Graph splicing systems

Rahul Santhanam, Kamala Krithivasan

Department of Computer Science and Engineering, Indian Institute of Technology, Madras, 600 036, India

Received 10 August 1998; received in revised form 5 December 2002; accepted 14 November 2005
Available online 28 February 2006

Abstract

In this paper, extended graph splicing systems are defined. It is shown that when strings are represented as linear graphs, any recursively enumerable set can be generated by an extended graph splicing system. It is also shown that the computational completeness of extended graph splicing systems can be proved under some constraints too.

© 2006 Published by Elsevier B.V.

Keywords: Extended graph splicing systems; Computational completeness; Constrained splicing systems

1. Introduction

Splicing systems were introduced by Head in [5] to investigate the power of computing with DNA [8]. Some decidability results about splicing systems were studied in [2]. One of the recent developments in this field is the generalization of the concept of “splicing system” to graphs by Rudolf Freund in [3]. Freund’s formulation is interesting because it is quite an accurate model of a system consisting of biochemical units and the interactions between them. In this paper, we present results concerning the generative capacity of certain versions of the splicing system defined in [3]. We prove that, if a distinction between terminals and nonterminals is made in the system, then these graph splicing systems with finite sets of axioms and rules are computationally complete in a certain sense. If we consider a graph language consisting only of directed paths to be equivalent to the corresponding string language, then these systems can generate all recursively enumerable sets of strings.

Graph splicing systems have various parameters like number of axioms, number of rules etc. We examine, in this paper, the effect of constraints upon these parameters on the computational completeness of splicing systems. We find the minimum values of these parameters for which the resulting (constrained) class of systems is still computationally complete. These results have practical significance since the values of the parameters will be bounded in practical biochemical situations. The completeness of the splicing systems even for small bounds on the parameters provides a good indication for the feasibility of practical bio-computers.

The layout of the paper is as follows. In Sections 2 and 3 we give a few concepts from the literature which are required for the understanding of the paper. In Section 4, we prove the computational completeness of graph splicing systems. In Section 5, we study the effect of imposing some constraints on graph splicing systems in terms of computational power. We conclude the paper in Section 6.
2. Preliminaries

Let $\Sigma$ be any finite alphabet. $\Sigma^*$ denotes the set of all strings over $\Sigma$. A graph $G$ over $\Sigma$ is a triple $(V, E, f)$ where $V$ is the node set, $E \subseteq V \times V$ is the edge set and $f$ is the vertex labelling function ($f: V \rightarrow \Sigma$). Given a graph $G$, $V(G)$ denotes the vertex set of $G$ and $E(G)$ the edge set of $G$. A path in $G$ between vertices $v_0$ and $v_n$ belonging to $V(G)$ is a set of vertices $v_0, v_1, \ldots, v_n$ such that there is an edge between $v_i$ and $v_{i+1}$ for $i = 0, \ldots, n - 1$. A graph $G$ is said to be connected if there is a path in $G$ between every pair of vertices that belong to $V(G)$. A graph language is a set of labelled graphs.

The graphs, $G_1 = (V_1, E_1, f_1)$ and $G_2 = (V_2, E_2, f_2)$, are said to be equivalent if there is a bijective function $g: V_1 \rightarrow V_2$ such that,

- for $x, y \in V_1$, $(x, y) \in E_1$ iff $(g(x), g(y)) \in E_2$,
- for all $x \in V_1$, $f_1(x) = f_2(g(x))$.

The above relation induces an equivalence relation creating equivalence classes on the set $U$ of all graphs. Each such equivalence class is called an abstract graph. The abstract graph that corresponds to a graph $G$ is represented by $[G]$. When we consider graph languages, we mean languages of abstract graphs. When we deal with unlabelled graphs, this notion is called Isomorphism.

We now define the notion of a Turing machine [6]. A Turing machine is defined by the tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ where $Q$ is the set of states, $\Sigma$ is the input alphabet, $\Gamma$ is the set of tape symbols, $\delta$ is a mapping from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$, $q_0$ is the initial state, $B$ is a symbol in $\Gamma$ called the blank symbol and $F$ is the set of final states.

