

```
d 1 'b' = 0
d 2 'a' = 2
d 2 'b' = 2
```

M_p



M_1




```
*AutomatonEx> take 10 $ Automaton.language mp
["b","ba","ab","baa","aba","aab","bbb","baaa","abaa","aaba"]
*AutomatonEx> take 5 $ Automaton.language m1
["a","aba","ababa","abababa","ababababa"]
```

```
t001 = take 10 $ Automaton.language mp
t002 = take 5 $ Automaton.language m1
```

3 Sticker Module

3.1 Domino

Let Σ be a set of alphabvet, $\mathbf{Z}$ a set of integers, and $\rho \subseteq \Sigma \times \Sigma$D

Definition 1 (Domino) *An element (l, r, x) of $\Sigma^* \times \Sigma^* \times \mathbf{Z}$ is a* **domino** *over (Σ, ρ), if following conditions holds:*

- *If $x \geq 0$ then $(l[i+x], r[i]) \in \rho, 1 \leq i \leq min(length(r) - x, length(l))$*
- *If $x < 0$ then $(l[i], r[i-1]) \in \rho, 1 \leq i \leq min(length(r) + x, length(l))$*

We denote the set of all dominos over (Σ, ρ) by D. We define $WK = \{(l, r, 0) \in D || l| = |r|\}$.

```
module Sticker where

import Data.List
import Automaton

type Domino = (SymbolString,SymbolString,Int)
type Rho = [(Symbol,Symbol)]
type Lrset = [(Symbol,Symbol)]
type Sticker = ([Symbol],Rho,[Domino],[(Domino,Domino)])
```

`(lrcheck rho (l,r,x))` is a function to chek (l, r, x) is a domino or not.

```
lr :: Domino -> Lrset

lr ([],[],x) = []
lr ([],(rh:rt),x) = []
```

```
lr ((lh:lt),[],x) = []
lr (l@(lh:lt),r@(rh:rt),x) = if (x > 0) then (lr (lt,r,x-1))
                             else (if (x == 0) then ((lh,rh):(lr (lt,rt,0)))
                                     else (lr (l,rt,x+1)))
lrcheck :: Rho -> Domino -> Bool
lrcheck rho (l,r,x) = ((length lrset1) == (length lrset2))
                    where lrset2 = filter (\x -> (x `elem` rho)) lrset1
                          lrset1 = (lr (l,r,x))
wk :: Domino -> Bool
wk (l,r,x) | x==0 = (length l) == (length r)
           | otherwise = False
```

Definition 2 (Sticker Operation) *Sticker operation $\mu : D \times D \to D \cup \{\perp\}$ is defined as follows:*

$$\mu((l_1, r_1, x_1), (l_2, r_2, x_2)) = \begin{cases} (l_1 l_2, r_1 r_2, x_1) & \textit{(if the condition (*) holds)} \\ \perp & \textit{(otherwise)} \end{cases}$$

(*) $(l_1 l_2, r_1 r_2, x) \in D$ *and* $x_1 + length(r_1) - length(l_1) = x_2$

The function `mu` is an implementation of the sticker operation μ.

```
mu :: Rho -> Domino -> Domino -> [Domino]
mu rho (l1,r1,x1) (l2,r2,x2) =
        if (((x1 + (length r1) - (length l1)) == x2) &&
           (lrcheck rho (l1++l2,r1++r2,x1))) then  [(l1++l2,r1++r2,x1)]
           else []

mu' :: Rho -> (Domino,Domino) -> Domino -> [Domino]
mu' rho (l,r) d = concat $ map (mu rho l) $ mu rho d r
```

3.2 Sticker System

Definition 3 (Sticker System) *Sticker System γ is four tuple $\gamma = (\Sigma, \rho, A, R)$ of an alphabet set Σ, $\rho \subseteq \Sigma \times \Sigma$, a finite set of axioms $A(\subseteq D)$ and a finite set of dominos $R \subseteq D \times D$.*

For a sticker system $\gamma = (\Sigma, \rho, A, R)$, we define a relation $\Rightarrow$ as follows.

$$x \Rightarrow y \quad \overset{def}{\Longleftrightarrow} \quad \exists (u, v) \in R, y = \mu(u, \mu(x, v))$$

Let $\Rightarrow^*$ be the reflective and transitive coluser of $\Rightarrow$.

The set of dominos $LM(\gamma)$ generated by γ is

$$LM(\gamma) = \{b \in WK | a \Rightarrow^* b, a \in A\}.$$

The language $L(\gamma)$ generated by γ is

$$L(\gamma) = \{l \in \Sigma^* | a \Rightarrow (l, r, 0) \in WK, \exists r \in \Sigma^*, \exists a \in A\}.$$

```
onestep :: Rho -> [(Domino,Domino)] -> Domino -> [Domino]
onestep rho rr d = concat $ map (\x->(mu' rho x d)) rr

language::Sticker->[SymbolString]
language stk = map f $ filter wk $ Sticker.generate stk
         where f (x,y,z) = x

generate::Sticker->[Domino]
generate (s,rho,a,r) = (sstep (Sticker.onestep rho r)) a
```

3.3 Sticker System vs Automaton

Definition 4 *For a finite automaton $M = (Q, \Sigma, \delta, q_0, F_M)$, the sticker sytem $\gamma_M = (\Sigma, \rho, A, R)$ is defined as follows.*

$$
\begin{aligned}
\rho &= \{(a,a) | a \in \Sigma\} \\
A &= A_1 \cup A_2 \\
A_1 &= \{(x,x,0) | x \in L(M), |x| \leq k+2\} \\
A_2 &= \bigcup_{i=1}^{k+1} \{(xu, x, 0) | x \in \Sigma^{\leq (k+2-i)}, u \in \Sigma^i, \delta^*(0, xu) = i-1\} \\
R &= D \cup F \\
D &= \bigcup_{i=1}^{k+1} \bigcup_{j=1}^{k+1} \{((\lambda,\lambda,0),(xu, vx, -|v|)) | x \in \Sigma^{\leq (k+2-i)}, u \in \Sigma^i, v \in \Sigma^j \\
&\quad \delta^*(j-1, xu) = i-1\} \\
F &= \bigcup_{i=1}^{k+1} \bigcup_{j=1}^{k+1} \{((\lambda,\lambda,0),(x, vx, -|v|)) | v \in \Sigma^j, x \in \Sigma^i, \\
&\quad \delta(j-1, x) \in F_M\} \\
k &= |Q| - 1
\end{aligned}
$$

