

## On Classes of One-dimensional Self-assembling Automata

Kazuhiro Saitou\*

Department of Mechanical Engineering and Applied Mechanics,  
University of Michigan, Ann Arbor, MI

Mark J. Jakiela†

Hunter Associate Professor of Mechanical Design,  
Department of Mechanical Engineering,  
Washington University, St. Louis, MO

**Abstract.** An abstract model of self-assembling systems is presented where assembly instructions are written as *conformational switches*—local rules that specify conformational changes of a component. The *self-assembling automaton* model is defined as a sequential rule-based machine that operates on one-dimensional strings of symbols. An algorithm is provided for constructing a self-assembling automaton that self-assembles an one-dimensional string of distinct symbols in a given particular subassembly sequence. Classes of self-assembling automata are then defined based on classes of subassembly sequences in which the components self-assemble. For each class of subassembly sequence, the minimum number of conformations is provided which is necessary to encode subassembly sequences in the class. It is shown that three conformations for each component are enough to encode any subassembly sequences of a string with arbitrary length.

### 1. Introduction

#### 1.1 Coded and uncoded self-assembly

Nature exhibits various kinds of self-assembly. One of the simplest is rain-drops on a leaf which, when placed close enough, merge together spontaneously to form one big drop with a smooth, curved shape. On the other extreme in complexity, protein molecules inside biological cells self-assemble to reproduce cells each time they divide. These two examples represent two types of self-assembly in nature—*coded* and *uncoded* self-assembly [20]. The

---

\*Electronic mail address: kazu@umich.edu.

†Electronic mail address: mjj@mecf.wustl.edu.

self-assembly of raindrops is an example of uncoded self-assembly, where assembly of each component (in this case each raindrop) is directed simply by minimization of potential (in this case thermodynamic) energy. Uncoded self-assembly, therefore, works to construct only the simplest of such structures. On the other hand, many complex structures in nature, for example, biological cells, arise *via* coded self-assembly, where instructions for the assembly of the system are built into its components. Self-assembly of biological cells, for example, is directed by conformational changes in protein molecules realized by energy-dissipating structures such as adenosine triphosphate.

A well-studied example of coded self-assembly in biology is the assembly of bacteriophages, a type of virus which infect bacterial cells. It is known that the assembly of new progeny viruses in their host cell occurs in a fixed morphogenic pattern, indicating coded self-assembly. Biologists believe that assembly instructions for this self-assembly of bacteriophages are written in each component molecule in the form of *conformational switches*. In a protein molecule with several bond sites, a conformational switch causes the formation of a bond at one site to change the conformation of another bond site. As a result, a conformational change which occurred at an assembly step provides the essential substrate for assembly at the next step [19].

Designing artificial systems with such a built-in set of assembly instructions is of great interest from an engineering point of view. The previous work of the authors on evolutionary design of mechanical conformational switches [15, 16] was a first attempt toward the design of (coded) self-assembling mechanical systems. In section 1.2, we overview some of the other previous work on self-assembling systems.

## 1.2 Previous work on self-assembling systems

Coded self-assembly of bacteriophages has been studied by a number of biologists (e.g., [2, 5]), and several computational models have been developed. In [6, 17, 18] a finite automaton model of the protein molecules which undergo conformational changes during the self-assembly of bacteriophage T4 is proposed. In [1] a set of local rules that specify the conformational changes of the component protein subunits in the self-assembly of icosahedral virus shells is identified. Using computer simulation it is shown that the subunits can form a closed icosahedral shell with the desired symmetry by following the local rules.

In engineering, several approaches have been proposed on the uncoded self-assembly of mechanical parts. In [11–13] a layered palletization technique is developed. Parts are “palletized” *via* vibratory agitation on a plastic pallet with an array of relief shapes that trap and orient the flowing parts. Assembly of two parts is done by palletizing the first part, and then palletizing the second part on top of the first part. In [21] a nonlayered palletization technique is used to integrate trapezoidal GaAs micro blocks on a Si substrate with trapezoidal holes by dispensing these in a carrier fluid (ethanol) onto the Si substrate. In [3] an experiment with the self-assembly of small hexag-

onal parts (1 mm in diameter) by placing a quantity of them on a slightly concave diaphragm that was agitated with a loudspeaker is discussed. In [8] the surface tension of water is utilized to self-assemble micro parts floating on the water surface. Self-assembly in these examples is driven by minimization of potential energy, and no explicit assembly instructions are built into the components.

Due to its inherent complexity, little work has been done on the coded self-assembly of physical systems. In [14] several designs of mechanical conformational switches are suggested that are used in devices that “self-reproduce.” These conformational switches cause a bond at one location to break a bond existing at another location or prevent a bond from occurring at another location. In [7] triangular parts are developed that employ switches realized with movable magnets which allow parts to bond together to form hexagons. The switches allow a part to be either in an active or inactive state. An activated part can bond to an inactivated part, turning the part to an activated state. These parts are assembled in a rotating box randomizer.

### 1.3 Motivation

The work described in this paper was motivated by our previous work on the evolutionary design of mechanical conformational switches [15, 16], where two different kinds of conformational switch models are studied. In these efforts, it was found that conformational switches can encode one or more subassembly sequences and that the encodable subassembly sequences depend on the conformational switch model employed. In particular, some subassembly sequences *cannot* be encoded by a conformational switch model no matter how many conformations are assumed. This raises the following questions.

- Is it possible to tell whether a subassembly sequence can be encoded by a given switch model?
- If so, how many conformations (or switch states) are necessary to encode a given subassembly sequence?

The relationship between subassembly sequences and conformational switch models is analogous to that between languages and machines (models of computation) in the theory of computation [10], with a subassembly sequence being a language, and a conformational switch that encodes the subassembly sequence being a machine that accepts the language. Under this view, the stated problems can be seen as analogs of the problem of finding a class of machines that accepts a given language, and the problem of finding the minimum number of states of a machine that accepts a given instance of the language.

This analogy motivated us to address the stated problems in the case of general self-assembling systems, not in the case of our particular implementation of conformational switches. More specifically, we approach the problems

by developing a formal model of self-assembling systems that abstracts the built-in assembly instructions in the form of conformational switches, and by identifying classes of self-assembling systems based on subassembly sequences in which the components of the systems self-assemble. As an initial attempt, an abstract representation of such a system is defined as a sequential machine that processes one-dimensional strings of symbols, which we refer to as an one-dimensional *self-assembling automaton*. This paper deals with symbolic one-dimensional assembly because of the following.

1. It is the simplest case of assembly and hence appropriate for the initial attempt.
2. It is easy to focus on subassembly sequences.
3. It is simple enough to generalize to complex self-assembling systems.

Before proceeding to a formal definition of a self-assembling system we discuss two simple examples that emphasize the essential aspect of machines of this type and illustrate how conformational switches can encode a subassembly sequence.

## 2. Theory of one-dimensional self-assembling automata

### 2.1 Conformational switches as assembly instructions

We consider the following scenario of simple one-dimensional assembly. Suppose we are given a one-dimensional component bin that initially contains a random assortment of components. Further suppose we are given a set of assembly instructions, or simply rules, describing which components can bind to which other components. Let the rules be of the form  $a + b \rightarrow ab$ , which means a component  $a$  and a component  $b$  can bind together to form an assembly  $ab$ . Assembly occurs by randomly picking two assemblies in the bin and mating them together, it is assumed that a component is a special case of an assembly. If one of the built-in rules *fires*, the two assemblies can bind together, and the resulting assembly is returned to the bin. To keep track of the subassembly sequences, the resulting assembly is parenthesized when it is formed. The rule  $a + b \rightarrow ab$  fires, for example, when a component  $a$  and a component  $b$  are picked and an assembly  $(ab)$  is added to the bin as a result. If no rules fire, the two assemblies are simply returned to the bin. Note that the rules are assumed to be *local* so that  $a + b \rightarrow ab$  also fires when, for example, an assembly  $(ca)$  and an assembly  $(ba)$  are picked, which results in forming an assembly  $((ca)(ba))$ . This random picking continues until no further rule firing is possible by picking *any* two assemblies in the bin. To see how the above scenario abstracts the behavior of self-assembling systems, consider a trivial example.

**Example 1.** Suppose our initial bin contains one component  $a$ , one component  $b$ , and two components  $c$  (which we represent as  $\langle a, b, c, c \rangle$ ), and our

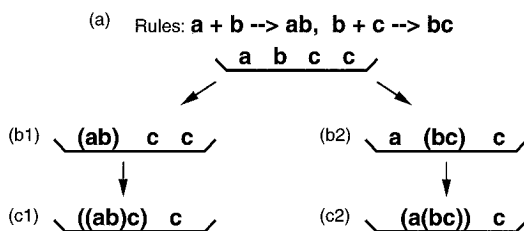


Figure 1: One-dimensional self-assembly with no conformational switches:  $abc$  can assemble in either order  $((ab)c$ ) or  $(a(bc))$ .

rule set contains  $a + b \rightarrow ab$  and  $b + c \rightarrow bc$ , as shown in Figure 1(a). As we continue the random picking process described previously, no change occurs to the contents of the bin until  $a$  and  $b$  are picked, or  $b$  and  $c$  are picked. If  $a$  and  $b$  are picked, the rule  $a + b \rightarrow ab$  fires and the resulting bin becomes  $\langle (ab), c, c \rangle$  (Figure 1(b1)). After that,  $((ab)c)$  is eventually formed when  $(ab)$  and  $c$  are picked and  $b + c \rightarrow bc$  fires. The resulting bin becomes  $\langle ((ab)c), c \rangle$  and no further rule firing is possible (Figure 1(c1)). Similarly, if  $b$  and  $c$  are picked, the rule  $b + c \rightarrow bc$  fires and the resulting bin becomes  $\langle a, (bc), c \rangle$  (Figure 1(b2)). After that,  $(a(bc))$  is formed eventually when  $a$  and  $(bc)$  are picked and  $a + b \rightarrow ab$  fires. The resulting bin becomes  $\langle (a(bc)), c \rangle$ , and no further rule firing is possible (Figure 1(c2)).

In Example 1, the rules do not enforce any subassembly sequences to assemble  $abc$ , in other words, the final bin contains either  $((ab)c)$  or  $(a(bc))$  depending on the order of rule firing. One can design conformational switches that enforce  $abc$  to be assembled only in one of the given subassembly sequences. Since conformational switches are essentially rules of state transition of components triggered by local interaction with other components, we can also represent them as rules of the form  $a + b \rightarrow a'b'$ , which means a component  $a$  and a component  $b$  can bind together to form an assembly  $a'b'$ , where  $a'$  and  $b'$  represent different conformations of  $a$  and  $b$  after conformational changes, respectively. Again, the rules are assumed to be local, hence for example,  $(ca)$  and  $(ba)$  can form  $((ca')(b'a))$  by applying this rule.