We denote an instantaneous description (ID) of the Turing machine by $z_1qz_2$. Here $q$, the current state of $M$, is in $Q$; $z_1$ and $z_2$ is the string in $\Gamma^*$ that is the contents of the tape. The tape head is assumed to be scanning the leftmost symbol of $z_2$, or if $z_2 = \varepsilon$, the head is scanning a blank symbol. We denote the moves of the Turing machine by the relation $\vdash$ defined on the IDs of the machine. The relation, $\vdash^*$, denotes the reflexive transitive closure of $\vdash$. We refer the reader to [6] for further details. The language accepted by a Turing machine, $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, denoted by $L(M)$, is defined as

$$L(M) = \{w | w \in \Sigma^* \text{ and } q_0w\vdash^*z_1pz2 \text{ for some } p \in F, \text{ and } z_1 \text{ and } z_2 \in \Gamma^*\}.$$ 

3. Definitions

**Definition 1 (Graph splicing system).** A graph splicing system $\sigma$ is a pair $(\Sigma, P)$ where $\Sigma$ is an alphabet and $P$ is a finite set of rules of the form, $((h[1], E'[1]), \ldots, (h[k], E'[k]); E)$, where

- $h[i] = (V[i], E[i], L[i])$, $1 \leq i \leq k$, is a graph ($L[i]$ is the vertex labelling function),
- $E'[i] \subseteq E[i]$, $1 \leq i \leq k$. $E'[i]$ is called the set of cut-edges;
- the node sets, $V[i]$, are mutually disjoint;
- $E$ must obey the following rules:
  - Each edge $(n, m) \in E'[i]$ is supposed to have been divided into two parts, i.e. the start part $(n, m)$ and the end part $[n, m]$ (refer to Fig. 1).
  - The elements of $E$ are pairs of the form $((n, m), [n', m'])$, where $(n, m)$ and $(n', m')$ are edges from $\bigcup E'[i]$.
  - Every element from $\{((n, m), [n, m]) | (n, m) \in \bigcup E'[i]\}$ must appear exactly once in a pair of $E$.

![Fig. 1. View of a cut-edge.](image-url)
Definition 2 (Graph splicing step). Let \( \sigma = (\Sigma, P) \) be a graph splicing system. Let \( p = (h[1], E'[1]), \ldots, (h[k], E'[k]); E \) be a rule in \( P \). We say that a set \( R \) of graphs derives a set \( S \) of graphs through the splicing rule \( p \) if there exist graphs \( g[1], g[2], \ldots, g[k] \in R \) and graphs \( g'[1], g'[2], \ldots, g'[m] \in S \) such that,

- For all \( i \) with \( 1 \leq i \leq k \), \( h[i] \) is an induced subgraph\(^1\) of \( g[i] \).
- From \( g[1], g[2], \ldots, g[k] \) we delete all edges corresponding to edges in \( \bigcup E'[i] \) but add each edge corresponding to the edge \((n, m')\) such that \((n, m')[n', m'] \in E \), which yields the uniquely determined union of \( m \) connected graphs \( g'[1], g'[2], \ldots, g'[m] \). \( \square \)

Here we are implicitly using multisets of graphs. \( g[j] \) may be another copy of \( g[i] \). The basic idea of a graph splicing rule is to cut various edges in some connected graphs and rejoin the edges in another fashion.

We give below the definition of an extended graph splicing system. It is slightly different from the definition given in [3].

Definition 3 (Extended graph splicing system). Let \( \sigma = (\Sigma, P) \) be a graph splicing system and \( I \) be a finite set of graphs, called the set of axioms. The quadruple, \( (N, T, P, I) \) where \( N \) and \( T \) are disjoint sets of nonterminals and terminals, respectively, and \( N \cup T = \Sigma \), is called an extended graph splicing system (in an ordinary graph splicing system, no distinction between terminals and nonterminals is made). \( \sigma(I) \) is the minimal set of graphs obtained by applying some splicing rules to some subset of \( I \). \( \sigma^n(I) \) is defined to be \( I \) if \( n = 0 \) and \( \sigma^n(I) \) is defined iteratively as \( \sigma(\sigma^{n-1}(I)) \). The graph language, \( L(F) \), generated by an extended graph splicing system \( F \), \( F = (N, T, P, I) \), is

\[
L(F) = \{ g = (V, E, f) \in \sigma^n(I) | f(n) \in T \forall n \in V(g) \}. \quad \square
\]