Theorem 1 ([4](Theorem 7))

$$L(\gamma_M) = L(M)$$

■

```
module StickerEx1 where

import Data.List
import Automaton
import Sticker
import AutomatonEx

aA::Automaton->[Domino]
aA m = (aA1 m)++(aA2 m)

aA1::Automaton->[Domino]
aA1 m@(q,s,d,q0,f) = [(x,x,0)|x<-(accepts m (sigmann s (k+2)))]
                     where k = (length q)-1

aA2::Automaton->[Domino]
aA2 m@(q,s,d,q0,f) = concat [(aA2' m i)| i<- [1..(k+1)]]
                     where k = (length q)-1

aA2'::Automaton->Int->[Domino]
aA2' m@(q,s,d,q0,f) i =  [(x++u,x,0)|
                          (x,u) <- xupair, (dstar d 0 (x++u)) ==(i-1)]
       where xupair = [(x,u)|x<-(sigmann s (k+2-i)), u<-(sigman s i)]
             k = (length q)-1

dD::Automaton->[Domino]
dD m@(q,s,d,q0,f) = concat [(dD' m (i,j))|i<-[1..(k+1)],j<-[1..(k+1)]]
                    where k = (length q)-1

dD'::Automaton->(Int,Int)->[Domino]
```

```
dD' m@(q,s,d,q0,f) (i,j) = concat [
            [(x++u,v++x,-(length v))|
                x<-(sigmann s ((k+2)-i)), (dstar d (j-1) (x++u))==(i-1)]|
                                    u<- (sigman s i), v<-(sigman s j)]
                    where k = (length q)-1

dF::Automaton->[Domino]
dF m@(q,s,d,q0,f) = concat $ map (dF' m) [(i,j)|i<-[1..(k+2)],j<-[1..(k+1)]]
      where k = (length q)-1

dF'::Automaton->(Int,Int)->[Domino]
dF' m@(q,s,d,q0,f) (i,j)=
     concat [
       [(x,v++x,-(length v))| x<-(sigman s i),
                              (dstar d ((length v)-1) x) `elem` f
             ]| v<-(sigman s j)]
     where k = (length q)-1

mGamma::Automaton->Sticker
mGamma m@(q,s,d,q0,f) = (s,rho,(aA m),dd)
        where dd = map (\x -> (("","",0),x)) ((dD m)++(dF m))
              rho = map (\x -> (x,x)) s
```

Example 2 *The followings are sticker systems induced by finite automata defined in Example 1.*

```
*Sticker> let (s,rho,a,r)=mGamma mp in (map snd r)
[("aaa","aaa",-1),("bba","abb",-1),("abb","aab",-1),("bab","aba",-1),
("aaa","baa",-1),("bba","bbb",-1),("abb","bab",-1),("bab","bba",-1),
("aba","aaab",-2),("baa","aaba",-2),("aab","aaaa",-2),
("bbb","aabb",-2),("aba","abab",-2),("baa","abba",-2),
("aab","abaa",-2),("bbb","abbb",-2),("aba","baab",-2),
("baa","baba",-2),("aab","baaa",-2),("bbb","babb",-2),
("aba","bbab",-2),("baa","bbba",-2),("aab","bbaa",-2),
("bbb","bbbb",-2),("baa","ab",-1),("aab","aa",-1),("aba","aa",-1),
("bbb","ab",-1),("baa","bb",-1),("aab","ba",-1),("aba","ba",-1),
("bbb","bb",-1),("aaa","aaa",-2),("bab","aab",-2),("bba","aab",-2),
("abb","aaa",-2),("aaa","aba",-2),("bab","abb",-2),("bba","abb",-2),
("abb","aba",-2),("aaa","baa",-2),("bab","bab",-2),("bba","bab",-2),
("abb","baa",-2),("aaa","bba",-2),("bab","bbb",-2),("bba","bbb",-2),
("abb","bba",-2),("b","ab",-1),("b","bb",-1),("a","aaa",-2),
("a","aba",-2),("a","baa",-2),("a","bba",-2),("ab","aab",-1),
("ba","aba",-1),("ab","bab",-1),("ba","bba",-1),("aa","aaaa",-2),
("bb","aabb",-2),("aa","abaa",-2),("bb","abbb",-2),("aa","baaa",-2),
("bb","babb",-2),("aa","bbaa",-2),("bb","bbbb",-2),("aab","aaab",-1),
("baa","abaa",-1),("aba","aaba",-1),("bbb","abbb",-1),
("aab","baab",-1),("baa","bbaa",-1),("aba","baba",-1),
("bbb","bbbb",-1),("aaa","aaaaa",-2),("bab","aabab",-2),
("bba","aabba",-2),("abb","aaabb",-2),("aaa","abaaa",-2),
("bab","abbab",-2),("bba","abbba",-2),("abb","ababb",-2),
("aaa","baaaa",-2),("bab","babab",-2),("bba","babba",-2),
("abb","baabb",-2),("aaa","bbaaa",-2),("bab","bbbab",-2),
("bba","bbbba",-2),("abb","bbabb",-2)]
*Sticker> let (s,rho,a,r)=mGamma mp in a
[("aab","aab",0),("baa","baa",0),("aba","aba",0),("bbb","bbb",0),
("aaa","aa",0),("abb","ab",0),("bba","bb",0),("bab","ba",0),
("baa","b",0),("aab","a",0),("aba","a",0),("bbb","b",0)]
*Sticker> let (s,rho,a,r)=mGamma mp in rh
[('a','a'),('b','b')]
*StickerEx1> take 30 $ Sticker.language $ mGamma mp
["b","ab","ba","aab","baa","aba","bbb","aaab","abbb","babb",
 "bbab","aaba","abaa","baaa","bbba","aaaab","abbab","babab",
 "bbaab","aabaa","abaaa","baaaa","bbbaa","aaaba","abbba",
 "babba","bbaba","aabbb","ababb","baabb"]
```

```
*StickerEx1> let (s,rho,a,r)=mGamma m1 in a
[("a","a",0),("aba","aba",0),("abab","aba",0),("aaaa","a",0),
("aaab","a",0),("abaa","a",0),("bbab","b",0),("aaba","a",0),
("abba","a",0),("aabb","a",0),("abbb","a",0),("baaa","b",0),
("baab","b",0),("bbaa","b",0),("baba","b",0),("bbba","b",0),
("babb","b",0),("bbbb","b",0)]
*StickerEx1> take 4 $ Sticker.language $ mGamma m1
["a","aba","ababa","abababa"]
```

```
mprr = let (s,rho,a,r)=mGamma mp in (map snd r)
mpa = let (s,rho,a,r)=mGamma mp in a
mprho = let (s,rho,a,r)=mGamma mp in rho
mplanguage = take 30 $ Sticker.language $ mGamma mp
m1rr = let (s,rho,a,r)=mGamma m1 in (map snd r)
m1a = let (s,rho,a,r)=mGamma m1 in a
m1rho = let (s,rho,a,r)=mGamma m1 in rho
```

```
m1language = take 4 $ Sticker.language $ mGamma m1
```

4 Grammar Module

4.1 Grammar

In this section, we define some types and functions about elementary definitions of formal language theories.

```
module Grammar where

import Data.List
import Automaton

type Rule = (Symbol, SymbolString)
type RuleSet = [Rule]
type StartSymbol = Symbol
type TerminalSymbol = SymbolSet
type NonTerminalSymbol = SymbolSet
type Grammar = (TerminalSymbol, NonTerminalSymbol, RuleSet, StartSymbol)
```

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.