**Example 2.** Suppose a component  $b$  can take two conformations  $b$  and  $b'$ , and conformational switching between  $b$  and  $b'$  occurs according to the rules  $a + b \rightarrow ab'$  and  $b' + c \rightarrow b'c$ , as shown in Figure 2(a). An implementation of such mechanical conformational switches can be found in [16]. Starting with the same initial bin,  $\langle a, b, c, c \rangle$ , the random picking process eventually picks up  $a$  and  $b$ . As a result of firing the rule  $a + b \rightarrow ab'$ , the state of the bin becomes  $\langle (ab'), c, c \rangle$  (Figure 2(b)). After that, the rule  $b' + c \rightarrow b'c$  eventually fires to form  $((ab')c)$ . The resulting bin becomes  $\langle ((ab')c), c \rangle$ , and no further rule firing is possible (Figure 2(c)). Note that conformational change of component  $b$  after binding to  $a$  enforces  $abc$  to be assembled *only*

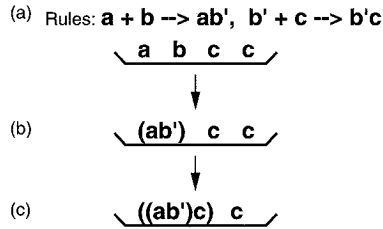


Figure 2: One-dimensional self-assembly with conformational switches: components  $abc$  can assemble only in the order  $((ab)c)$ .

in the order  $((ab)c)$ , by sending out a “signal” that indicates it has bound to  $a$  so it is ready to bind to  $c$ . We consider that  $((ab)c)$  and  $((ab')c)$  are the same assembly.

Similarly, the rules  $a + b' \rightarrow ab'$ ,  $b + c \rightarrow b'c$  enforce the only possible assembly order to be  $(a(bc))$ .

In Example 2, the rules  $a + b \rightarrow ab'$  and  $b' + c \rightarrow b'c$  can be viewed as a representation of the subassembly sequence  $((ab)c)$ . In other words, the subassembly sequence  $((ab)c)$  is *encoded* by the conformational switches represented by the rules  $a + b \rightarrow ab'$  and  $b' + c \rightarrow b'c$ . In the following sections, we discuss which type of conformational switches (equivalently, types of rules which represent switches) can encode which types of subassembly sequences.

## 2.2 Definition of one-dimensional self-assembling automata

We define a formal model of an one-dimensional self-assembling system as a sequential rule-based machine that operates on one-dimensional strings of symbols. We refer to this machine as an one-dimensional *self-assembling automaton*. In the following, we consider a *component* of a one-dimensional self-assembling automaton as an element of a finite set  $\Sigma$ , and an *assembly* is a string in  $\Sigma^+$ . Additionally, a component  $a \in \Sigma$  can take a finite number of *conformations* represented by  $a, a', a'', a''' \dots$ , and the transition from one conformation to another is triggered by local interactions with other components specified by a set of *assembly rules*. Each component, therefore, can be viewed as a finite automaton, hence the name self-assembling automata. In this paper, the symbol  $\Lambda$  is used to represent a null string.

**Definition 1.** An *one-dimensional self-assembling automaton* (SA) is a pair  $M = (\Sigma, R)$ , where  $\Sigma$  is a finite set of *components*, and  $R$  is a finite set of *assembly rules* of either of the forms  $a^\alpha + b^\beta \rightarrow a^\gamma b^\delta$  (*attaching rule*) or  $a^\alpha b^\beta \rightarrow a^\gamma b^\delta$  (*propagation rule*), where  $a, b \in \Sigma$  and  $\alpha, \beta, \gamma, \delta \in \{'\}^*$ . It is assumed that  $a^\Lambda = a$ . The *conformation set* of  $a \in \Sigma$  is a set  $Q_a = \{a^\alpha \mid \alpha \in \{'\}^*, a^\alpha \text{ appears in } R\}$ . The *conformation set* of  $M$  is the union of all conformation sets of  $a \in \Sigma$ .

We will often call a string in  $Q^+$  an *assembly by conformation*, or simply an *assembly* if there is no ambiguity with a string in  $\Sigma^+$ .

**Example 3.** Using Definition 1, the self-assembling system in Example 2 can be defined as  $M_1 = (\Sigma, R)$ , where  $\Sigma = \{a, b, c\}$ , and  $R = \{a + b \rightarrow ab', b' + c \rightarrow b'c\}$ . The conformation set of  $M_1$  is  $Q = \{a, b, b', c\}$ .

We view an SA as having an associated *component bin*, with an infinite number of slots, each of which can store an assembly (in conformation). Initially, a finite number of the slots contain assemblies. Self-assembly of components proceeds by *selecting* either a random pair of assemblies or an assembly in the component bin, and then *applying* the rules in  $R$  to the selected assembly. As a result of the rule application, assemblies are *deleted* from and *added* to the component bin, just like assertions are deleted from and added to working memory in rule-based inference systems. The rule application is done according to the following procedure where  $a, b \in \Sigma$  and  $\alpha, \beta, \gamma, \delta \in \{'\}^*$ .

1. If a pair of assemblies  $(x, y) = (za^\alpha, b^\beta u)$  for some  $z, u \in \Sigma^*$  is selected, and  $R$  contains the rule  $r = a^\alpha + b^\beta \rightarrow a^\gamma b^\delta$  ( $r$  fires), delete  $x$  and  $y$  and add  $za^\gamma b^\delta u$ .
2. If an assembly  $x = za^\alpha b^\beta u$  for some  $z, u \in \Sigma^*$  is selected, and  $R$  contains the rule  $r = a^\alpha b^\beta \rightarrow a^\gamma b^\delta$  ( $r$  fires), delete  $x$  and add  $za^\gamma b^\delta u$ .

If neither of the above applies, the selected assembly is simply returned to the component bin, leaving the bin unchanged. Note that at any point of self-assembly, the component bin contains a finite number of nonnull strings with finite length, since the total number of components in the initial bin is finite and no new components are created when applying the rules to the bin.

In order to describe the state of self-assembly at any point, there should be a representation of the component bin which lists all current assemblies. It is also convenient for the representation to keep a record of the sequence in which each assembly has been assembled from its components. To define such a representation, a few notations need to be introduced.

Let  $A$  be a finite set. A *bag*  $U$  over a finite set  $A$  is a list of some number of elements in  $A$ , written as  $U = \langle a \mid a \in A \rangle$ . In particular,  $U$  can contain more than one copy of elements in  $A$ . We write  $a \in U$  if  $\text{NUM}_a(U) > 0$ , where  $\text{NUM}_a(U)$  is the number of  $a$ s in  $U$ . Also, we define  $\text{SEQ}(A)$  as the language generated by the context-free grammar  $\forall a \in A, S \rightarrow (SS) \mid a$ . Note that  $A \subset \text{SEQ}(A)$ . A string  $x$  in  $\text{SEQ}(A)$  is a full parenthesization of a string  $u = \text{RM-PAREN}(x)$  in  $A^+$ , where  $\text{RM-PAREN}$  is a function that removes parentheses from its argument string. We interpret the parse tree of  $x$  as a (binary) assembly tree, that is, a representation of a pairwise assembly sequence of  $u$ .

**Definition 2.** Let  $\Sigma$  be a component set of an SA. A *subassembly sequence* is a string in  $\text{SEQ}(\Sigma)$ . A subassembly sequence  $x$  is *basic* if  $x$  contains at most one copy of elements in  $\Sigma$ , that is,  $\forall a \in \Sigma, N_a(x) \leq 1$ .

**Definition 3.** Let  $M = (\Sigma, R)$  be an SA. A *configuration* of  $M$  is a bag  $\langle x \mid x \in \text{SEQ}(Q) \rangle$ , where  $Q$  is the conformation set of  $M$ . Let  $x \in \text{SEQ}(\Sigma)$  be a subassembly sequence. A configuration  $\Gamma$  *covers*  $x$  if  $\Gamma = \langle a \mid a \in \Sigma \rangle$  and  $\forall a \in \Sigma, N_a(x) \leq \text{NUM}_a(\Gamma)$ .

The sequence of self-assembly can be traced by examining the configuration each time the component bin changes as a result of applying the rules in  $R$  to the component bin. To keep track of the order of assembly, the non-null strings newly added to the component bin are parenthesized in the new configuration if they were added by an attaching rule.

For two configurations  $\Gamma$  and  $\Phi$ , we write  $\Gamma \vdash_M \Phi$  if the configuration of  $M$  changes from  $\Gamma$  to  $\Phi$  as a result of applying a rule in  $R$  to the component bin *exactly once*, reading “ $\Phi$  is derived from  $\Gamma$  at one step.” Similarly,  $\Gamma \vdash_M^* \Phi$  if the configuration of  $M$  changes from  $\Gamma$  to  $\Phi$  as a result of applying the rules in  $R$  to the component bin *zero or more times*, reading “ $\Phi$  is derived from  $\Gamma$ .” If there is no ambiguity,  $\vdash_M$  and  $\vdash_M^*$  are often shortened to  $\vdash$  and  $\vdash^*$ , respectively.

**Example 4.** Consider  $M_1$  in Example 3. Let  $\Gamma = \langle a, b, c, c \rangle$  and  $\Phi = \langle a, b, c \rangle$ . The configurations  $\Gamma$  and  $\Phi$  cover the subassembly sequence  $((ab)c)$ . Self-assembly of  $((ab)c)$  from  $\Gamma$  proceeds as follows:

$$\langle a, b, c, c \rangle \vdash_{M_1} \langle (ab'), c, c \rangle \vdash_{M_1} \langle ((ab')c), c \rangle.$$

Given an SA as defined so far, the process of self-assembly eventually terminates when no rule firing is possible, or runs forever due to an infinite cycle of rule firing. It is natural to say an SA self-assembles a given string in a given sequence if the process of self-assembly terminates, and *all* terminating configurations contain the string that is assembled in the sequence. This is a conservative definition, requiring *stable* and *reliable* production of the string assembled in the sequence. Formally, this can be stated as in Definition 4.

**Definition 4.** Let  $M = (\Sigma, R)$  be an SA,  $\Gamma$  be a configuration of  $M$ , and  $x \in \text{SEQ}(\Sigma)$  be a subassembly sequence.  $\Gamma$  is *stable* if there is no rule firing from  $\Gamma$ , that is,  $C_M(\Gamma) = \{\Gamma\}$ , where  $C_M(\Gamma) = \{\Phi \mid \Gamma \vdash_M^* \Phi\}$ .  $M$  *terminates* from  $\Gamma$  if all configurations derived from  $\Gamma$  can derive a stable configuration, that is,  $\forall \Phi \in C_M(\Gamma), \exists \Phi_1 \in C_M(\Phi), C_M(\Phi_1) = \{\Phi_1\}$ .  $M$  *self-assembles*  $x$  from  $\Gamma$  if both of the following hold.

1.  $M$  terminates from  $\Gamma$ .
2.  $\forall \Phi \in C_M^*(\Gamma), \exists y \in \Phi$  such that  $x = \text{RM-PRIME}(y)$ , where  $C_M^*(\Gamma)$  is a set of stable configurations derived from  $\Gamma$ , and  $\text{RM-PRIME}$  is a function that removes the prime symbols (') from its argument.