The form of the splicing rules in a graph splicing system places a restriction on the structure of the corresponding graph language. We define a graph language \( L \) to be of bounded degree if there is an integer \( k \) such that no vertex of any graph in \( L \) has degree greater than \( k \). For example, the set of all binary trees is a graph language of bounded degree while the set of all complete graphs is not. All graph splicing languages are graph languages of bounded degree. This is because the degrees of vertices of graphs participating in splicing are preserved in a graph splicing step. Thus, if \( S = (N, T, P, I) \) is an extended graph splicing system then,

Maximum degree of any vertex of a graph in \( L(S) \) is \( \max \{ \text{degree of } v | v \text{ is a vertex of a graph in } I \} \).

Since the axiom set \( I \) is finite, the maximum exists and is an integer that bounds the degree of any vertex of a graph in \( L \).

A graph language is said to be Turing machine representable if the set of effective representations of the graphs in the language is recursively enumerable (r.e.). Here “effective representation” means a representation that can be generated from the standard set representation of a graph by a Turing machine. The adjacency matrix of a graph, if considered in a linearized (row-major or column-major) form, is an example of such a representation. It is obvious that there are graph languages that are Turing machine representable but not degree-bounded, for example the set of complete graphs. A Turing machine could easily be designed that accepts all sequences of \( n^{2} \) 1’s for integer \( n \) and such a sequence is an effective representation of a complete graph. Thus the class of Turing machine representable graph languages does not coincide with the class of graph splicing languages. It is an open question whether the class of Turing machine representable graph languages of bounded degree is identical to the class of graph splicing languages.

We next consider a subclass of graph languages called linear graph languages. A linear graph language is a graph language that consists only of directed paths. The effective representation of a linear graph can be the string they represent. For example, the graph shown in Fig. 2 represents the string \( abc \cdots n \). Example 1 gives a graph splicing system which has all its graphs to be linear.

\(^1\)A graph \( H = (V', E') \) is an induced subgraph of a graph \( G, G = (V, E) \) if,

\[
\forall a, b \in V', \quad (a, b) \in E' \iff (a, b) \in E.
\]
Example 1. Consider the graph splicing system $\sigma$ defined as, $\sigma = (N, T, P, I)$ where the set of axioms, $I$, is given in Fig. 3. The set $P$ consists of the rule, $R = ((h[1], E'[1]), (h[2], E'[2]), E)$, where the graphs $h[1]$ and $h[2]$ are shown in Figs. 4 and 5, respectively. The cut-edges are shown in dotted lines in the figures.

The set $E$ is defined as

$$E = \{(2, 6), (7, 3), (5, 8)\},$$

where the labelling is as in the Figs. 4 and 5. The language generated by the system consists of the linear graphs representing the language, $\{a^n b^n | n \geq 1\}$.

In the next section, we prove that for every r.e. language $L$, there is a graph splicing system that generates the linear graph language representing the language $L$.

4. A completeness result

Computational completeness of extended regular H systems is provided in [7]. In this section, we prove the computational completeness of graph splicing systems. We prove that, for every r.e. language $L$, there is a graph splicing system that generates the linear graph language corresponding to $L$. The basic idea of the proof is to simulate the working of a one-tape Turing machine $M$ accepting $L$ using a graph splicing system. Since every r.e. set is accepted by some one-tape Turing machine, the simulation establishes the result.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, \{q_f\})$ be a Turing machine (without loss of generality, we can assume that there is only one final state). We construct an extended graph splicing system $F = (N, T, P, I)$ to simulate the working of $M$ as follows. We define, $V = \Gamma \cup Q \cup \{C, D, E, X, Y, Z\}$, $N = V \setminus \Sigma$ and $T = \Sigma$. The graphs in the set $I$ are given in the Fig. 6.