Let `s`=T. For a string `w` in $(T \cup N)^*$, `(terminalQ s w)` is **True** if `w` $\in T^*$.

```
terminalQ::TerminalSymbol->SymbolString->Bool
terminalQ _  []   = True
terminalQ s (h:t) | (h `elem` s) = terminalQ s t
                  | otherwise    = False
```

Let `r` be a rule in R and `w` a string over $T \cup N$. The set $\{w' | w \Rightarrow_{\{r\}} w'\}$ is computed by `(gonestep r w)`.

```
onestep::Rule->SymbolString->[SymbolString]
onestep (l,r) []  =  []
onestep (l,r) s@(h:t)|  (h == l)  = nub (a++[r++t])
                     |  otherwise = a
        where a = nub $ (map ([h]++) (onestep (l,r) t)) \\ [s]

onestep'::RuleSet->SymbolString->[SymbolString]
onestep' rs s = nub $ concat [Grammar.onestep r s | r <- rs]
```

The language $L(G)$ generated by a grammar $G = (\Sigma, N, R, S)$ is defined as follows.

$$L(G) = \{w \in \Sigma^* | S \Rightarrow_G^* w\}$$

For a grammar `g`= G, `(Grammar.language g)` compute the $L(G)$.

```
language::Grammar->[SymbolString]
language g@(t,n,rs,s0) = filter (Grammar.terminalQ t)
                                (Grammar.generate g)

generate::Grammar->[SymbolString]
generate g@(t,n,rs,s0) =  (sstep (Grammar.onestep' rs)) [[s0]]

fgen::(SymbolString->Bool)->Grammar->Symbol->[SymbolString]
fgen f g@(t,n,rs,s0) s1 =  filter (Grammar.terminalQ t)
```

```
                          ((sfstep f (Grammar.onestep' rs)) [[s1]])

ffgen::(SymbolString->Bool)->Grammar->Symbol->Symbol->[SymbolString]
ffgen f g@(t,n,rs,s0) s1 s2 = filter (elem s2)
                           ((sfstep f (Grammar.onestep' rs)) [[s1]])
```

4.2 Linear Grammar

```
lincheck::Grammar->Bool
lincheck g@(t,n,rs,s0) = and $ map f rs
        where f (c,s) = lincheck' n s

lincheck'::NonTerminalSymbol->SymbolString->Bool
lincheck' n w = (length (filter (\x->(x `elem` n)) w)) <= 1

lpart::NonTerminalSymbol->SymbolString->SymbolString
lpart n []    = []
lpart n (h:t) | (h `elem` n)   = []
              | otherwise = [h]++(lpart n t)

rpart::NonTerminalSymbol->SymbolString->SymbolString
rpart n []    = []
rpart n (h:t) | (h `elem` n)   = t
              | otherwise = rpart n t

jpart::NonTerminalSymbol->SymbolString->Int
jpart n [] = 0
jpart n (h:t) | (h `elem` n) = ((head (elemIndices h n))+1)
              | otherwise = jpart n t
```

4.3 Examples

```
module GrammarEx where

import Data.List
import Automaton
import Grammar
```

Example 3 `gex1::Grammar`

```
gex1=(['a','b'],['S'],[('S',"aSb"),('S',"")],'S')
gex2::Grammar
gex2=(['a','b'],['S','A'],[('S',"A"),('S',"aSb"),('A',"aA"),('A',"")],'S')
gex3::Grammar
```

```
*GrammarExChar> gex1
("ab","S",[('S',"aSb"),('S',"")],'S')
*GrammarExChar> take 10 $ Grammar.language gex1
["","ab","aabb","aaabbb","aaaabbbb","aaaaabbbbb","aaaaaabbbbbb",
 "aaaaaaabbbbbbb","aaaaaaaabbbbbbbb","aaaaaaaaabbbbbbbbb"]
*GrammarExChar> gex2
("ab","SA",[('S',"A"),('S',"aSb"),('A',"aA"),('A',"")],'S')
*GrammarExChar> take 10 $ Grammar.language GrammarEx.gex2
["","a","ab","aa","aab","aabb","aaa","aaab","aaabb","aaabbb"]
```

```
t002::[SymbolString]
t002=take 10 $ Grammar.language GrammarEx.gex1
```

```
t004::[SymbolString]
t004=take 10 $ Grammar.language GrammarEx.gex2
gex3=(['0','1','(',')','+'],['S'],
      [('S',"(S+S)"),('S',"0"),('S',"1")],'S')
t006=take 10 $ Grammar.language GrammarEx.gex3

flanguage::(SymbolString->Bool)->Grammar->[SymbolString]
flanguage f g@(t,n,rs,s0) = filter (Grammar.terminalQ t) $
                                  (sfstep f (Grammar.onestep' rs)) [[s0]]
```

Example 4 *Examples using* `flanguage` *function.*

```
*GrammarEx> flanguage (\x->((length x)<20)) gex1
["","ab","aabb","aaabbb","aaaabbbb","aaaaabbbbb","aaaaaabbbbbb",
 "aaaaaaabbbbbbb","aaaaaaaabbbbbbbb","aaaaaaaaabbbbbbbbb"]
*GrammarEx> flanguage (\x->((length x)<10)) gex2
["","a","ab","aa","aab","aabb","aaa","aaab","aaabb","aaabbb",
 "aaaa","aaaab","aaaabb","aaaabbb","aaaabbbb","aaaaa","aaaaab",
 "aaaaabb","aaaaabbb","aaaaaa","aaaaaab","aaaaaabb","aaaaaaa",
 "aaaaaaab","aaaaaaaa"]
```

```
t007=flanguage (\x->((length x)<20)) GrammarEx.gex1
t008=flanguage (\x->((length x)<10)) GrammarEx.gex2
```

4.4 Sticker System vs Linear Grammar

Definition 5 *For a linear grammar $G = (N, \Sigma, S, P)$, sticker system $\gamma_G = (\sigma, \rho, A, R)$ is defined as follows.*

$$
\begin{aligned}
\rho &= \{(a,a) | a \in \Sigma\} \\
X_1 &= S \qquad (\textit{if } i = 1 \textit{ then } X_i = S) \\
T(i,k) &= \{w \in T^* | X_i \Rightarrow^* w, |w| = k\} \\
T(i,l,r) &= \{(w_l, j, w_r) \in (T^* \times \boldsymbol{N} \times T^*) | X_i \Rightarrow w_l X_j w_r, |w_l| = l, |w_r| = r\} \\
A &= A_1 \cup A_2 \cup A_3 \\
A_1 &= \{(x, x, 0) | x \in T(0,m), m \leq 3k+1\} \\
A_2 &= \{(ux, x, |u|) | w \in T(i,m), 1+m \leq m \leq 3k+1, \\
&\quad\; x = Right(w, m-i), u = Left(w,i)\} \\
A_3 &= \{(xu, x, 0) | w \in T(i,m), 1+m \leq m \leq 3k+1, \\
&\quad\; x = Left(w, m-i), u = Right(w,i)\} \\
R &= R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 \cup R_6 \\
R_1 &= \bigcup_{i=1}^{k} \bigcup_{l=0}^{k+1} \{((ux, xv, |u|), (z,z,0)) | (w,j,z) \in T(i, k+1, l), \\
&\quad\; u = Left(w,i), x = Right(w,i), v \in \Sigma^j\} \\
R_2 &= \bigcup_{i=1}^{k} \bigcup_{l=0}^{k+1} \{((x, xv, 0), (zu, z, 0)) | (x,j,w) \in T(i, l, k+1), \\
&\quad\; z = Left(w, k+1-i), u = Right(w,i), v \in \Sigma^j\} \\
R_3 &= \bigcup_{l=1}^{2k+1} \{((x, xv, 0), (z,z,0)) | (x,j,z) \in T(0,l,m), 1
\end{aligned}
$$