**Example 5.**  $M_1$  in Example 3 self-assembles  $((ab)c)$  from  $\langle a, b, c, c \rangle$ .

### 2.3 Constructing one-dimensional self-assembling automata

Given a basic subassembly sequence  $x \in \text{SEQ}(\Sigma)$ , one can write a procedure to construct a set of assembly rules  $R$  such that  $M = (\Sigma, R)$  self-assembles  $x$  from any configuration  $\Gamma$  that covers  $x$ . Since  $x$  is a representation of a binary assembly tree, such an algorithm can be written as a simple recursive procedure. The following procedure MAKE-RULE-SET takes as input a basic subassembly sequence  $x \in \text{SEQ}(\Sigma)$ , a flag  $\eta \in \{\text{left}, \text{right}, \text{none}\}$ , and a rule set  $R$ . The flag  $\eta$  indicates from which side the next assembly would occur, with *none* indicating there is no next assembly, that is, the current assembly is the last one. MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) returns a pair  $(u, R)$ , where  $u$  is the final assembly (by conformation) such that  $\text{RM-PRIME}(u) = \text{RM-PAREN}(x)$  and  $R$  is the rule set containing the assembly rules to assemble  $x$  from  $\Gamma$ . In the following pseudocode, using conventions from [4],  $x, y, z \in \text{SEQ}(\Sigma)$  are basic subassembly sequences,  $a, b \in \Sigma$ ,  $\alpha, \beta \in \{\prime\}^*$ , and  $u, v \in Q^*$  where  $Q = \{a^\alpha \mid a \in \Sigma, \alpha \in \{\prime\}^*\}$ , and LEFT and RIGHT are functions that return the symbol at the left and right ends of the argument string, respectively.

```

MAKE-RULE-SET( $x, \eta, R$ )
1  if  $x = a$ 
2    then return  $(a, R)$ 
3  if  $x = (yz)$ 
4    then  $(u, R) \leftarrow \text{MAKE-RULE-SET}(y, \text{right}, R)$ 
5          $(v, R) \leftarrow \text{MAKE-RULE-SET}(z, \text{left}, R)$ 
6          $a^\alpha \leftarrow \text{RIGHT}(u)$ 
7          $b^\beta \leftarrow \text{LEFT}(v)$ 
8         if  $\eta = \text{none}$ 
9           then  $R \leftarrow R \cup \{a^\alpha + b^\beta \rightarrow a^\alpha b^\beta\}$ 
10          return  $(uv, R)$ 
11        if  $\eta = \text{left}$ 
12          then  $R \leftarrow R \cup \{a^\alpha + b^\beta \rightarrow a^{\text{INC}(\alpha)} b^\beta\}$ 
13              $(u, R) \leftarrow \text{PROPAGATE-LEFT}(u, R)$ 
14          return  $(uv, R)$ 
15        if  $\eta = \text{right}$ 
16          then  $R \leftarrow R \cup \{a^\alpha + b^\beta \rightarrow a^\alpha b^{\text{INC}(\beta)}\}$ 
17              $(v, R) \leftarrow \text{PROPAGATE-RIGHT}(v, R)$ 
18          return  $(uv, R)$ 

```

MAKE-RULE-SET recursively traverses the left and right subtrees ( $y$  and  $z$  in line 3), and adds an attaching rule to  $R$  that assembles  $(yz)$ . If a component will be assembled from the left at the next assembly step ( $\eta = \text{left}$  in line 11), propagation rules are added (by the procedure PROPAGATE-LEFT in line 13) that propagate conformational changes through the assembly corresponding to the left subtree. If a component will be assembled from the right at the next assembly step ( $\eta = \text{right}$  in line 15), propagation rules are added (by the procedure PROPAGATE-RIGHT in line 17) that propagate conformational

changes through the assembly corresponding to the right subtree. If there is no next assembly step, that is,  $yz$  is the final assembly ( $\eta = \text{none}$  in line 8), no propagation rules are added. The subroutines PROPAGATE-LEFT and PROPAGATE-RIGHT are defined as follows.

PROPAGATE-LEFT( $u, R$ )

```

19  if  $u = a^\alpha$ 
20    then return  $(a^{\text{INC}(\alpha)}, R)$ 
21  if  $u = va^\alpha b^\beta$ 
22    then  $R \leftarrow R \cup \{a^\alpha b^{\text{INC}(\beta)} \rightarrow a^{\text{INC}(\alpha)} b^{\text{INC}(\beta)}\}$ 
23          $(u, R) \leftarrow \text{PROPAGATE-LEFT}(va^\alpha, R)$ 
24    return  $(ub^{\text{INC}(\beta)}, R)$ 

```

PROPAGATE-RIGHT( $u, R$ )

```

25  if  $u = a^\alpha$ 
26    then return  $(a^{\text{INC}(\alpha)}, R)$ 
27  if  $u = a^\alpha b^\beta v$ 
28    then  $R \leftarrow R \cup \{a^{\text{INC}(\alpha)} b^\beta \rightarrow a^{\text{INC}(\alpha)} b^{\text{INC}(\beta)}\}$ 
29          $(u, R) \leftarrow \text{PROPAGATE-RIGHT}(b^\beta v, R)$ 
30    return  $(a^{\text{INC}(\alpha)} u, R)$ 

```

INC is the “conformation incrementor” function which appends the prime symbol ( $'$ ) to its argument string such that for  $\alpha \in \{\}'^*$ ,  $\text{INC}(\alpha) = \alpha'$ . For example,  $\text{INC}(\Lambda) = \Lambda'$  and  $\text{INC}(\Lambda') = \Lambda''$ .

**Example 6.** Consider  $\Sigma = \{a, b, c, d\}$  and  $x = ((a(bc))d)$ . A call of MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) returns with  $(ab''c'd, R)$  where  $R$  contains the following rules:  $b + c \rightarrow b'c$ ,  $a + b' \rightarrow ab''$ ,  $b''c \rightarrow b''c'$ , and  $c' + d \rightarrow c'd$ . It is clear that an SA  $M = (\Sigma, R)$  self-assembles  $x$  from the configurations that cover  $x$ , for example,  $\langle a, b, c, d \rangle$  and  $\langle a, a, b, b, c, c, d, d \rangle$ .

Theorem 1 tells the correctness of MAKE-RULE-SET in the general case of  $x$ . Namely, for any basic subassembly sequence  $x \in \text{SEQ}(\Sigma)$ , MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) returns a pair  $(u, R)$ , where  $\text{RM-PRIME}(u) = \text{RM-PAREN}(x)$  and  $R$  is a set of assembly rules such that an SA  $(\Sigma, R)$  self-assembles  $x$  from any configuration that covers  $x$ .

**Theorem 1.** MAKE-RULE-SET is correct.

*Proof.* We abbreviate MAKE-RULE-SET as MRS. Let  $x \in \text{SEQ}(\Sigma)$  be a basic subassembly sequence. We wish to prove the following statement: MRS( $x, \text{none}, \emptyset$ ) returns  $(u, R)$  such that  $\text{RM-PRIME}(u) = \text{RM-PAREN}(x)$  and  $M = (\Sigma, R)$  self-assembles  $x$  from any configuration  $\Gamma$  that covers  $x$ . The proof is done using mathematical induction on  $L(x)$ , where  $L(x) = |\text{RM-PAREN}(x)|$ .

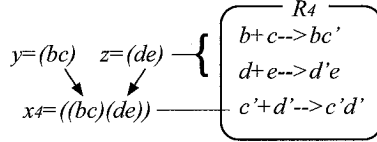


Figure 3: An example of  $R_k$  when  $k = 4$  and  $x_k = ((bc)(de))$ . In this case,  $u = bc'$  and  $v = d'e$ .

I. If  $L(x) = 1$ ,  $x = a$ ,  $a \in \Sigma$ .  $MRS(x, none, \emptyset)$  immediately returns  $(a, R)$  (line 2), where  $R = \emptyset$ . Since no assembly is necessary for  $x$ , the above statement is true.

II. Suppose the statement is true for  $L(x) = k$ . In other words, for any  $x_k \in SEQ(\Sigma)$  and  $L(x) = k$ ,  $M_k = (\Sigma, R_k)$  self-assembles  $x_k$  from any configuration  $\Gamma_k$  that covers  $x_k$ , where  $R_k$  is the rule set returned by  $MRS(x_k, none, \emptyset)$ . We are going to construct  $R_{k+1}$  from  $R_k$  and then show that  $R_{k+1}$  is in fact the rule set returned by  $MRS(x_{k+1}, none, \emptyset)$ .

Let  $x_k = (yz)$ ,  $y, z \in SEQ(\Sigma)$ ,  $(u, R_y) = MRS(y)$ , and  $(v, R_z) = MRS(z)$ . Without loss of generality, we can write  $u = a_1^{\alpha_1} a_2^{\alpha_2} \dots a_i^{\alpha_i}$  and  $v = a_{i+1}^{\alpha_{i+1}} a_{i+2}^{\alpha_{i+2}} \dots a_k^{\alpha_k}$ , where  $\forall i \in \{1, \dots, k\}$ ,  $a_i \in \Sigma$ ,  $\alpha_i \in \{^*\}$ . Figure 3 shows an example of  $R_k$  when  $k = 4$  and  $x_k = ((bc)(de))$ . Note in this example that  $u = bc'$  and  $v = d'e$ .

Since we can choose  $x_k$  arbitrarily, any basic subassembly sequence  $x_{k+1} \in SEQ(\Sigma)$  with  $L(x_{k+1}) = k+1$  can be written as either  $(a_{k+1}(yz))$  or  $((yz)a_{k+1})$ , where  $a_{k+1} \in \Sigma$  and  $\forall i \in \{1, \dots, k\}$ ,  $a_i \neq a_{k+1}$ . We consider these two cases separately.

a. If  $x_{k+1} = (a_{k+1}(yz))$ ,  $R_{k+1}$  must contain the following.

1. The rules that assemble  $y$  and  $z$ .
2. The rules that bring  $y$  and  $z$  together.
3. The rules that propagate conformational changes through  $y$  to the left.
4. The rules that bring  $a_{k+1}$  and  $(yz)$  together.

Figure 4 shows an example of  $R_{k+1}$  when  $k = 4$  and  $x_{k+1} = (a((bc)(de)))$ , constructed in this manner from the  $R_k$  shown in Figure 3.

In general, we can construct  $R_{k+1}$  from  $R_k$  by replacing the attaching rule

$$a_i^{\alpha_i} + a_{i+1}^{\alpha_{i+1}} \rightarrow a_i^{\alpha_i} a_{i+1}^{\alpha_{i+1}}$$

with the attaching rule

$$a_i^{\alpha_i} + a_{i+1}^{\alpha_{i+1}} \rightarrow a_i^{INC(\alpha_i)} a_{i+1}^{\alpha_{i+1}}$$

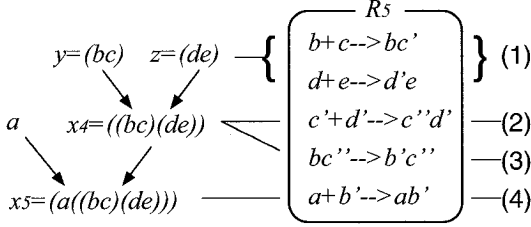


Figure 4: An example of  $R_{k+1}$  when  $k = 4$  and  $x_{k+1} = (a((bc)(de)))$ , constructed from the  $R_k$  in Figure 3.