The prime idea behind the simulation of the working of the Turing machine on a string in $\Sigma^*$ is as follows. For each string in $\Sigma^*$, we generate a graph with two copies of the string, i.e., a graph of the form shown in Fig. 7. Then, we simulate a Turing machine on one of the copies of the string. If a halting state is reached, the part of the graph involved in the simulation is cut off and the linear graph representation of a string belonging to the r.e. set is generated.

We now describe the rule set $P$. The set $P$ consists of nine classes of rules. The classes are described in the Tables 1 and 2. Each class of rules are such that every rule in the class has exactly two graphs, $h[i]$ and $h[2]$. For the sake of brevity, we have given the graphs, $h[i]$’s and the resultant graphs obtained from these graphs after the splitting and recombining for each class of rules. We depict the cut-edges as dotted lines in the graphs $h[i]$’s.
Fig. 6. The graphs in the set \( I \). Here, \( a \in \Sigma, b, c \in \Gamma, q \in Q \). 

\[
\begin{align*}
C & \rightarrow Y \rightarrow D \rightarrow D \rightarrow D ; \\
C & \rightarrow b \rightarrow q \rightarrow D ; \\
C & \rightarrow q \rightarrow b \rightarrow c \rightarrow D ; \\
C & \rightarrow q \rightarrow D ; \\
C & \rightarrow a
\end{align*}
\]

Fig. 7. Graph generated during the simulation. Here the string is \( w = a_1 a_2 \cdots a_n \).

Table 1

<table>
<thead>
<tr>
<th>Classes of rules in ( P )</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Class 1:</th>
<th>In the figure, ( a, b \in \Sigma ).</th>
</tr>
</thead>
</table>
| \begin{align*}
X \rightarrow a & \rightarrow a \\
C \rightarrow D & \rightarrow E
\end{align*} |

<table>
<thead>
<tr>
<th>Class 2:</th>
<th>In the figure, ( a \in \Sigma ).</th>
</tr>
</thead>
</table>
| \begin{align*}
X \rightarrow a & \rightarrow a \\
C \rightarrow X & \rightarrow D
\end{align*} |

<table>
<thead>
<tr>
<th>Class 3:</th>
<th>In the figure, ( a \in \Sigma ).</th>
</tr>
</thead>
</table>
| \begin{align*}
Y \rightarrow a & \rightarrow a \\
C \rightarrow D
\end{align*} |

<table>
<thead>
<tr>
<th>Class 4:</th>
<th>In the figure, ( a \in \Sigma ).</th>
</tr>
</thead>
</table>
| \begin{align*}
Z \rightarrow a & \rightarrow a \\
C \rightarrow D
\end{align*} |

<table>
<thead>
<tr>
<th>Class 5:</th>
<th>In the figure, ( a, b, a' \in \Gamma, c, d \in N, p, q \in Q ) such that ( \delta(p, a) = (q, L, a') ).</th>
</tr>
</thead>
</table>
| \begin{align*}
C \rightarrow b & \rightarrow P \rightarrow a \rightarrow d \\
C \rightarrow a & \rightarrow a' \rightarrow d
\end{align*} |
Table 2
Classes of rules in $P$

| Class 6: In the figure, $a, d, a' \in \Gamma$, $b \in N$, $p, q \in Q$ such that $\delta(p, a) = (q, R, a')$. |
|---|---|
| Fig. 8. Linear graph of the ID before applying the rules in Class 5. |
| Fig. 9. Linear graph of the ID after applying the rules in Class 5. |
| Class 7: In the figure, $a, c \in N$, $b \in \Gamma$. |
| Class 8: In the figure, $a, c \in N$, $b \in \Gamma$. |
| Class 9: In the figure, $a, b \in \Sigma$. |