$$
\begin{aligned}
& 0 \leq m \leq 2k+1-l, v \in \Sigma^j\} \\
R_4 &= \bigcup_{i=1}^{k} \bigcup_{l=0}^{k+1} \{((z,z,0),(xu,vx,-|v|))|(z,j,w) \in T(i,l,k+1), \\
& x = Left(w,k+1-i), u = Right(w,i), v \in \Sigma^j\} \\
R_5 &= \bigcup_{i=1}^{k} \bigcup_{l=0}^{k+1} \{((uz,z,|u|),(x,vx,-|v|))|(w,j,z) \in T(i,k+1,l), \\
& u = Left(w,i), x = Right(w,k+1-i), v \in \Sigma^j\} \\
R_6 &= \bigcup_{l=1}^{2k+1} \{((z,z,0),(x,vx,-|v|))|(z,j,x) \in T(0,m,l), \\
& 0 \leq m \leq 2k+1-l, v \in \Sigma^j\} \\
k &= |N|
\end{aligned}
$$

Theorem 2 ([4](Theorem 8))

$$L(\gamma_G) = L(G)$$

■

```
module StickerEx2 where

import Data.List
import Automaton
import Sticker
import Grammar
import GrammarEx

tik::Grammar->Int->Int->[SymbolString]
tik g@(t,n,rs,s0) i k = [w| w<-(fgen m g (n!!(i-1))), (length w)==k]
      where m s = ((length s) <= k+1)

tilr::Grammar->Int->Int->Int->[(SymbolString,Int,SymbolString)]
tilr g@(t,n,rs,s0) i l r = concat [(tilr' g i l r j)|j<-[1..k]]
       where k = length n

tilr'::Grammar->Int->Int->Int->Int->[(SymbolString,Int,SymbolString)]
tilr' g@(t,n,rs,s0) i l r j = [(lpart n w,j,rpart n w)|
                  w<-(ffgen m g (n!!(i-1)) (n!!(j-1))),
                  and[((length (lpart n w))==l),(length (rpart n w))==r]]
      where m s = ((length s) <= (l+r+1))

gA::Grammar->[Domino]
gA s = (gA1 s) ++ (gA2 s) ++ (gA3 s)

gA1::Grammar->[Domino]
gA1 s@(t,n,rs,s0) = [(x,x,0)|m<-[1..(3*k+1)],x<-(tik s 1 m)]
          where k = (length n)

gA2'::Int->Grammar->[Domino]
gA2' i s@(t,n,rs,s0) = [(w,(drop i w),i)|m<-[1..(3*k+1)],w<-(tik s i m)]
          where k = length n

gA2::Grammar->[Domino]
gA2 g@(t,n,rs,s0) = concat [(gA2' i g)|i<-[1..(length n)]]
```

```
gA3'::Int->Grammar->[Domino]
gA3' i g@(t,n,rs,s0) = [(w,(take (m-i) w),0)|
                             m<-[1..(3*k+1)],w<-(tik g i m)]
          where k  = length n

gA3::Grammar->[Domino]
gA3 g@(t,n,rs,s0) = concat [(gA3' i g)|i<-[1..(length n)]]

gR::Grammar->[(Domino,Domino)]
gR g = concat [gR1 g, gR2 g, gR3 g, gR4 g, gR5 g, gR6 g]

gR1'::Int->Int->Grammar->[(Domino,Domino)]
gR1' i l g@(t,n,rs,s0) = [((w,(drop i w)++v,i),(z,z,0))|
         (w,j,z)<-(tilr g i (k+1) l), v<-(sigman t j)]
         where k = length n

gR1::Grammar->[(Domino,Domino)]
gR1 g@(t,n,rs,s0) = concat [(gR1' i l g)|i<-[1..k],l<-[0..(k+1)]]
          where k = length n

gR2'::Int->Int->Grammar->[(Domino,Domino)]
gR2' i l g@(t,n,rs,s0) = [((x,x++v,0),(w,(take i' w),0))|
          (x,j,w)<-(tilr g i l (k+1)), v<-(sigman t j)]
          where k = length n
                i' = k+1-i

gR2::Grammar->[(Domino,Domino)]
gR2 g@(t,n,rs,s0) = concat [(gR2' i l g)|i<-[1..k],l<-[0..(k+1)]]
          where k = length n

gR3'::Int->Int->Grammar->[(Domino,Domino)]
gR3' l m g@(t,n,rs,s0) = [((x,x++v,0),(z,z,0))|
          (x,j,z)<-(tilr g 1 l m), v<-(sigman t j)]
          where k = length n

gR3::Grammar->[(Domino,Domino)]
gR3 g@(t,n,rs,s0) = concat [(gR3' l m g)|l<-[1..(2*k+1)],m<-[0..(2*k+1-l)]]
          where k = length n

gR4'::Int->Int->Grammar->[(Domino,Domino)]
gR4' i l g@(t,n,rs,s0) = [((z,z,0),(w,v++(take i' w),-j))|
          (z,j,w)<-(tilr g i l (k+1)), v<-(sigman t j)]
          where k = length n
                i' = k+1-i

gR4::Grammar->[(Domino,Domino)]
gR4 g@(t,n,rs,s0) = concat [(gR4' i l g)|i<-[1..k],l<-[0..(k+1)]]
          where k = length n

gR5'::Int->Int->Grammar->[(Domino,Domino)]
gR5' i l g@(t,n,rs,s0) = [((w,(drop i w),i),
                          (x,v++x,-j))|
          (w,j,x)<-(tilr g 1 (k+1) l), v<-(sigman t j)]
          where k = length n

gR5::Grammar->[(Domino,Domino)]
gR5 g@(t,n,rs,s0) = concat [(gR5' i l g)|i<-[1..k],l<-[0..(k+1)]]
```

```
          where k = length n

gR6'::Int->Int->Grammar->[(Domino,Domino)]
gR6' l m g@(t,n,rs,s0) = [((z,z,0),(x,v++x,-j))|
          (z,j,x)<-(tilr g 1 m l), v<-(sigman t j)]
          where k = length n

gR6::Grammar->[(Domino,Domino)]
gR6 g@(t,n,rs,s0) = concat [(gR6' l m g)|l<-[1..(2*k+1)],m<-[0..(2*k+1-l)]]
          where k = length n

gGamma::Grammar->Sticker
gGamma g@(t,n,rs,s0) = (t,rho,(gA g),(gR g))
          where rho = map (\x->(x,x)) t
```