(this corresponds to condition 2), adding the propagation rules

$$\begin{aligned}
 a_{i-1}^{\alpha_{i-1}} a_i^{\text{INC}(\alpha_i)} &\rightarrow a_{i-1}^{\text{INC}(\alpha_{i-1})} a_i^{\text{INC}(\alpha_i)} \\
 a_{i-2}^{\alpha_{i-2}} a_{i-1}^{\text{INC}(\alpha_{i-1})} &\rightarrow a_{i-2}^{\text{INC}(\alpha_{i-2})} a_{i-1}^{\text{INC}(\alpha_{i-1})} \\
 &\vdots \\
 a_1^{\alpha_1} a_2^{\text{INC}(\alpha_2)} &\rightarrow a_1^{\text{INC}(\alpha_1)} a_2^{\text{INC}(\alpha_2)}
 \end{aligned}$$

(this corresponds to condition 3), and adding the attaching rule

$$a_{k+1} + a_1^{\text{INC}(\alpha_1)} \rightarrow a_{k+1} a_1^{\text{INC}(\alpha_1)}$$

(this corresponds to condition 4). Since the rules in  $R_k$  other than  $a_i^{\alpha_i} + a_{i+1}^{\alpha_{i+1}} \rightarrow a_i^{\alpha_i} a_{i+1}^{\alpha_{i+1}}$  are unchanged in  $R_{k+1}$ ,  $R_{k+1}$  contains all the rules that assemble  $x$  and  $y$  separately (this corresponds to condition 1).

Now we wish to show that  $R_{k+1}$  constructed as described is in fact the same as the rule set returned by  $\text{MRS}(x_{k+1}, \text{none}, \emptyset)$ . Since  $x_{k+1} = (a_{k+1}(yz))$ ,  $\text{MRS}(x_{k+1}, \text{none}, \emptyset)$  calls  $\text{MRS}(a_{k+1}, \text{right}, \emptyset)$  (line 4) which immediately returns  $(a_{k+1}, \emptyset)$ , and then calls  $\text{MRS}((yz), \text{left}, \emptyset)$  (line 5). Let  $\tilde{R}_k$  be the rule set returned by  $\text{MRS}((yz), \text{left}, \emptyset)$ .  $\tilde{R}_k$  is the same as  $R_k$  except that  $\tilde{R}_k$  contains the rule

$$a_i^{\alpha_i} + a_{i+1}^{\alpha_{i+1}} \rightarrow a_i^{\text{INC}(\alpha_i)} a_{i+1}^{\alpha_{i+1}}$$

instead of

$$a_i^{\alpha_i} + a_{i+1}^{\alpha_{i+1}} \rightarrow a_i^{\alpha_i} a_{i+1}^{\alpha_{i+1}}$$

(line 12), and additionally contains the propagation rules added by PROPAGATE-LEFT (line 13) which propagate conformational changes through  $y$  to the left. After  $\text{MRS}((yz), \text{left}, \emptyset)$  returns, the rule

$$a_{k+1} + a_1^{\text{INC}(\alpha_1)} \rightarrow a_{k+1} a_1^{\text{INC}(\alpha_1)}$$

is added to  $\tilde{R}_k$  and  $\text{MRS}(x_{k+1}, \text{none}, \emptyset)$  returns. The returned rule set, therefore, contains exactly the same rules as in  $R_{k+1}$  constructed as above.

**b.** If  $x_{k+1} = ((yz)a_{k+1})$ ,  $R_{k+1}$  must contain the following.

1. The rules that assemble  $y$  and  $z$ .
2. The rules that bring  $y$  and  $z$  together.
3. The rules that propagate conformational changes through  $z$  to the right.
4. The rules that bring  $(yz)$  and  $a_{k+1}$  together.

A discussion similar to that of part **a** tells that  $\text{MRS}(x_{k+1}, \text{none}, \emptyset)$  returns the rule set that contains the rules described above. ■

The running time of **MAKE-RULE-SET** depends on the shape of the parse tree of the input (basic) subassembly sequence. The worst case behavior of **MAKE-RULE-SET** occurs when, at every step of its recursion, either **PROPAGATE-LEFT** or **PROPAGATE-RIGHT** is called. This is the case when new components are added from alternate directions at every step of assembly. The best case, on the other hand, is when there is no call of **PROPAGATE-LEFT** and **PROPAGATE-RIGHT**; that is, at every step of assembly, new components are added from the same direction. Theorem 2 provides the running time of **MAKE-RULE-SET** in the worst, best, and average cases.

**Theorem 2.** *Let  $x \in \text{SEQ}(\Sigma)$  be a basic subassembly sequence, and  $n = L(x)$ . The worst, best, and average running time of **MAKE-RULE-SET**( $x, \text{none}, \emptyset$ ) is  $\Theta(n^2)$ ,  $\Theta(n)$ , and  $\Theta(n \log n)$ , respectively.*

*Proof.* Let  $x$  be an input string of **MAKE-RULE-SET** and  $x = (yz)$ . In the worst case scenario, new components are added from alternate directions at every step of assembly. This implies that the input string  $x$  has totally unbalanced subtrees at every recursive step, that is,  $L(y) = 1$  or  $L(y) = n - 1$ . Since **PROPAGATE-LEFT** and **PROPAGATE-RIGHT** run in  $\Theta(n)$ , the recurrence of the running time of **MAKE-RULE-SET** is given as

$$T(n) = T(n - 1) + T(1) + \Theta(n).$$

Since  $T(1) = \Theta(1)$ ,  $T(n) = \Theta(n^2)$ . In the best case, new components are added from the same direction at every step of assembly. This also implies  $L(y) = 1$  or  $L(y) = n - 1$  at every recursive step. Since there is no call of **PROPAGATE-LEFT** and **PROPAGATE-RIGHT**, the running time is

$$T(n) = T(n - 1) + T(1) + \Theta(1) = T(n - 1) + \Theta(1) = \Theta(n).$$

On average, we can expect  $L(y) = n/2$  and hence the running time is

$$T(n) = 2T(n/2) + \Theta(n).$$

In the average case, therefore,  $T(n) = \Theta(n \log n)$ . ■

## 2.4 Classes of one-dimensional self-assembling automata

The best running time of MAKE-RULE-SET occurs when neither PROPAGATE-LEFT nor PROPAGATE-RIGHT are called during its execution. In this case, therefore, the rule set  $R$  returned by MAKE-RULE-SET contains *only* attaching rules, whereas  $R$  contains *both* attaching rules *and* propagation rules in other cases. Accordingly, two classes of SA can be defined based on the presence of propagation rules in the rule set.

**Definition 5.** Let  $M = (\Sigma, R)$  be an SA.  $M$  is *class I* if  $R$  contains *only* attaching rules.  $M$  is *class II* if  $R$  contains *both* attaching rules *and* propagation rules.

We now define the classes of *basic* subassembly sequences which correspond to each class of SA. The basic subassembly sequences that correspond to the best running time (class I SA) are those in which the direction from which new components are added does not alter during the entire self-assembly process. On the other hand, the direction must alter at least once in the basic subassembly sequences that correspond to the worst and average running time (class II SA). Classes of such subassembly sequences are described more precisely in the rest of the paper.

**Definition 6.** An *assembly template* is a string  $t \in \text{SEQ}(\{p\})$ . An *instance* of  $t$  on a finite set  $\Sigma$  is a subassembly sequence  $x \in \text{SEQ}(\Sigma)$  obtained by replacing each  $p$  in  $t$  by some  $a \in \Sigma$ . If  $x$  is an instance of  $t$ ,  $t$  is an *assembly template of  $x$* .

**Example 7.** Two strings  $t_1 = ((pp)(pp))$  and  $t_2 = ((p(pp))p)$  are assembly templates. Let  $\Sigma = \{a, b, c, d\}$ . The basic subassembly sequences  $x_1 = ((ab)(cd))$  and  $x_2 = ((b(ad))c)$  are instances of  $t_1$  and  $t_2$  on  $\Sigma$ , respectively.

**Definition 7.** An *assembly grammar* is a context-free grammar with a language that is a subset of  $\text{SEQ}(\{p\})$ . The class I assembly grammar  $G_1$  is defined by the following production rules:

$$\begin{aligned} S &\rightarrow (LR) \\ L &\rightarrow (Lp) \mid p \\ R &\rightarrow (pR) \mid p. \end{aligned}$$

The assembly templates in  $L(G_1)$  have the structure

$$(((\dots((pp)p)\dots)p)(p(\dots(p(pp))\dots)))$$

whose parse tree is shown in Figure 5. Each of the left and right subtrees is a *linear* assembly tree, which specifies self-assembly proceeding in one direction. The parse trees of the assembly templates in  $\text{SEQ}(\{p\})$  are general binary trees with no special structures.

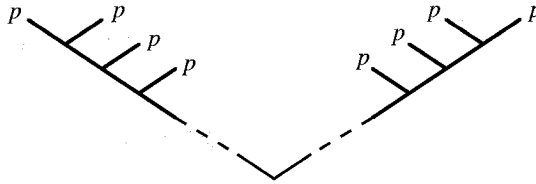


Figure 5: Parse tree of an assembly template generated by  $G_1$ .

**Example 8.** The assembly template  $t_1$  in Example 7 can be generated by  $G_1$ , for example, through the following derivation:

$$S \Rightarrow (LR) \Rightarrow ((Lp)R) \Rightarrow ((pp)R) \Rightarrow ((pp)(pR)) \Rightarrow ((pp)(pp))$$

and hence  $t_1 \in L(G_1)$ .

We can interpret  $L(G_1)$  and  $SEQ(\{p\})$  as sets of assembly templates with different numbers of changes in the direction of self-assembly. Let  $x$  be a subassembly sequence that is an instance of an assembly template  $t$ . If  $t \in L(G_1)$ , the direction of self-assembly does *not* alter during the self-assembly of  $x$ . If  $t \in SEQ(\{p\}) \setminus L(G_1)$ , the direction of self-assembly alters *at least once* during the self-assembly of  $x$ . Based on these observations, we claim that for any basic subassembly sequence with an assembly template in  $L(G_1)$ , there exists a corresponding class I SA, and for any basic subassembly sequence with an assembly template in  $SEQ(\{p\}) \setminus L(G_1)$ , there exists a corresponding class II SA. In the following proofs, we abbreviate MAKE-RULE-SET as MRS, and  $D(t)$  as the depth of the parse tree of  $t$ .

**Theorem 3.** For any basic subassembly sequence  $x$  that is an instance of an assembly template  $t \in L(G_1)$ , there exists a class I SA which self-assembles  $x$  from a configuration that covers  $x$ .