in Class 5 simulate a left move namely, $\delta(p, a) = (q, a', L)$. The corresponding portion of the ID in the linear graph as shown in Fig. 8 is changed accordingly to obtain the linear graph representation of the resultant ID as shown in Fig. 9 by splicing using these rules. Rules in class 6 simulate a right move, $\delta(p, a) = (q, a', R)$. The corresponding portion of the ID in the linear graph as shown in Fig. 10 is changed accordingly to obtain the linear graph representation of the resultant ID as shown in Fig. 11 by splicing using these rules. By this simulation, successive IDs appear between $Y$ and $Z$. Once the final state $q_f$ is reached, symbols next to $q_f$ are consumed by the rules in Classes 7 and 8 ending up with the linear graph shown in Fig. 12. The 9th class rules remove the $Y, q_f$ and $Z$ ending up with $w = a_1a_2\ldots a_n$ in the linear graph.
5. Results on constrained splicing systems

Consider a class $C$ of graph splicing systems with finite sets of axioms and rules. The family of graph languages, $\mathcal{L}(C)$, corresponding to $C$ is defined as

$$\mathcal{L}(C) = \{L(F) | F \in C\}.$$  

If $\mathcal{L}(C)$ contains the class of recursively enumerable languages represented as linear graphs, then $C$ is said to be computationally complete. We proved earlier that the class $U$ of all graph splicing systems with finite sets of axioms and rules is computationally complete.

Next we define the following terms:

$$\alpha(C) = \max \{|I| | F = (N, T, P, I) \in C \text{ for some } N, T, P\},$$

$$\beta(C) = \max \{|P| | F = (N, T, P, I) \in C \text{ for some } N, T, I\},$$

$$\gamma(C) = \max \{|N| | F = (N, T, P, I) \in C \text{ for some } T, P, I\},$$

$$\delta(C) = \max \{|k| | k \text{ graphs occur in a rule of } P \text{ for some } F = (N, T, P, I) \in C\},$$

$$\omega(C) = \max \{|k| | k \text{ edges are split in some rule of } P \text{ for some } F = (N, T, P, I) \in C\}.$$  

Most of the results in this section are obtained by slightly modifying the simulation of a Turing machine that yielded the earlier completeness result. Therefore, instead of giving the detailed simulation for each result, we specify the changes to be made to the original simulation.

**Theorem 1.** There is a set $C_1$ of graph splicing systems with $\alpha(C_1) = 1$ that is computationally complete.

**Proof.** In order to prove the theorem, we shall show that any r.e. language represented as linear graphs can be generated by a graph splicing system with a single axiom. We mainly use the simulation techniques used in the previous section. The basic idea of the proof is to encode many axioms by one axiom, from which the original axioms can be derived by the application of a special rule.

Let $L$ be a r.e. language. Then, by the result in the previous section, there exists an extended graph splicing system, $F = (N', T, P, I')$, which generates $L$ (represented as linear graphs). We construct a graph splicing system $F' = (N', T, P', I')$ from $F$ such that $L(F) = L(F')$ and $|I'| = 1$.

Let the axioms in $I$ be $I_1, I_2, \ldots, I_n$. Then $F'$ has $N' = N \cup \{c_1, c_2\}$ where $c_1$ and $c_2$ are symbols not in $N$. $I'$ consists of a single axiom which encodes the information in all the axioms $I_1, I_2, \ldots, I_n$ of $F$. Let us abuse notation by referring to the single graph in $I'$ as $I'$. Then, $I'$ is given in Fig. 13. The subgraphs $J_i$ refers to the graphs shown in Fig. 14. In the subgraph $J_i$, $a_i$ is the first symbol of $I_i$ and $a'_i$ is the last symbol of $I_i$, for each $i$. The symbol $a$ in $I'$ is any symbol in $N$. Each $J_i$ is attached transversely in $I'$, i.e., each $J_i$ is connected to its neighbours only through the node labelled $c_2$.

The rule set $P'$ of $F'$ is defined as,

$$P' = P \cup Q_1 \cup Q_2 \cup \cdots \cup Q_n.$$
Theorem 2. There does not exist a computationally complete set \( C_3 \) of graph splicing systems such that \( \gamma(C_3) = 0 \).

Proof. This result is given in [4]. It is obvious that languages such as the regular language \((aa)^*\) cannot be generated by a graph splicing system without nonterminals.