Example 5 *The followings are sticker systems induced by linear grammars defined in Example 3.*

```
*StickerEx2> let (t,n,rs,s)=gex1 in rs
[('S',"aSb"),('S',"")]
*StickerEx2> let (t,n,rs,s)=gex2 in rs
[('S',"A"),('S',"aSb"),('A',"aA"),('A',"")]
*StickerEx2> gA gex1
[("ab","ab",0),("aabb","aabb",0),("ab","b",1),("aabb","abb",1),
 ("ab","a",0),("aabb","aab",0)]
*StickerEx2>  gGamma gex1
("ab",[('a','a'),('b','b')],[("ab","ab",0),("aabb","aabb",0),
 ("ab","b",1),("aabb","abb",1),("ab","a",0),("aabb","aab",0)],
 [(("aa","aa",1),("bb","bb",0)),(("aa","ab",1),("bb","bb",0)),
 (("aa","aaa",0),("bb","b",0)),(("aa","aab",0),("bb","b",0)),
 (("a","aa",0),("b","b",0)),(("a","ab",0),("b","b",0)),
 (("aa","aa",0),("bb","ab",-1)),(("aa","aa",0),("bb","bb",-1)),
 (("aa","a",1),("bb","abb",-1)),(("aa","a",1),("bb","bbb",-1)),
 (("a","a",0),("b","ab",-1)),(("a","a",0),("b","bb",-1))])
*StickerEx2> take 10 $ Sticker.language $ gGamma gex1
["ab","aabb","aaabbb","aaaabbbb","aaaaabbbbb","aaaaaabbbbbb",
 "aaaaaaabbbbbbb","aaaaaaaabbbbbbbb","aaaaaaaaabbbbbbbbb",
 "aaaaaaaaaabbbbbbbbbb"]
*StickerEx2> take 20 $ Sticker.language $ gGamma gex2
["a","ab","aa","aab","aaa","aabb","aaab","aaaa","aaabb","aaaab",
 "aaaaa","aaabbb","aaaabb","aaaaab","aaaaaa","aaaabbb","aaaaabb",
 "aaaaaab","aaaaaaa","aaaabbbb"]
*StickerEx2> take 10 $ Sticker.generate $ gGamma gex1
[("ab","ab",0),("aabb","aabb",0),("ab","b",1),("aabb","abb",1),
 ("ab","a",0),("aabb","aab",0),("aaabbb","aabbb",1),
 ("aaaabbbb","aaabbbb",1),("aaabbb","aaabb",0),
 ("aaaabbbb","aaaabbb",0)]
```

```
gex1rs = let (t,n,rs,s)=gex1 in rs
gex2rs = let (t,n,rs,s)=gex2 in rs
t101 = gA gex1
t102 = gGamma gex1
t103 = take 10 $ Sticker.language $ gGamma gex1
t104 = take 20 $ Sticker.language $ gGamma gex2
t105 = take 10 $ Sticker.generate $ gGamma gex1
```

5 LaTeX Output Module

5.1 Code

```
module OutputEx where

import Data.List
import Automaton
import AutomatonEx
import Sticker
import StickerEx1
import Grammar
import GrammarEx
import StickerEx2
import System

st2file::Sticker->String->IO ()
st2file st file = writeFile file $ unlines $ st2tex st

r2file::[(Domino,Domino)]->String->IO ()
r2file rs file = writeFile file $ unlines $ concat $ map r2tex rs

a2file::[Domino]->String->IO ()
a2file rs file = writeFile file $ unlines $ concat $ map d2tex rs

d2s::Domino->Domino
d2s (x,y,k) | (lx < ly) = ((spapnd (ly-lx) ' ' x'), y', 0)
            | otherwise = (x', (spapnd (lx-ly) ' ' y'), 0)
             where lx = length x'
                   ly = length y'
                   (x',y') = f (x,y,k)
                   f (x,y,n) | (n < 0)   = ((spinst (-n) ' ' x),y)
                             | otherwise = (x, (spinst n ' ' y))

spinst::Eq a=>Int->a->[a]->[a]
spinst 0 c l = l
spinst k c l | (k <= 0) = l
             | otherwise = (c:(spinst (k-1) c l))

sps::Eq a=>Int->a->[a]
sps k c | (k <= 0)  = []
        | otherwise = (c:(sps (k-1) c))

spapnd::Eq a=>Int->a->[a]->[a]
spapnd k c [] = sps k c
spapnd k c (h:t) = (h:(spapnd k c t))

s2tex::(Show a)=>[a]->String
s2tex []    = ""
s2tex (h:t) = (show h)++" & "++(s2tex t)

i2tex::[Char]->String
i2tex [] = ""
i2tex (h:t) | (h == ' ')  = "\\ "++(i2tex t)
            | otherwise = (h:(i2tex t))

d2tex::Domino->[String]
d2tex ([],[],_) = ["\\begin{tabular}{|l|} \\hline",
```

```
                   "\\ \\\\ \\ \\\\ \\hline",
                   "\\end{tabular} "]
d2tex d = ["\\begin{tabular}{|l|} \\hline",
           (i2tex x)++" \\\\", (i2tex y)++" \\\\",
                   "\\hline","\\end{tabular} "]
          where (x,y,k) = d2s d

r2tex::(Domino,Domino)->[String]
r2tex (dx,dy) = [ " ( "]++(d2tex dx)
                  ++[" , "]++(d2tex dy)++[") "]

st2tex::Sticker->[String]
st2tex (s,rho,a,d) = [ "\\begin{description}",
                       "\\item[A] "        ]
                     ++  concat (map d2tex a)
                     ++ [ "\\item[R] "     ]
                     ++  concat (map r2tex d)
                     ++ [ "\\end{description}",""]
```

Example 6 *The followings are command lists to make prety printed domino output files using in this article.*

```
t301=st2file (gGamma gex1) "gex1.tex"
t302=st2file (gGamma gex2) "gex2.tex"
t303=a2file (gA1 gex1) "gex1a1.tex"
t304=a2file (gA2 gex1) "gex1a2.tex"
t305=a2file (gA3 gex1) "gex1a3.tex"
t306=r2file (gR1 gex1) "gex1r1.tex"
t307=r2file (gR2 gex1) "gex1r2.tex"
t308=r2file (gR3 gex1) "gex1r3.tex"
t309=r2file (gR4 gex1) "gex1r4.tex"
t310=r2file (gR5 gex1) "gex1r5.tex"
t311=r2file (gR6 gex1) "gex1r6.tex"
t312=a2file (gA1 gex2) "gex2a1.tex"
t313=a2file (gA2 gex2) "gex2a2.tex"
t314=a2file (gA3 gex2) "gex2a3.tex"
t315=r2file (gR1 gex2) "gex2r1.tex"
t316=r2file (gR2 gex2) "gex2r2.tex"
t317=r2file (gR3 gex2) "gex2r3.tex"
t318=r2file (gR4 gex2) "gex2r4.tex"
t319=r2file (gR5 gex2) "gex2r5.tex"
t320=r2file (gR6 gex2) "gex2r6.tex"
t321=a2file (aA1 mp) "mpa1.tex"
t322=a2file (aA2 mp) "mpa2.tex"
t323=a2file (dD mp)  "mpd.tex"
t324=a2file (dF mp)  "mpf.tex"
```