*Proof.* Let  $x \in SEQ(\Sigma)$ . By Theorem 1, it suffices to show that the rule set returned by  $MRS(x, none, \emptyset)$  contains no propagation rules. If  $D(t) = 0$ ,  $x = a \in \Sigma$ . Therefore,  $MRS(x, none, \emptyset)$  immediately returns  $(a, \emptyset)$  (line 2). If  $D(t) \geq 1$ , let  $m_l$  and  $m_r$  be the depth of the left and right subtree of  $t$ , respectively. Without loss of generality we can write  $t = (l_{m_l} r_{m_r})$  where  $l_i = (l_{i-1}p)$  for  $i = 1, \dots, m_l$ ,  $r_i = (pr_{i-1})$  for  $i = 1, \dots, m_r$ , and  $l_0 = r_0 = p$ . Let  $y_i$  and  $z_j$  be substrings of  $x$  that correspond to  $l_i$  and  $r_j$ , respectively. In this case,  $MRS(x, none, \emptyset)$  recursively calls  $MRS(y_{m_l}, right, \emptyset)$  and  $MRS(z_{m_r}, left, R_1)$  (lines 4 and 5), where  $R_1$  is the rule set returned by  $MRS(y_{m_l}, right, \emptyset)$ . Let  $R_2$  be the rule set returned by  $MRS(z_{m_r}, left, R_1)$ . Since no propagation rules are added to  $R_2$  before  $MRS(x, none, \emptyset)$  returns in line 10, it suffices to show that neither  $R_1$  nor  $R_2$  contain propagation rules. We consider these two cases separately.

**A.** To prove  $R_1$  contains no propagation rules, we wish to show that for any  $n \geq 0$ , the rule set  $\tilde{R}_n$  returned by  $\text{MRS}(y_n, \text{right}, \emptyset)$  contains no propagation rules. We will prove the statement using mathematical induction on  $n$ .

i. If  $n = 0$ ,  $y_0 = a_0 \in \Sigma$ . Since  $\text{MRS}(a_0, \text{right}, \emptyset)$  returns  $(a_0, \emptyset)$ , no propagation rules are in  $\tilde{R}_0$ .

ii. Suppose for some  $k > 0$ ,  $\text{MRS}(y_k, \text{right}, \emptyset)$  returns with the rule set  $\tilde{R}_k$  which contains no propagation rules. Since  $y_{k+1} = (y_k a_k)$  where  $a_k \in \Sigma$ ,  $\text{MRS}(y_{k+1}, \text{right}, \emptyset)$  recursively calls  $\text{MRS}(y_k, \text{right}, \emptyset)$  and  $\text{MRS}(a_k, \text{left}, \tilde{R}_k)$ . By the inductive hypothesis, no propagation rules are in  $\tilde{R}_k$ . Also,  $\text{MRS}(a_k, \text{left}, \tilde{R}_k)$  returns  $(a_k, \tilde{R}_k)$  since  $a_k \in \Sigma$ . After these calls return, the condition in line 15 is satisfied and an attaching rule is added to  $\tilde{R}_k$  (line 16). Let this new rule set be  $\hat{R}$ .  $\text{PROPAGATE-RIGHT}(a_k, \hat{R})$  is then called (line 17), which returns  $(a'_k, \hat{R})$ . Therefore,  $\text{MRS}(y_{k+1}, \text{right}, \emptyset)$  returns with the rule set  $\tilde{R}_{k+1} = \hat{R}$ , which contains no propagation rules (line 18).

**B.** To prove  $R_2$  contains no propagation rules, we wish to show that for any  $n \geq 0$ , the rule set  $\tilde{R}_n$  returned by  $\text{MRS}(z_n, \text{left}, R_0)$  contains no propagation rules, where  $R_0$  is a rule set containing no propagation rules. Mathematical induction on  $n$  similar to part **A** tells the statement holds. ■

**Theorem 4.** For any basic subassembly sequence  $x$  that is an instance of an assembly template  $t \in \text{SEQ}(\{p\}) \setminus L(G_1)$ , there exists a class II SA which self-assembles  $x$  from a configuration that covers  $x$ .

*Proof.* Let  $x \in \text{SEQ}(\Sigma)$ . By Theorem 1, it suffices to show that the rule set returned by  $\text{MRS}(x, \text{none}, \emptyset)$  contains at least one propagation rule. Since

$$\{s | s \in \text{SEQ}(\{p\}), D(s) \leq 2\} = \{p, (pp), ((pp)p), (p(pp)), ((pp)(pp))\} \subset L(G_1)$$

we consider  $D(t) \geq 3$ . Let  $m_l$  and  $m_r$  be the depth of the left and right subtree of  $t$ , respectively. Without loss of generality, we can write  $t = (l_{m_l}^l r_{m_r}^r)$  where  $l_i^l = (l_{i-1}^l l_{i-1}^r)$  for  $i = 1, \dots, m_l$ ,  $r_i^r = (r_{i-1}^l r_{i-1}^r)$  for  $i = 1, \dots, m_r$ ,  $l_i^l, r_i^l \in \text{SEQ}(\{p\})$ , and  $l_0^l = r_0^r = p$ . Then  $\exists j \in \{1, \dots, m_l\}$ ,  $L(l_j^l) \geq 2$ , or  $\exists j \in \{1, \dots, m_r\}$ ,  $L(r_j^r) \geq 2$ , since otherwise  $t \in L(G_1)$ . We consider these two cases separately.

**A.** Suppose  $\exists j \in \{1, \dots, m_l\}$ ,  $L(l_j^l) \geq 2$ . Let  $y_j^l, y_j^r$  and  $y_{j+1}^l$ , be substrings of  $x$  that correspond to  $l_j^l, l_j^r$ , and  $l_{j+1}^l$ . Let  $R_0$  be a rule set containing no propagation rules. It suffices to show that the rule set returned by  $\text{MRS}(y_{j+1}^l, \text{right}, R_0)$  contains at least one propagation rule. Since  $y_{j+1}^l = (y_j^l y_j^r)$ ,  $\text{MRS}(y_{j+1}^l, \text{right}, R_0)$  recursively calls  $\text{MRS}(y_j^l, \text{right}, R_0)$  and  $\text{MRS}(y_j^r, \text{left}, R_1)$  (lines 4 and 5), where  $R_1$  is the rule set returned by  $\text{MRS}(y_j^l, \text{right}, R_0)$ . Let  $(v_j^r, R_2)$  be a return value of  $\text{MRS}(y_j^r, \text{left}, R_1)$ , where  $\text{RM-PRIME}(v_j^r) = \text{RM-PAREN}(y_j^r)$ . Since  $L(y_j^r) \geq 2$ ,  $|v_j^r| \geq 2$ . After  $\text{MRS}(y_j^r, \text{left}, R_1)$  returns, the condition in line 13 is satisfied and an attaching rule is added to  $R_2$  (line 16). Let this new rule set be  $\hat{R}_2$ .  $\text{PROPAGATE-RIGHT}(v_j^r, \hat{R}_2)$  is then called

(line 17). Since  $|v_j^r| \geq 2$ , the condition in line 27 is satisfied and at this point, a propagation rule is added to  $\hat{R}_2$ . Therefore, the rule set returned by  $\text{MRS}(y_{j+1}^l, \text{right}, R_0)$  contains at least one propagation rule.

**B.** Suppose  $\exists j \in \{1, \dots, m_r\}$ ,  $L(r_j^l) \geq 2$ . A discussion similar to part **A** tells that the rule set returned by  $\text{MRS}(z_{j+1}^r, \text{left}, R_0)$  contains at least one propagation rule. ■

In addition to Theorems 3 and 4, we can say that class I SA is not “powerful” enough to encode any class II basic subassembly sequences.

**Corollary 1.** *For any class II basic subassembly sequence  $x$ , there exist no class I SA which self-assembles  $x$  from a configuration that covers  $x$ .*

*Proof.* By Theorem 4, there exists a class II SA  $M_{\text{II}} = (\Sigma, R_{\text{II}})$  which self-assembles  $x$  from a configuration that covers  $x$ . Since the conformational changes realized by a propagation rule cannot be realized by using only attaching rules, there are no rule sets containing only attaching rules which is equivalent to  $R_{\text{II}}$ . ■

It is important to point out that Theorem 4 and Corollary 1 state that a class II basic subassembly sequence  $x$  (i.e., strings “outside”  $L(G_1)$ ) cannot be *stably* and *reliably* self-assembled as defined in Definition 4 without propagation rules. A class I SA can produce  $x$ , with some probability, in *some* (not all) of its terminating configurations.

## 2.5 Minimum conformation self-assembling automata

In this section, we provide the minimum number of conformations necessary to encode a given subassembly sequence based on the classes of basic subassembly sequences introduced in section 2.4. Since the number of conformations may vary for each component, we simply define the number of conformations of an SA to be the maximum number of conformations of all components.

**Definition 8.** Let  $M$  be an SA.  $M$  is an SA with  $n$  conformations if

$$n = \max_{\alpha \in Q} |\alpha|,$$

where  $Q$  is the conformation set of  $M$ .

**Definition 9.** The class II assembly grammar  $G_{\text{II}}$  is defined by the following production rules:

$$\begin{aligned} S &\rightarrow (L_0 R_0) \\ L_0 &\rightarrow (L_0 R_1) \mid R_1 \\ R_0 &\rightarrow (L_1 R_0) \mid L_1 \\ L_1 &\rightarrow (L_1 p) \mid p \\ R_1 &\rightarrow (p R_1) \mid p. \end{aligned}$$

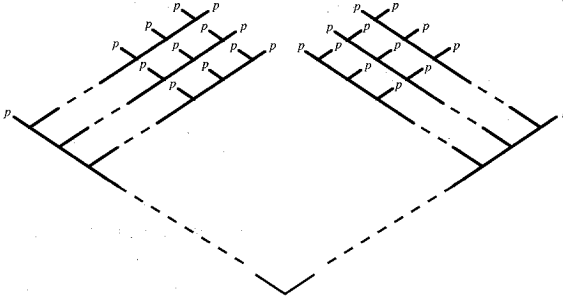


Figure 6: Parse tree of an assembly template generated by  $G_{II}$ .

Note that  $L(G_I) \subset L(G_{II}) \subset \text{SEQ}(\{p\})$ . The assembly templates in  $L(G_{II})$  have the structure

$$((((\dots((R_1R_1)R_1)\dots)R_1)(L_1(\dots(L_1(L_1L_1))\dots))))$$

where  $L_1$  and  $R_1$  are strings of the forms  $((\dots((pp)p)\dots)p)$  and  $(p(\dots(p(pp))\dots))$ , respectively. The corresponding parse tree is shown in Figure 6. The parse tree in Figure 6 can be obtained from the parse tree in Figure 5, by replacing leaves at the right branches of the left subtree by a linear assembly tree, and *vice versa*. Let  $x$  be a subassembly sequence and  $t$  is an assembly template of  $x$ . If  $t \in L(G_{II}) \setminus L(G_I)$ , the direction of self-assembly alters *exactly once*, and if  $t \in \text{SEQ}(\{p\}) \setminus L(G_{II})$ , the direction of self-assembly alters *more than once* during the self-assembly of  $x$ .