Theorem 3. There exists a set \( C_3 \) of graph splicing systems with \( \gamma(C_3) = 1 \) such that \( C_3 \) is computationally complete.
Fig. 16. Replacement subgraph for $N_i$.

Fig. 17. The graph $h[1]$ of $p'$.

Fig. 18. The graph $h[1]$ of $p$.

Fig. 19. Axiom set $I$. Here $a \in \Sigma$, and $b \in V$.

Table 3
Classes of rules in $P$

<table>
<thead>
<tr>
<th>Class</th>
<th>In the figure, $a, b \in \Sigma$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1:</td>
<td>$a \rightarrow X \cdots \rightarrow a$</td>
</tr>
<tr>
<td></td>
<td>$Z_1 \cdots b' \cdots Z_2$</td>
</tr>
<tr>
<td>Class 2:</td>
<td>$a \rightarrow X \rightarrow b' \rightarrow a$</td>
</tr>
<tr>
<td></td>
<td>$Z_1 \cdots b' \cdots Z_2$</td>
</tr>
<tr>
<td>Class 3:</td>
<td>$b' \rightarrow X \cdots \rightarrow b'$</td>
</tr>
<tr>
<td></td>
<td>$Z_1 \cdots b \cdots Z_2$</td>
</tr>
<tr>
<td>Class 4:</td>
<td>$b' \rightarrow X \rightarrow b \rightarrow b'$</td>
</tr>
<tr>
<td></td>
<td>$Z_1 \cdots b \cdots Z_2$</td>
</tr>
<tr>
<td>Class 5:</td>
<td>$b \rightarrow \cdots \rightarrow a$</td>
</tr>
<tr>
<td></td>
<td>$Z_1 \rightarrow Z_2$</td>
</tr>
</tbody>
</table>
### Table 4

#### Classes of rules in \( P \)

**Class 6:** In the figure, \( b \in \Sigma \).

![Figure 1](#)

**Class 7:** In the figure, \( a \in \Sigma \).

![Figure 2](#)

**Class 8:** In the figure, \( b \in \Sigma \).

![Figure 3](#)

**Class 9:**

![Figure 4](#)

**Class 10:** In the figure, \( a \in Q \cup \Gamma \).

![Figure 5](#)

---

**Proof.** We simulate a graph splicing system \( F = (N, T, P, I) \) with a graph splicing system \( F' = (N', T', P', I') \) which has \(|N'| = 1\). The simulation is a slight modification of the standard Turing machine simulation. Consider a symbol \( a \in T \). Let \(|N| = k\). We code the nonterminals in \( N \) using only \( a \) and a single nonterminal \( S \). Let \( N_1, N_2 \ldots N_k \) be an enumeration of the members of \( N \). Then \( N_i \) is coded as \( Sa^iSS \). \( P' \) is the same set of rules as \( P \) except that \( N_i \) is replaced by the subgraph shown in Fig. 16, wherever it occurs. \( I' \) is the same set of axioms as \( I \) except that \( N_i \) is replaced appropriately wherever it occurs.

First, we prove that \( L(F) \subseteq L(F') \). This is obvious, since axioms and rules used in the derivation of a graph in \( F \) can be replaced by corresponding axioms and rules of \( F' \) with the derivation still holding.

The proof that \( L(F') \subseteq L(F) \) is slightly more difficult. The proof is by induction on the number of steps required to derive a graph in \( F' \).

Consider the function \( f : N \rightarrow \{S, a\}^* \) defined by

\[
f(N_i) = Sa^iSS.
\]

Extend the function to \( f : (N \cup T)^* \rightarrow (\{S\} \cup T)^* \) in the usual way.

The induction hypothesis is as follows. If a linear graph \( w \) is derivable in \( F' \) in \( n \) steps or less, there is a unique graph \( w' \) corresponding to it in \( F \) that is derivable in \( F \) in the same number of steps.