5.2 Examples

Example 7 *For an automaton $M_p = (\{0,1\}, \Sigma, \delta, 0, \{1\})$ in Example 1, we have the sticker system $gamma_{M_p} = (\Sigma, \rho, A, R)$.*

$$
\begin{aligned}
\gamma_{G_1} &= (\Sigma, \rho, A, R) \\
\rho &= \{(a,a), (b,b)\} \\
A &= A_1 \cup A_2
\end{aligned}
$$

(A1)

b	ab	ba	aab	baa	aba	bbb
b	ab	ba	aab	baa	aba	bbb

(A2) $\binom{aaa}{aa}$ $\binom{bba}{bb}$ $\binom{abb}{ab}$ $\binom{bab}{ba}$ $\binom{baa}{b}$ $\binom{aab}{a}$ $\binom{aba}{a}$ $\binom{bbb}{b}$

$$R \quad = \quad D \cup F$$

(D) $\binom{aaa}{aaa}$ $\binom{bba}{abb}$ $\binom{aaa}{baa}$ $\binom{bba}{bbb}$ $\binom{abb}{aab}$ $\binom{bab}{aba}$ $\binom{abb}{bab}$ $\binom{bab}{bba}$ $\binom{aba}{aaab}$ $\binom{baa}{aaba}$ $\binom{aba}{abab}$ $\binom{baa}{abba}$
$\binom{aba}{baab}$ $\binom{baa}{baba}$ $\binom{aba}{bbab}$ $\binom{baa}{bbba}$ $\binom{aab}{aaaa}$ $\binom{bbb}{aabb}$ $\binom{aab}{abaa}$ $\binom{bbb}{abbb}$ $\binom{aab}{baaa}$ $\binom{bbb}{babb}$ $\binom{aab}{bbaa}$
$\binom{bbb}{bbbb}$ $\binom{baa}{ab}$ $\binom{baa}{bb}$ $\binom{aab}{aa}$ $\binom{aab}{ba}$ $\binom{aba}{aa}$ $\binom{aba}{ba}$ $\binom{bbb}{ab}$ $\binom{bbb}{bb}$ $\binom{aaa}{aaa}$ $\binom{aaa}{aba}$ $\binom{aaa}{baa}$
$\binom{aaa}{bba}$ $\binom{bab}{aab}$ $\binom{bab}{abb}$ $\binom{bab}{bab}$ $\binom{bab}{bbb}$ $\binom{bba}{aab}$ $\binom{bba}{abb}$ $\binom{bba}{bab}$ $\binom{bba}{bbb}$ $\binom{abb}{aaa}$ $\binom{abb}{aba}$
$\binom{abb}{baa}$ $\binom{abb}{bba}$

(F) $\binom{b}{ab}$ $\binom{b}{bb}$ $\binom{a}{aaa}$ $\binom{a}{aba}$ $\binom{a}{baa}$ $\binom{a}{bba}$ $\binom{ab}{aab}$ $\binom{ba}{aba}$ $\binom{ab}{bab}$ $\binom{ba}{bba}$ $\binom{aa}{aaaa}$ $\binom{bb}{aabb}$ $\binom{aa}{abaa}$ $\binom{bb}{abbb}$
$\binom{aa}{baaa}$ $\binom{bb}{babb}$ $\binom{aa}{bbaa}$ $\binom{bb}{bbbb}$ $\binom{aab}{aaab}$ $\binom{baa}{abaa}$ $\binom{aba}{aaba}$ $\binom{bbb}{abbb}$ $\binom{aab}{baab}$ $\binom{baa}{bbaa}$ $\binom{aba}{baba}$ $\binom{bbb}{bbbb}$
$\binom{aaa}{aaaaa}$ $\binom{bab}{aabab}$ $\binom{bba}{aabba}$ $\binom{abb}{aaabb}$ $\binom{aaa}{abaaa}$ $\binom{bab}{abbab}$ $\binom{bba}{abbba}$ $\binom{abb}{ababb}$ $\binom{aaa}{baaaa}$ $\binom{bab}{babab}$ $\binom{bba}{babba}$
$\binom{abb}{baabb}$ $\binom{aaa}{bbaaa}$ $\binom{bab}{bbbab}$ $\binom{bba}{bbbba}$ $\binom{abb}{bbabb}$

Example *8 For a linear grammar $G_1 = \{\{S\}, \{a, b\}, S, \{S \to \epsilon, S \to aSb\}\}$, we have the sticker system $\gamma_{G_1} = (\Sigma, \rho, A, R)$.*

$$\begin{aligned} \gamma_{G_1} &= (\Sigma, \rho, A, R) \\ \rho &= \{(a,a),(b,b)\} \\ A &= A_1 \cup A_2 \cup A_3 \end{aligned}$$

(A1) $\binom{ab}{ab}$ $\binom{aabb}{aabb}$

(A2) $\binom{ab}{b}$ $\binom{aabb}{abb}$

(A3) $\binom{ab}{a}$ $\binom{aabb}{aab}$

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

(R1) ($\binom{aa}{aa}$, $\binom{bb}{bb}$) ($\binom{aa}{ab}$, $\binom{bb}{bb}$)

(R2) ($\binom{aa}{aaa}$, $\binom{bb}{b}$) ($\binom{aa}{aab}$, $\binom{bb}{b}$)

(R3) ($\binom{a}{aa}$, $\binom{b}{b}$) ($\binom{a}{ab}$, $\binom{b}{b}$)

(R4) ($\binom{aa}{aa}$, $\binom{bb}{ab}$) ($\binom{aa}{aa}$, $\binom{bb}{bb}$)

(R5) ($\binom{aa}{a}$, $\binom{bb}{abb}$) ($\binom{aa}{a}$, $\binom{bb}{bbb}$)

(R6) $\left(\left[{a \atop a}\right], \left[{b \atop ab}\right]\right) \left(\left[{a \atop a}\right], \left[{b \atop bb}\right]\right)$

Example 9 *For a linear grammar* $G_2 = \{\{S, A\}, \{a, b\}, S, \{S \to aSb, S \to A, A \to \epsilon, A \to aA\}\}$, *we have the sticker system* $\gamma_{G_2} = (\Sigma, \rho, A, R)$.

$$\begin{aligned} \gamma_{G_2} &= (\Sigma, \rho, A, R) \\ \rho &= \{(1,1), (2,2)\} \\ A &= A_1 \cup A_2 \cup A_3 \end{aligned}$$

(A1) $\left[{a \atop a}\right]$ $\left[{ab \atop ab}\right]$ $\left[{aa \atop aa}\right]$ $\left[{aab \atop aab}\right]$ $\left[{aaa \atop aaa}\right]$ $\left[{aabb \atop aabb}\right]$ $\left[{aaab \atop aaab}\right]$ $\left[{aaaa \atop aaaa}\right]$ $\left[{aaabb \atop aaabb}\right]$ $\left[{aaaab \atop aaaab}\right]$ $\left[{aaaaa \atop aaaaa}\right]$ $\left[{aaabbb \atop aaabbb}\right]$ $\left[{aaaabb \atop aaaabb}\right]$ $\left[{aaaaab \atop aaaaab}\right]$ $\left[{aaaaaa \atop aaaaaa}\right]$ $\left[{aaaabbb \atop aaaabbb}\right]$ $\left[{aaaaabb \atop aaaaabb}\right]$ $\left[{aaaaaab \atop aaaaaab}\right]$ $\left[{aaaaaaa \atop aaaaaaa}\right]$