**Example 9.** The assembly template  $t_2$  in Example 7 cannot be generated by  $G_I$  but can be generated by  $G_{II}$ , for example, through the following derivation:

$$\begin{aligned} S &\Rightarrow (L_0R_0) \Rightarrow ((L_0R_1)R_0) \Rightarrow ((pR_1)R_0) \Rightarrow ((p(pR_1))R_0) \\ &\Rightarrow ((p(pp))R_0) \Rightarrow ((p(pp))p) \end{aligned}$$

and hence  $t_2 \in L(G_{II}) \setminus L(G_I)$ . An assembly template  $t_3 = (p((p(pp))p)) \in \text{SEQ}(\{p\})$  cannot be generated by  $G_{II}$  and hence  $t_3 \in \text{SEQ}(\{p\}) \setminus L(G_{II})$ .

The minimum number of conformations of SA that are necessary to self-assemble a given basic subassembly sequence  $x$  depends on whether  $x$  is an instance of an assembly template in  $L(G_I)$ ,  $L(G_{II}) \setminus L(G_I)$ , or  $\text{SEQ}(\{p\}) \setminus L(G_{II})$ . Since any attaching rules produced by MAKE-RULE-SET requires at most two conformations for each component, the minimum number is two if  $x$  is an instance of an assembly template in  $L(G_I)$ . The proof is very similar to that of Theorem 3.

**Theorem 5.** For any basic subassembly sequence  $x$  that is an instance of an assembly template  $t \in L(G_I)$ , there exists a class I SA  $M$  with two conformations which self-assembles  $x$  from a configuration  $\Gamma$  that covers  $x$ . For  $L(x) \geq 3$ ,  $M$  is an SA with the minimum number of conformations which self-assembles  $x$  from  $\Gamma$ .

*Proof.* Let  $x \in \text{SEQ}(\Sigma)$  and  $R$  be the rule set returned by  $\text{MRS}(x, \text{none}, \emptyset)$ . For the first part, we wish to show the following statement:  $R$  contains only attaching rules of the form  $a^\alpha + b^\beta \rightarrow a^\gamma b^\delta$  such that  $|\alpha|, |\beta|, |\gamma|, |\delta| < 2$ , where  $a, b \in \Sigma$  and  $\alpha, \beta, \gamma, \delta \in \{'\}^*$ . If  $D(t) = 0$ ,  $x = a \in \Sigma$ . Therefore,  $\text{MRS}(x, \text{none}, \emptyset)$  immediately returns  $(a, \emptyset)$  (line 2), hence the statement holds. If  $D(t) \geq 1$ , without loss of generality we can write  $t = (l_{m_l} r_{m_r})$  where  $l_i = (l_{i-1}p)$  for  $i = 1, \dots, m_l$ ,  $r_i = (pr_{i-1})$  for  $i = 1, \dots, m_r$ , and  $l_0 = r_0 = p$ . Let  $y_i$  and  $z_j$  be substrings of  $x$  that correspond to  $l_i$  and  $r_j$ , respectively. In this case,  $\text{MRS}(x, \text{none}, \emptyset)$  recursively calls  $\text{MRS}(y_{m_l}, \text{right}, \emptyset)$  and  $\text{MRS}(z_{m_r}, \text{left}, R_1)$  (lines 4 and 5), where  $R_1$  is the rule set returned by  $\text{MRS}(y_{m-1}, \text{right}, \emptyset)$ . Let  $R_2$  be the rule set returned by  $\text{MRS}(z_{m_r}, \text{left}, R_1)$ . Since there are no INC in the attaching rule added to  $R_2$  in line 9, it suffices to show that the statement holds for both  $R_1$  and  $R_2$ . We consider these two cases separately.

**A.** To prove that the statement holds for  $R_1$ , we wish to show that for any  $n \geq 0$ , the statement holds for the rule set  $\tilde{R}_n$  returned by  $\text{MRS}(y_n, \text{right}, \emptyset)$ . We prove this using mathematical induction on  $n$ .

i. If  $n = 0$ ,  $y_0 = a_0 \in \Sigma$ . Since  $\text{MRS}(a_0, \text{right}, \emptyset)$  returns  $(a_0, \emptyset)$ , the statement holds for  $\tilde{R}_0 = \emptyset$ .

ii. Suppose for some  $k > 0$ ,  $\text{MRS}(y_k, \text{right}, \emptyset)$  returns with the rule set  $\tilde{R}_k$  for which the statement is true. Since  $y_{k+1} = (y_k a_k)$  where  $a_k \in \Sigma$ ,  $\text{MRS}(y_{k+1}, \text{right}, \emptyset)$  recursively calls  $\text{MRS}(y_k, \text{right}, \emptyset)$  and  $\text{MRS}(a_k, \text{left}, \tilde{R}_k)$ . By the inductive hypothesis, the statement is true for  $\tilde{R}_k$ . Also,  $\text{MRS}(a_k, \text{left}, \tilde{R}_k)$  returns  $(a_k, \tilde{R}_k)$  since  $a_k \in \Sigma$ . After these calls return, the condition in line 15 is satisfied and an attaching rule  $a_{k-1}^\alpha + a_k \rightarrow a_{k-1}^\alpha a'_k$  is added to  $\tilde{R}_k$ . Let this new rule set be  $\hat{R}$ . Since the statement holds for  $\tilde{R}_k$ ,  $|\alpha| < 2$ . Therefore, the statement also holds for  $\hat{R}$ . PROPAGATE-RIGHT( $a_k, R_1$ ) is then called (line 17), which returns  $(a'_k, R_1)$ . Therefore,  $\text{MRS}(y_{k+1}, \text{right}, \emptyset)$  returns with the rule set  $\tilde{R}_{k+1} = \hat{R}$  for which the statement holds.

**B.** To prove that the statement holds for  $R_2$ , we wish to show that for any  $n \geq 0$ , the statement holds for the rule set  $\tilde{R}_n$  returned by  $\text{MRS}(z_n, \text{left}, R_0)$ , where  $R_0$  is a rule set for which the statement holds. Mathematical induction on  $n$  similar to part **A** tells that this is the case.

Since at least two conformations are necessary for any  $x$  with  $L(x) \geq 3$ ,  $M$  is an SA with the minimum number of conformations which self-assembles  $x$  from  $\Gamma$ . ■

The “conformation incrementor” INC used in MAKE-RULE-SET simply appends the prime symbol (') to its argument string each time it is called. The number of conformations of a component, therefore, could be very large depending on how many times INC is called for the component before MAKE-RULE-SET returns. Alternatively, we can use a “modulo  $n$ ” conformation incrementor  $\text{INC}_n$  such that

$$\text{INC}_n(\alpha) = \begin{cases} \alpha' & \text{if } |\alpha| < n \\ \Lambda & \text{if } |\alpha| = n. \end{cases}$$

For example,  $\text{INC}_2(\Lambda) ='$  and  $\text{INC}_2(') = \Lambda$ . Using this notation, we can write  $\text{INC}$  as  $\text{INC}_\infty$ . Running  $\text{MAKE-RULE-SET}$  with  $\text{INC}_n$ , instead of  $\text{INC}_\infty$ , produces assembly rules with at most  $n$  conformations for a component. Such rules, however, are no longer guaranteed to self-assemble the components in a given subassembly sequence. In particular, there could be more than one conflicting propagation rule that specifies different conformational changes for the same two adjacent components. In order to show that  $\text{MAKE-RULE-SET}$  run with  $\text{INC}_n$  instead of  $\text{INC}_\infty$  is correct, therefore, it suffices to show that no such conflicts among propagation rules are possible. There are two cases to be considered. First, if the rule set  $R$  contains at most one propagation rule for each two adjacent components, no conflict is possible. Therefore, the statement is true for the smallest possible  $n$ , that is,  $n = 2$ . This is the case if the subassembly sequence  $x$  is an instance of an assembly template  $t \in L(G_{\text{II}}) \setminus L(G_{\text{I}})$ , when the direction of self-assembly alters *exactly once*. Second, if  $R$  contains more than one propagation rule for the same two adjacent components,  $n$  must be large enough to cause no conflicts among the propagation rules. This corresponds to the case where  $x$  is an instance of  $t \in \text{SEQ}(\{p\}) \setminus L(G_{\text{II}})$ , when the direction of self-assembly alters *more than once*. In the following proof,  $G_{\text{L}}$  and  $G_{\text{R}}$  are the assembly grammars defined by the production rules  $S \rightarrow (Sp) \mid p$  and  $S \rightarrow (pS) \mid p$ , respectively.

**Theorem 6.** *For any basic subassembly sequence  $x$  that is an instance of an assembly template  $t \in L(G_{\text{II}}) \setminus L(G_{\text{I}})$ , there exists a class II SA  $M$  with two conformations which self-assembles  $x$  from a configuration  $\Gamma$  that covers  $x$ . And  $M$  is an SA with the minimum number of conformations which self-assembles  $x$  from  $\Gamma$ .*

*Proof.* Let  $x \in \text{SEQ}(\Sigma)$ , and  $R$  be the rule set returned by  $\text{MAKE-RULE-SET}(x, \text{none}, \emptyset)$ . For the first part, we wish to show that for any two adjacent components  $ab$  in  $x$ ,  $R$  contains at most one propagation rule of the form  $a^\alpha b^\beta \rightarrow a^\gamma b^\delta$ , where  $\alpha, \beta, \gamma, \delta \in \{'\}^*$ ,  $a, b \in \Sigma$ , and  $ab$  is a substring in  $\text{RM-PAREN}(x)$ . Without loss of generality, we can write  $t = (l_{m_l}^l r_{m_r}^r)$  where  $l_i^l = (l_{i-1}^l l_{i-1}^r)$  for  $i = 1, \dots, m_l$ ,  $r_i^r = (r_{i-1}^l r_{i-1}^r)$  for  $i = 1, \dots, m_r$ ,  $l_i^l \in L(G_{\text{R}})$ ,  $r_i^r \in L(G_{\text{L}})$ , and  $l_0^l = r_0^r = p$ . Then  $\exists j \in \{1, \dots, m_l\}$ ,  $L(l_j^l) \geq 2$ , or  $\exists j \in \{1, \dots, m_r\}$ ,  $L(r_j^r) \geq 2$ , since otherwise  $t \in L(G_{\text{I}})$ . We consider these two cases separately.

**A.** Suppose  $\exists j \in \{1, \dots, m_l\}$ ,  $L(l_j^l) \geq 2$ . Let  $y_j^l$ ,  $y_j^r$  and  $y_{j+1}^l$ , be substrings of  $x$  that correspond to  $l_j^l$ ,  $l_j^r$  and  $l_{j+1}^l$ . Let  $ab$  be an arbitrary substring of  $\text{RM-PAREN}(y_{j+1}^l)$ , and  $R_0$  be a rule set containing no propagation rules. Since  $y_{j+1}^l$  and  $y_{j+1}^r$  do not overlap,  $ab$  is not a substring of  $\text{RM-PAREN}(y_{j+1}^r)$ . It suffices, therefore, to show the following statement: the rule set returned by  $\text{MRS}(y_{j+1}^l, \text{right}, R_0)$  contains at most one propagation rule of the form  $a^\alpha b^\beta \rightarrow a^\gamma b^\delta$ , where  $\alpha, \beta, \gamma, \delta \in \{'\}^*$ .