The base case is \( n = 0 \). The graphs of \( F' \) derivable in zero steps are the axioms in \( I' \). These obviously correspond to unique axioms of \( I \) (by construction).
Let us assume that the induction hypothesis is true for \( n = n_0 \). Consider a linear graph that is derivable in \( n_0 + 1 \) steps in \( F' \). This implies that there are linear graphs \( w_1 \) and \( w_2 \) in \( F' \) and a rule \( p' \) of \( F' \) such that \( w_1 \) and \( w_2 \) are derivable in \( n_0 \) or fewer steps and the application of \( p' \) to \( w_1 \) and \( w_2 \) yields \( w \).

Consider the linear graph \( h[1] \) of \( p' \). Let this graph be as given in Fig. 17. In the graph, \( u_i \)'s belong to \( T^* \) and \( v_i \)'s are codings of nonterminals in \( F \). If \( v_1 \) is a coding of \( N_f(1) \) for each \( i \), then this rule corresponds to a rule \( p \) of \( F \) which has \( h[1] \) as given in Fig. 18. \( h[1] \) is an induced subgraph of \( w_1 \). In \( w_1 \), the subgraph in Fig. 17 can only be a code of the graph in Fig. 18. This is because the first character of \( v_1 \) is an \( S \) and it is followed by a terminal. This can only indicate the beginning of a code for a nonterminal. Similar reasoning holds for the graph \( h[2] \) of \( p' \) and \( w_2 \). Thus, if \( w_1 \) and \( w_2 \) yield \( w \) by splicing using \( p' \), then their corresponding graphs in \( F \) yield the corresponding graph of \( w \) in \( F \) by splicing using \( p \). Thus, this graph is derivable in \( F \) in \( n_0 + 1 \) steps.

In particular, linear graphs with only terminal labels code for themselves. Therefore, if a string \( w \in T^* \) is derivable in \( F' \), it is derivable in \( F \). This implies \( L(F') \subseteq L(F) \). \( \square \)

**Theorem 4.** There does not exist a set \( C_4 \) of graph splicing systems with \( \delta(C_4) = 1 \) that is computationally complete.

**Proof.** If there is only one graph in each rule, then the size of graphs in the corresponding splicing language is bounded. This means that the splicing language is finite. Therefore, no infinite set of strings can be generated by a splicing system with maximum 1 graph per rule. \( \square \)

**Theorem 5.** There exists a set \( C_5 \) of graph splicing systems with \( \delta(C_5) = 2 \) that is computationally complete.

**Proof.** Graph splicing systems used in the Turing machine simulation of Section 5 are precisely of this form. \( \square \)
Theorem 6. There exists a set $C_6$ of graph splicing systems with $\omega(C_6) = 3$ that is computationally complete.

Proof. We prove this by actually constructing, for every Turing machine $M$, a splicing system in which no more than three edges are cut and which simulates the Turing machine. The splicing system generates, for each string, a graph containing a copy of the string and a copy of the reverse of the string. Then we make use of the property that, for each Turing machine $M$, there exists a Turing machine $M_R$ such that $w \in L(M)$ iff $wR \in L(M_R)$. We simulate the machine $M_R$ on a copy of $wR$ and generate $w$ iff $wR$ is accepted by $M_R$.

Let $M_R = (Q, \Sigma, \Gamma, \delta, q_0, B, \{q_f\})$. Then we construct a graph splicing system $(N, T, P, I)$ as follows:

$$V = \Gamma \cup Q \cup \{E, X, X', Y, Z, C, Z_1, Z_2\} \cup \{(q, a)|q \in Q, a \in \Gamma\} \cup \{(q, a)|q \in Q, a \in \Gamma\} \cup \{a'|a \in \Gamma\}.$$ 

The axiom set $I$ is given in Fig. 19. The rule set $P$ has 29 finite classes of rules. We describe each class pictorially in Tables 3–7. Each rule has two graphs, $h_1$ and $h_2$. We describe these graphs and the resultant graphs obtained from these graphs after splitting and recombining. The cut-edges of the rules are shown in dotted lines (Table 8).