(A2) $\left[{a \atop \ }\right]$ $\left[{ab \atop b}\right]$ $\left[{aa \atop a}\right]$ $\left[{aab \atop ab}\right]$ $\left[{aaa \atop aa}\right]$ $\left[{aabb \atop abb}\right]$ $\left[{aaab \atop aab}\right]$ $\left[{aaaa \atop aaa}\right]$ $\left[{aaabb \atop aabb}\right]$ $\left[{aaaab \atop aaab}\right]$ $\left[{aaaaa \atop aaaa}\right]$ $\left[{aaabbb \atop aabbb}\right]$ $\left[{aaaabb \atop aaabb}\right]$ $\left[{aaaaab \atop aaaab}\right]$ $\left[{aaaaaa \atop aaaaa}\right]$ $\left[{aaaabbb \atop aaabbb}\right]$ $\left[{aaaaabb \atop aaaabb}\right]$ $\left[{aaaaaab \atop aaaaab}\right]$ $\left[{aaaaaaa \atop aaaaaa}\right]$ $\left[{a \atop \ }\right]$ $\left[{aa \atop \ }\right]$ $\left[{aaa \atop a}\right]$ $\left[{aaaa \atop aa}\right]$ $\left[{aaaaa \atop aaa}\right]$ $\left[{aaaaaa \atop aaaa}\right]$ $\left[{aaaaaaa \atop aaaaa}\right]$

(A3) $\left[{a \atop \ }\right]$ $\left[{ab \atop a}\right]$ $\left[{aa \atop a}\right]$ $\left[{aab \atop aa}\right]$ $\left[{aaa \atop aa}\right]$ $\left[{aabb \atop aab}\right]$ $\left[{aaab \atop aaa}\right]$ $\left[{aaaa \atop aaa}\right]$ $\left[{aaabb \atop aaab}\right]$ $\left[{aaaab \atop aaaa}\right]$ $\left[{aaaaa \atop aaaa}\right]$ $\left[{aaabbb \atop aaabb}\right]$ $\left[{aaaabb \atop aaaab}\right]$ $\left[{aaaaab \atop aaaaa}\right]$ $\left[{aaaaaa \atop aaaaa}\right]$ $\left[{aaaabbb \atop aaaabb}\right]$ $\left[{aaaaabb \atop aaaaab}\right]$ $\left[{aaaaaab \atop aaaaaa}\right]$ $\left[{aaaaaaa \atop aaaaaa}\right]$ $\left[{a \atop \ }\right]$ $\left[{aa \atop \ }\right]$ $\left[{aaa \atop a}\right]$ $\left[{aaaa \atop aa}\right]$ $\left[{aaaaa \atop aaa}\right]$ $\left[{aaaaaa \atop aaaa}\right]$ $\left[{aaaaaaa \atop aaaaa}\right]$

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

(R1) $\left(\left[{aaa \atop aaaa}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop aaab}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop aaba}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop aabb}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop aaaa}\right], \left[{b \atop b}\right]\right) \left(\left[{aaa \atop aaab}\right], \left[{b \atop b}\right]\right) \left(\left[{aaa \atop aaba}\right], \left[{b \atop b}\right]\right) \left(\left[{aaa \atop aabb}\right], \left[{b \atop b}\right]\right) \left(\left[{aaa \atop aaaa}\right], \left[{bb \atop bb}\right]\right) \left(\left[{aaa \atop aaab}\right], \left[{bb \atop bb}\right]\right) \left(\left[{aaa \atop aaba}\right], \left[{bb \atop bb}\right]\right) \left(\left[{aaa \atop aabb}\right], \left[{bb \atop bb}\right]\right) \left(\left[{aaa \atop aaa}\right], \left[{bbb \atop bbb}\right]\right) \left(\left[{aaa \atop aab}\right], \left[{bbb \atop bbb}\right]\right) \left(\left[{aaa \atop aaaa}\right], \left[{bbb \atop bbb}\right]\right) \left(\left[{aaa \atop aaab}\right], \left[{bbb \atop bbb}\right]\right) \left(\left[{aaa \atop aaba}\right], \left[{bbb \atop bbb}\right]\right) \left(\left[{aaa \atop aabb}\right], \left[{bbb \atop bbb}\right]\right) \left(\left[{aaa \atop aaa}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop aab}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop aba}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aaa \atop abb}\right], \left[{\ \atop \ }\right]\right)$

(R2) $\left(\left[{aaa \atop aaaa}\right], \left[{bbb \atop bb}\right]\right) \left(\left[{aaa \atop aaab}\right], \left[{bbb \atop bb}\right]\right) \left(\left[{aaa \atop aaaaa}\right], \left[{bbb \atop bb}\right]\right) \left(\left[{aaa \atop aaaab}\right], \left[{bbb \atop bb}\right]\right) \left(\left[{aaa \atop aaaba}\right], \left[{bbb \atop bb}\right]\right) \left(\left[{aaa \atop aaabb}\right], \left[{bbb \atop bb}\right]\right)$

(R3) $\left(\left[{a \atop aaa}\right], \left[{\ \atop \ }\right]\right) \left(\left[{a \atop aab}\right], \left[{\ \atop \ }\right]\right) \left(\left[{a \atop aba}\right], \left[{\ \atop \ }\right]\right) \left(\left[{a \atop abb}\right], \left[{\ \atop \ }\right]\right) \left(\left[{a \atop aa}\right], \left[{b \atop b}\right]\right) \left(\left[{a \atop ab}\right], \left[{b \atop b}\right]\right) \left(\left[{a \atop aaa}\right], \left[{b \atop b}\right]\right) \left(\left[{a \atop aab}\right], \left[{b \atop b}\right]\right) \left(\left[{a \atop aba}\right], \left[{b \atop b}\right]\right) \left(\left[{a \atop abb}\right], \left[{b \atop b}\right]\right) \left(\left[{aa \atop aaaa}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aa \atop aaab}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aa \atop aaba}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aa \atop aabb}\right], \left[{\ \atop \ }\right]\right) \left(\left[{aa \atop aaaa}\right], \left[{b \atop b}\right]\right) \left(\left[{aa \atop aaab}\right], \left[{b \atop b}\right]\right) \left(\left[{aa \atop aaba}\right], \left[{b \atop b}\right]\right) \left(\left[{aa \atop aabb}\right], \left[{b \atop b}\right]\right) \left(\left[{aa \atop aaa}\right], \left[{bb \atop bb}\right]\right) \left(\left[{aa \atop aab}\right], \left[{bb \atop bb}\right]\right) \left(\left[{aa \atop aaaa}\right], \left[{bb \atop bb}\right]\right)$