Since  $y_{j+1}^l = (y_j^l y_j^r)$ ,  $\text{MRS}(y_{j+1}^l, \text{right}, R_0)$  recursively calls  $\text{MRS}(y_j^l, \text{right}, R_0)$  and  $\text{MRS}(y_j^r, \text{left}, R_1)$  (lines 4 and 5), where  $R_1$  is the rule set returned by

$MRS(y_j^l, right, R_0)$ . Let  $(v_j^r, R_2)$  be a return value of  $MRS(y_j^r, left, R_1)$ , where  $RM-PRIME(v_j^r) = RM-PAREN(y_j^r)$ . Since  $l_j^r \in L(G_R)$  and  $L(G_R) \subset L(G_I)$ , only attaching rules are required to assemble  $y_j^r$ , and hence  $R_2$  contains no propagation rules. Since  $L(y_j^r) \geq 2$ ,  $|v_j^r| \geq 2$ . After  $MRS(y_j^r, left, R_1)$  returns, the condition in line 13 is satisfied and an attaching rule is added to  $R_2$  (line 16). Let this new rule set be  $\hat{R}_2$ .  $PROPAGATE-RIGHT(v_j^r, \hat{R}_2)$  is then called (line 17). Since  $|v_j^r| \geq 2$ , the condition in line 27 is satisfied and when  $PROPAGATE-RIGHT(v_j^r, \hat{R}_2)$  returns, exactly one propagation rule of the form  $a^\alpha b^\beta \rightarrow a^\alpha b(INC(\beta))$  is added to  $\hat{R}_2$  for each substring  $ab$  of  $RM-RAREN(y_j^r)$ . Since this is the only time the propagation rules are added, the statement holds.

**B.** Suppose  $\exists j \in \{1, \dots, m_r\}$ ,  $L(r_j^l) \geq 2$ . Let  $ab$  be an arbitrary substring of  $RM-PAREN(z_{j+1}^r)$ . A discussion similar to part **A** tells that the rule set returned by  $MRS(z_{j+1}^r, left, R_0)$  contains at most one propagation rule of the form  $a^\alpha b^\beta \rightarrow a^\gamma b^\delta$  for each substring  $ab$  of  $RM-RAREN(z_j^l)$ .

$MAKE-RULE-SET(x, none, \emptyset)$  run with  $INC_2$  causes no conflict among propagation rules, since  $R$  contains at most one propagation rule for any two adjacent components in  $x$ . By Theorems 1 and 4, therefore, there exists a class II SA  $M$  with two conformations which self-assembles  $x$  from  $\Gamma$ . Since  $L(G_I) \subset L(G_{II})$ , Theorem 5 tells at least two conformations are necessary and therefore,  $M$  is an SA with minimum conformation which self-assembles  $x$  from  $\Gamma$ . ■

**Example 10.** Consider  $\Sigma = \{a, b, c, d\}$  and  $x = ((a(bc))d)$ . The subassembly sequence  $x$  is an instance of  $t_2 = ((p(pp))p)$  in Example 7. From Example 9,  $t_2 \in L(G_{II}) \setminus L(G_I)$ . A call of  $MAKE-RULE-SET(x, none, \emptyset)$  run with  $INC_2$  returns with  $(abc'd, R)$  where  $R$  contains the following rules:  $b+c \rightarrow b'c$ ,  $a+b' \rightarrow ab$ ,  $bc \rightarrow bc'$ , and  $c'+d \rightarrow c'd$ . It is clear that  $M = (\Sigma, R)$  is an SA with two conformations which self-assembles  $x$  from the configurations that cover  $x$ , for example,  $\langle a, b, c, d \rangle$  and  $\langle a, a, b, b, c, c, d, d \rangle$ .

In the case where a basic subassembly sequence  $x$  is an instance of an assembly template in  $SEQ(\{p\}) \setminus L(G_{II})$ , we claim that only three conformations are necessary to encode an arbitrary  $x$ . This might sound counter-intuitive since we are claiming that only three conformations can encode basic subassembly sequences with arbitrary (possibly very large) sizes. The proof of this claim is based on the observation that there are only two kinds of propagation rules; the rules which propagate conformational changes to the left, and the rules which propagate conformational changes to the right, and that for any given two adjacent components these two kinds of propagation rules *always* fire in alternative order. As in the previous case, we will prove this statement by showing that  $MAKE-RULE-SET$  run with  $INC_n$  causes no conflicts among propagation rules of the same adjacent components in the case of  $n = 3$ . To do this, we define a concept called a  $n$ -conformation transition cycle. Then, we prove that no such conflicts among propagation rules

are possible for  $\text{INC}_n$  if there exists a  $n$ -conformation transition cycle. Finally, we show that there exists a three-conformation transition cycle. Since our focus is on conformational propagation between two arbitrary adjacent components, it is convenient to introduce several notations first.

A *conformational transition rule* is a rule of the form  $\alpha \cdot \beta \rightarrow \gamma \cdot \delta$ , where  $\alpha, \beta, \gamma, \delta \in \{\}'^*$ . Let  $r$  be a propagation rule and  $\rho$  be a conformational transition rule. We write  $\rho = \text{TRN}(r)$  if  $\rho = \alpha \cdot \beta \rightarrow \gamma \cdot \delta$  and  $r = a^\alpha b^\beta \rightarrow a^\gamma b^\delta$  where  $a, b \in \Sigma$ . For two conformational transition rules  $\rho_1$  and  $\rho_2$ , we write  $\rho_1 \xrightarrow{n} \rho_2$  if *one* of the following holds.

- $\rho_1 = \alpha \cdot \beta \rightarrow \text{INC}_n(\alpha) \cdot \beta$  and  $\rho_2 = \text{INC}_n(\text{INC}_n(\alpha)) \cdot \beta \rightarrow \text{INC}_n(\text{INC}_n(\alpha)) \cdot \text{INC}_n(\beta)$ .
- $\rho_1 = \alpha \cdot \beta \rightarrow \alpha \cdot \text{INC}_n(\beta)$  and  $\rho_2 = \alpha \cdot \text{INC}_n(\text{INC}_n(\beta)) \rightarrow \text{INC}_n(\alpha) \cdot \text{INC}_n(\text{INC}_n(\beta))$ .

In addition, we say that two conformational transition rules  $\rho_1 = \alpha_1 \cdot \beta_1 \rightarrow \gamma_1 \cdot \delta_1$  and  $\rho_2 = \alpha_2 \cdot \beta_2 \rightarrow \gamma_2 \cdot \delta_2$  are *conflicting* if  $(\alpha_1, \beta_1) = (\alpha_2, \beta_2)$  and  $(\gamma_1, \delta_1) \neq (\gamma_2, \delta_2)$ .

**Definition 10.** A  $n$ -conformation transition cycle is a finite sequence of nonconflicting conformational transition rules  $(\rho_1, \rho_2, \dots, \rho_m)$  such that for  $i = 1, 2, \dots, m-1$ ,  $\rho_i \xrightarrow{n} \rho_{i+1}$ , and  $\rho_m \xrightarrow{n} \rho_1$ .

**Corollary 2.** Let  $x \in \text{SEQ}(\Sigma)$  be a basic subassembly sequence that is an instance of an assembly template  $t \in \text{SEQ}(\{p\}) \setminus L(G_{\text{II}})$ . Let  $R$  be the rule set returned by  $\text{MAKE-RULE-SET}(x, \eta, \emptyset)$  run with  $\text{INC}_\infty$ . There exists a substring  $ab$  of  $\text{RM-PAREN}(x)$  such that  $R$  contains more than one propagation rule for  $ab$ . For any two propagation rules  $r_1, r_2 \in R$  of  $ab$ ,  $\text{TRN}(r_1) \overset{\infty}{\rightsquigarrow} \text{TRN}(r_2)$  if  $r_2$  fires after  $r_1$  and no propagation rules of  $ab$  fires between  $r_1$  and  $r_2$ .

*Proof.* Let  $\alpha, \beta \in \{\}'^*$ . During the self-assembly of  $x$ , the direction of self-assembly alters more than once since otherwise  $t \in L(G_{\text{II}}) \setminus L(G_{\text{I}})$ . This implies that there exists at least one substring  $ab$  of  $\text{RM-PAREN}(x)$  such that  $R$  contains more than one propagation rule of the form  $a^\alpha b^\beta \rightarrow a^\gamma b^\delta$ .

If  $r_1$  is added to  $R$  by  $\text{PROPAGATE-LEFT}$  in line 22, we can write  $r_1$  as  $r_1 = a^\alpha b^\beta \rightarrow a^{\text{INC}_\infty(\alpha)} b^\beta$ . We are going to show that  $r_2$  must then be added to  $R$  by  $\text{PROPAGATE-RIGHT}$ . Let us suppose  $r_2$  is added to  $R$  by  $\text{PROPAGATE-LEFT}$ . This implies that  $\text{PROPAGATE-LEFT}$  is called twice in the two consecutive assembly steps. This then implies that two components are added from the left in the two consecutive assembly steps, since  $\text{PROPAGATE-LEFT}$  is called when the component at the next assembly step is added from the left. If this is the case, however, the second call of  $\text{PROPAGATE-LEFT}$  returns in line 20, without adding any propagation rules to  $R$ . This is a contradiction. Therefore,  $r_2$  must be added to  $R$  by  $\text{PROPAGATE-RIGHT}$ . Since  $r_1 = a^\alpha b^\beta \rightarrow a^{\text{INC}_\infty(\alpha)} b^\beta$  and no propagation rules of  $ab$  fire between  $r_1$  and  $r_2$ ,  $r_2$  must be of the form  $r_2 = a^{\text{INC}_\infty(\text{INC}_\infty(\alpha))} b^\beta \rightarrow a^{\text{INC}_\infty(\text{INC}_\infty(\alpha))} b^{\text{INC}_\infty(\beta)}$ . Hence  $\text{TRN}(r_1) \overset{\infty}{\rightsquigarrow} \text{TRN}(r_2)$ .