The rules in Classes 1–6 generate the graph shown in Fig. 20. Rules in Classes 7–9 introduce $q_0$ after $X$ and change $X$ into $Y$ to start the simulation in $wR$. Rules in Classes 10 and 11 are used to introduce blank $B$ after $Y$ or before $Z$. Rules in Classes 12–17 simulate the right move of the Turing machine of the form $\delta(q, a) = (q', a', R)$. The subgraph shown in Fig. 21 is changed to the subgraph shown in Fig. 22 by the rules in Class 12. $a$ is removed by the rules in Class 13 and $a'$ is introduced in that place by the rules in Class 14. $q$ is removed by the rules in Class 15 and $q'$ is introduced by the rules in Class 16. Rules in Class 17 remove $[q', a']$. After using the rules in Class 17, we end up with the graph as shown in Fig. 23. Rules in Classes 18 to 23 simulate a left move of the form $\delta(q, a) = (q', a', L)$ in a similar manner. When a final state $q_f$ is reached, rules in Classes 24 to 29 remove from the subgraph as shown.
Table 7
Classes of rules in \( P \)

<table>
<thead>
<tr>
<th>Class 21:</th>
<th>In the figure, ( b, a' \in \Gamma, q, q' \in Q. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} b \rightarrow {a} \ldots \rightarrow {a'} \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td>[ \begin{array}{c} b \rightarrow {a} \rightarrow Z \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 22:</th>
<th>In the figure, ( a', c, b, d \in \Gamma, q' \in Q. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow {a} \ldots \rightarrow {a'} \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td>[ \begin{array}{c} a \rightarrow {a} \rightarrow b \rightarrow {a'} \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 23:</th>
<th>In the figure, ( a', b \in \Gamma, q' \in Q. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow {a} \ldots \rightarrow {a'} \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td>[ \begin{array}{c} a \rightarrow {a} \rightarrow b \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 24:</th>
<th>In the figure, ( a, c \in N, b \in \Gamma. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow {a} \ldots \rightarrow {a'} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td>[ \begin{array}{c} a \rightarrow {a} \rightarrow b \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 25:</th>
<th>In the figure, ( a, c \in N, b \in \Gamma. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow {a} \ldots \rightarrow {a'} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td>[ \begin{array}{c} a \rightarrow {a} \rightarrow b \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
</tr>
</tbody>
</table>

---

Table 8
Classes of rules in \( P \)

<table>
<thead>
<tr>
<th>Class 26:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} X \rightarrow {a} \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 27:</th>
<th>In the figure, ( a \in \Sigma. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow {a} \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 28:</th>
<th>In the figure, ( a, b \in \Sigma. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow {a} \rightarrow b \rightarrow \ldots \rightarrow {l} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class 29:</th>
<th>In the figure, ( a, b \in \Sigma. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \begin{array}{c} a \rightarrow \ldots \rightarrow {a} \rightarrow {b} \ Z \rightarrow {l} \rightarrow Z' \end{array} ]</td>
<td></td>
</tr>
</tbody>
</table>
in Fig. 24 all other nodes except the nodes representing \( w \) between \( E \) and \( Y \) and we end up with a linear graph representing \( w \).

It can be seen as in the previous section that the graph language generated consists of linear graphs representing strings accepted by the Turing machine \( M \) and nothing else. □

6. Conclusions

The paper considers graph splicing systems and proves universality of the model with various restrictions. Graph splicing systems seem to be a good abstract model of systems in which chemical/bio-chemical processes occur. They are also very general in the sense that most formulations of string splicing systems in the literature can be obtained as special cases of graph splicing systems. The result that extended graph splicing systems with only finite sets of axioms and rules are computationally complete, even when some constraints are imposed on them, is a powerful result since most completeness results concerning splicing systems in the literature use either infinite sets of axioms/rules (see [4]) or place artificial constraints on the form of the rules (see [1]).

It is an open problem to find the minimum number of rules required for obtaining computational completeness of the model. It would be worthwhile to modify the definition of the graph splicing systems, for example, to allow for only undirected graphs, and study their generative power.

Acknowledgements

The authors wish to acknowledge the help of Ms. K. Arthi in the preparation of this manuscript.

References