$) (\binom{aa}{aaab} , \binom{bb}{bb}) (\binom{aa}{aaba} , \binom{bb}{bb}) (\binom{aa}{aabb} , \binom{bb}{bb}) (\binom{aaa}{aaaaa} , \binom{\ }{\ }) (\binom{aaa}{aaaab}$
$, \binom{\ }{\ }) (\binom{aaa}{aaaba} , \binom{\ }{\ }) (\binom{aaa}{aaabb} , \binom{\ }{\ }) (\binom{aaa}{aaaaa} , \binom{b}{b}) (\binom{aaa}{aaaab} , \binom{b}{b}) (\binom{aaa}{aaaba}$
$, \binom{b}{b}) (\binom{aaa}{aaabb} , \binom{b}{b}) (\binom{aaa}{aaaaa} , \binom{bb}{bb}) (\binom{aaa}{aaaab} , \binom{bb}{bb}) (\binom{aaa}{aaaba} , \binom{bb}{bb})$
$(\binom{aaa}{aaabb} , \binom{bb}{bb}) (\binom{aaaa}{aaaaaa} , \binom{\ }{\ }) (\binom{aaaa}{aaaaab} , \binom{\ }{\ }) (\binom{aaaa}{aaaaba} , \binom{\ }{\ }) (\binom{aaaa}{aaaabb}$
$, \binom{\ }{\ }) (\binom{aaaa}{aaaaaa} , \binom{b}{b}) (\binom{aaaa}{aaaaab} , \binom{b}{b}) (\binom{aaaa}{aaaaba} , \binom{b}{b}) (\binom{aaaa}{aaaabb} , \binom{b}{b})$
$(\binom{aaaaa}{aaaaaaa} , \binom{\ }{\ }) (\binom{aaaaa}{aaaaaab} , \binom{\ }{\ }) (\binom{aaaaa}{aaaaaba} , \binom{\ }{\ }) (\binom{aaaaa}{aaaaabb} , \binom{\ }{\ })$

(R4) $(\binom{aaa}{aaa} , \binom{bbb}{abb}) (\binom{aaa}{aaa} , \binom{bbb}{bbb}) (\binom{aaa}{aaa} , \binom{bbb}{aabb}) (\binom{aaa}{aaa} , \binom{bbb}{abbb}) (\binom{aaa}{aaa}$
$, \binom{bbb}{babb}) (\binom{aaa}{aaa} , \binom{bbb}{bbbb})$

(R5) $(\binom{aaa}{aa} , \binom{\ }{aa}) (\binom{aaa}{aa} , \binom{\ }{ab}) (\binom{aaa}{aa} , \binom{\ }{ba}) (\binom{aaa}{aa} , \binom{\ }{bb}) (\binom{aaa}{aa} , \binom{b}{aab}$
$) (\binom{aaa}{aa} , \binom{b}{abb}) (\binom{aaa}{aa} , \binom{b}{bab}) (\binom{aaa}{aa} , \binom{b}{bbb}) (\binom{aaa}{aa} , \binom{bb}{aabb}) (\binom{aaa}{aa}$
$, \binom{bb}{abbb}) (\binom{aaa}{aa} , \binom{bb}{babb}) (\binom{aaa}{aa} , \binom{bb}{bbbb}) (\binom{aaa}{aa} , \binom{bbb}{abbb}) (\binom{aaa}{aa} , \binom{bbb}{bbbb}$
$) (\binom{aaa}{aa} , \binom{bbb}{aabbb}) (\binom{aaa}{aa} , \binom{bbb}{abbbb}) (\binom{aaa}{aa} , \binom{bbb}{babbb}) (\binom{aaa}{aa} , \binom{bbb}{bbbbb}) ($
$\binom{aaa}{a} , \binom{\ }{aa}) (\binom{aaa}{a} , \binom{\ }{ab}) (\binom{aaa}{a} , \binom{\ }{ba}) (\binom{aaa}{a} , \binom{\ }{bb}) (\binom{aaa}{a} , \binom{b}{aab}$
$) (\binom{aaa}{a} , \binom{b}{abb}) (\binom{aaa}{a} , \binom{b}{bab}) (\binom{aaa}{a} , \binom{b}{bbb}) (\binom{aaa}{a} , \binom{bb}{aabb}) (\binom{aaa}{a}$
$, \binom{bb}{abbb}) (\binom{aaa}{a} , \binom{bb}{babb}) (\binom{aaa}{a} , \binom{bb}{bbbb}) (\binom{aaa}{a} , \binom{bbb}{abbb}) (\binom{aaa}{a} , \binom{bbb}{bbbb}$
$) (\binom{aaa}{a} , \binom{bbb}{aabbb}) (\binom{aaa}{a} , \binom{bbb}{abbbb}) (\binom{aaa}{a} , \binom{bbb}{babbb}) (\binom{aaa}{a} , \binom{bbb}{bbbbb})$

(R6) $(\binom{a}{a} , \binom{b}{ab}) (\binom{a}{a} , \binom{b}{bb}) (\binom{a}{a} , \binom{b}{aab}) (\binom{a}{a} , \binom{b}{abb}) (\binom{a}{a} , \binom{b}{bab}) (\binom{a}{a}$
$, \binom{b}{bbb}) (\binom{aa}{aa} , \binom{b}{aab}) (\binom{aa}{aa} , \binom{b}{abb}) (\binom{aa}{aa} , \binom{b}{bab}) (\binom{aa}{aa} , \binom{b}{bbb}) (\binom{aaa}{aaa}$
$, \binom{b}{aab}) (\binom{aaa}{aaa} , \binom{b}{abb}) (\binom{aaa}{aaa} , \binom{b}{bab}) (\binom{aaa}{aaa} , \binom{b}{bbb}) (\binom{aaaa}{aaaa} , \binom{b}{aab})$
$(\binom{aaaa}{aaaa} , \binom{b}{abb}) (\binom{aaaa}{aaaa} , \binom{b}{bab}) (\binom{aaaa}{aaaa} , \binom{b}{bbb}) (\binom{aa}{aa} , \binom{bb}{abb}) (\binom{aa}{aa} ,$
$\binom{bb}{bbb}) (\binom{aa}{aa} , \binom{bb}{aabb}) (\binom{aa}{aa} , \binom{bb}{abbb}) (\binom{aa}{aa} , \binom{bb}{babb}) (\binom{aa}{aa} , \binom{bb}{bbbb}) ($
$\binom{aaa}{aaa} , \binom{bb}{aabb}) (\binom{aaa}{aaa} , \binom{bb}{abbb}) (\binom{aaa}{aaa} , \binom{bb}{babb}) (\binom{aaa}{aaa} , \binom{bb}{bbbb})$

References

[1] A.Alhazov,M.Cavaliere, Computing by Observing Bio-systems:The Case of Sticker Systems,LNCS 3384,2005: 1-13,Proc.DNA10

[2] L.Kari, G.Paun, G.Rozenberg, A.Salomaa,S.Yu,DNA computing, sticker systems, and universality,Acta Informatica 35,1998: 401-420

[3] Y.Sakakibara,S.Kobayashi, Sticker systems with complex structures, Soft Computing 5 2001: 114-120

[4] G.Paun,G.Rozenberg, Sticker systems, Theoretical Computer Science 204,1998: 183-203

[5] K.K.K.R.Perera, Y.Mizoguchi, Imprementation of Haskell Modules for Automata ans Sticker Systems, Journal of Math-for-industory, 1,2009A-7: 51-56.