If  $r_1$  is added to  $R$  by PROPAGATE-RIGHT in line 29, we can write  $r_1$  as  $r_1 = a^\alpha b^\beta \rightarrow a^\alpha b^{\text{INC}_\infty(\beta)}$ . Similar discussion shows that  $r_2$  must be added to  $R$  by PROPAGATE-RIGHT, and  $r_2$  must be of the form  $r_2 = a^\alpha b^{\text{INC}_\infty(\text{INC}_\infty(\beta))} \rightarrow a^{\text{INC}_\infty(\alpha)} b^{\text{INC}_\infty(\text{INC}_\infty(\beta))}$ . Hence in this case also,  $\text{TRN}(r_1) \overset{\infty}{\rightsquigarrow} \text{TRN}(r_2)$ . ■

**Corollary 3.** *Let  $n \geq 3$ . For any basic subassembly sequence  $x$  which is an instance of an assembly template  $t \in \text{SEQ}(\{p\}) \setminus L(G_{\text{II}})$ , there exists a class II SA with  $n$  conformations which self-assembles  $x$  from a configuration that covers  $x$  if there exists a  $n$ -conformation transition cycle.*

*Proof.* Let  $x \in \text{SEQ}(\Sigma)$ . From Theorem 1, it suffices to show that MAKE-RULE-SET run with  $\text{INC}_n$ , is correct. Let  $R_\infty$  and  $R_n$  be the rule sets returned by MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) run with  $\text{INC}_\infty$  and with  $\text{INC}_n$ , respectively. We know that  $R_\infty$  and  $R_n$  contain exactly the same attaching rules since  $n \geq 3$  and that attaching rules increment conformation of a component at most twice. Let  $\langle r_\infty^1, r_\infty^2, \dots, r_\infty^k \rangle$  be a sequence of propagation rules for two adjacent components  $ab$ , as produced by MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) with  $\text{INC}_\infty$ . Also, let  $\langle r_n^1, r_n^2, \dots, r_n^l \rangle$  be a sequence of propagation rules for  $ab$ , as produced by MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) with  $\text{INC}_n$ , where  $l \leq k$ . By definition of  $\text{INC}_n$ ,  $r_\infty^i$  corresponds to  $r_n^{i \bmod l}$  for  $i = 1, 2, \dots, k$ . From Corollary 2, therefore,  $\text{TRN}(r_n^{i \bmod l}) \overset{\infty}{\rightsquigarrow} \text{TRN}(r_n^{(i+1) \bmod l})$  for  $i = 1, 2, \dots, k-1$ . Since there exists a  $n$ -conformation transition cycle,  $\text{TRN}(r_n^i)$  and  $\text{TRN}(r_n^j)$  are not conflicting for any  $i, j \in \{1, 2, \dots, l\}$ . Since this holds for propagation rules of any two adjacent components,  $R_n$  contains the rules which assemble  $x$  from a configuration that covers  $x$ . ■

**Corollary 4.** *There exists a three-conformation propagation cycle.*

*Proof.* In this proof, we write the prime symbol ( $'$ ) as  $p$ . Consider a sequence of conformational transition rules  $\langle r_1, r_2, r_3, r_4, r_5, r_6 \rangle$  where

$$\begin{array}{lll} r_1 = \Lambda \cdot \Lambda \rightarrow p \cdot \Lambda & r_2 = pp \cdot \Lambda \rightarrow pp \cdot p & r_3 = pp \cdot pp \rightarrow \Lambda \cdot pp \\ r_4 = p \cdot pp \rightarrow p \cdot \Lambda & r_5 = p \cdot p \rightarrow pp \cdot p & r_6 = \Lambda \cdot p \rightarrow \Lambda \cdot pp. \end{array}$$

By Definition 10, the above sequence is a three-conformation propagation cycle, since  $\text{INC}_3(\Lambda) = p$ ,  $\text{INC}_3(p) = pp$ , and  $\text{INC}_3(pp) = \Lambda$ . ■

**Theorem 7.** *For any basic subassembly sequence  $x$  that is an instance of an assembly template  $t \in \text{SEQ}(\{p\}) \setminus L(G_{\text{II}})$ , there exists a class II SA  $M$  with three conformations which self-assembles  $x$  from a configuration  $\Gamma$  that covers  $x$ . And  $M$  is an SA with the minimum number of conformations which self-assembles  $x$  from  $\Gamma$ .*

*Proof.* The first part follows from Corollaries 3 and 4. We prove the second part by showing that there exists no class II SA with two conformations which self-assembles  $x$  from  $\Gamma$ . Since  $\text{INC}_2(\text{INC}_2(\alpha)) = \alpha$  for  $\alpha \in \{p\}^*$ , any two conformational transition rules  $\rho_1$  and  $\rho_2$  are conflicting if  $\rho_1 \overset{2}{\rightsquigarrow} \rho_2$ . By Corollary 2, therefore, running MAKE-RULE-SET with  $\text{INC}_2$  always causes at least one conflict among propagation rules, since  $t \in \text{SEQ}(\{p\}) \setminus L(G_{\text{II}})$ . ■

**Example 11.** Consider  $\Sigma = \{a, b, c, d, e\}$  and  $x = (a((b(cd))e))$ . The subassembly sequence  $x$  is an instance of  $t_3 = (p((p(pp))p))$  in Example 9, and  $t_3 \in \text{SEQ}(\{p\}) \setminus L(G_{\text{II}})$ . A call of MAKE-RULE-SET( $x, \text{none}, \emptyset$ ) run with INC<sub>3</sub> returns with  $(ab'c''d''e, R)$  where  $R$  contains the following rules:  $c + d \rightarrow c'd$ ,  $b + c' \rightarrow bc''$ ,  $c'd \rightarrow c''d'$   $d' + e \rightarrow d''e$ ,  $c''d'' \rightarrow cd''$ ,  $bc \rightarrow b'c$ , and  $a + b' \rightarrow ab'$ . It is clear that  $M = (\Sigma, R)$  is an SA with three conformations which self-assembles  $x$  from the configurations that cover  $x$ , for example,  $\langle a, b, c, d, e \rangle$  and  $\langle a, b, b, c, d, d, e, e \rangle$ .

### 3. Discussion and future work

In this paper, an abstract model of self-assembling systems is presented, where assembly instructions of components are written as *conformational switches*—local rules that specify conformational changes of a component. The model, called self-assembling automaton, is a sequential rule-based machine that operates on one-dimensional strings of symbols. An algorithm is provided to construct a self-assembling automaton which self-assembles an one-dimensional string of distinct symbols in a given particular subassembly sequence. Classes of self-assembling automata are then defined based on classes of subassembly sequences in which the components self-assemble. For each class of subassembly sequences, the minimum number of conformations necessary to encode subassembly sequences of the class are provided. In particular, it is shown that three conformations for each component are enough to encode any subassembly sequence of a string with arbitrary length.

There are a number of extensions that should be incorporated into the theory of one-dimensional self-assembling automata presented in this paper.

- *Classes of SA based on nonbasic subassembly sequences.* The current definition of classes of SA is based on the classes of subassembly sequences of an one-dimensional string of *distinct* symbols, which are referred to as *basic subassembly sequences*. Many self-assembling systems in nature, however, often involve self-assembly of identical components. Therefore, the definition of SA based on classes of *nonbasic* subassembly sequences would be desirable.
- *Detaching rules.* Most biochemical reactions are bidirectional; if the reaction  $a + b \rightarrow ab$  is possible, the reverse reaction  $ab \rightarrow a + b$  is also possible. Hence the current definition of SA should be extended such that the rule set can also contain detaching rules; rules of the form  $a^\alpha b^\beta \rightarrow a^\alpha + b^\beta$ . Accordingly, the definition of self-assembly must be modified.
- *Self-assembly in higher dimensions.* Since the classification of SA presented in this paper is based on subassembly sequences, it can also be applied to self-assembly in higher dimensions. However, there are no concepts of geometry and topology since the classes are only developed for one-dimensional self-assembly. These concepts, as discussed in [9]

for example, must be incorporated in order to extend SA to higher dimensions.

### Acknowledgments

This work was partially supported by the National Science Foundation with a Presidential Young Investigator's grant (DDM-9058415) to the second author. Matchable funds for this grant had been provided by Schlumberger Inc. These sources of support are gratefully acknowledged. The authors would also like to acknowledge the valuable comments on the draft of this manuscript by the anonymous reviewers.

### References

- [1] B. Berger, P. W. Shor, L. Tucker-Kellog, and J. King, "Local Rule-based Theory of Virus Shell Assembly," in *Proceedings of the National Academy of Science, USA*, **91** (1994) 7732-7736.
- [2] S. Casjens and J. King, "Virus Assembly," *Annual Review of Biochemistry*, **44** (1975) 555-604.
- [3] M. B. Cohn, C.-J. Kim, and A. P. Pisano, "Self-assembling Electrical Networks: An Application of Micromachining Technology," in *Transducers '91: 1991 Sixth International Conference on Solid-State Sensors and Actuators* (Institute of Electrical and Electronics Engineers, San Francisco, CA, 1991).
- [4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms* (The MIT Press/McGraw-Hill, Cambridge, MA/New York, 1989).
- [5] R. A. Crowther, E. V. Lenk, Y. Kikuchi, and J. King, "Molecular Reorganization in the Hexagon to Star Transition of the Baseplate of Bacteriophage T4," *Journal of Molecular Biology*, **116** (1977) 489-523.
- [6] N. S. Goel and R. L. Thompson, *Computer Simulations of Self-organization in Biological Systems*, (Croom Helm, London, 1988).
- [7] K. Hosokawa, I. Shimoyama, and H. Miura, "Dynamics of Self-assembling Systems: Analogy with Chemical Kinetics," *Artificial Life*, **1** (1994) 413-427.
- [8] K. Hosokawa, I. Shimoyama, and H. Miura, "Two-dimensional Micro-self-assembly using the Surface Tension of Water," *Sensors and Actuators, A: Physical*, **57** (1996) 117-125.
- [9] K. Lindgren, C. Moore, and M. G. Nordahl, "Complexity of Two-dimensional Patterns," *Journal of Statistical Physics*, to appear.
- [10] J. C. Martin, *Introduction to Language and the Theory of Computation* (McGraw-Hill, New York, 1991).
- [11] P. H. Moncevicz, "Orientation and Insertion of Randomly Presented Parts using Vibratory Agitation," master's thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, 1991.

- [12] P. H. Moncevicz and M. J. Jakiela, "Method and Apparatus for Automatic Parts Assembly," United States Patent 5,155,895, October 20, 1992.
- [13] P. H. Moncevicz, M. J. Jakiela, and K. T. Ulrich, "Orientation and Insertion of Randomly Presented Parts using Vibratory Agitation," in *Proceedings of the ASME Third Conference on Flexible Assembly Systems*, A. H. Soni, editor, (The American Society of Mechanical Engineers, DE-Vol. 33, New York, September, 1991).
- [14] L. S. Penrose, "Self-reproducing Machines," *Scientific American*, **200** (1959) 105-114.
- [15] K. Saitou and M. J. Jakiela, "Automated Optimal Design of Mechanical Conformational Switches," *Artificial Life*, **2** (1995) 129-156.
- [16] K. Saitou and M. J. Jakiela, "Subassembly Generation via Mechanical Conformational Switches," *Artificial Life*, **2** (1995) 377-416.
- [17] R. L. Thompson and N. S. Goel, "A Simulation of T4 Bacteriophage Assembly and Operation," *BioSystems*, **18** (1985) 23-45.
- [18] R. L. Thompson and N. S. Goel, "Movable Finite Automata (MFA) Models for Biological Systems I: Bacteriophage Assembly and Operation," *Journal of Theoretical Biology*, **131** (1988) 351-385.
- [19] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, *Molecular Biology of the Gene* (Benjamin/Cummings, Menlo Park, CA, 1987).
- [20] G. M. Whitesides, "Self-assembling Materials," *Scientific American*, September, 1995, 146-149.
- [21] H. J. Yeh and J. S. Smith, "Fluidic Self-assembly of GaAs Microstructures on Si Substrates," *Sensors and Materials*, **6** (1994) 319-332.