String Tile Models for DNA Computing by Self-Assembly

DNA6 Preliminary Proceedings, May 12, 2000

Erik Winfree
Depts. of Computer Science and CNS
California Institute of Technology
Pasadena CA 91125, USA
winfree@caltech.edu
http://gg.caltech.edu/~winfree

Tony Eng
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
tleng@theory.lcs.mit.edu

Grzegorz Rozenberg
Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1, 2333 CA Leiden
The Netherlands
and
Department of Computer Science
University of Colorado at Boulder
Boulder, CO 80309, USA.
rozenber@wi.leidenuniv.nl

Abstract. This paper investigates computation by linear assemblies of complex DNA tiles, which we call string tiles. By keeping track of the strands as they weave back and forth through the assembly, we show that surprisingly sophisticated calculations can be performed with linear assembly. Examples range from generating an addition table to providing $O(1)$ solutions to CNF-SAT and DHPP. We classify the families of languages that can be generated by various types of DNA molecules, and establish a correspondence to the existing classes $ET0L_{ml}$ and $ET0L_{fin}$. Thus, linear self-assembly of string tiles can generate the output languages of finite-visit Turing Machines.

1 Introduction

Adleman's original work on molecular computation [Adl94] made use of self-assembly for an important step in the computation: the generation of DNA representing paths through a graph of vertices. This is a useful preprocessing step, reducing the set of all possible sequences of vertices to just a subset (the valid paths) that can be exponentially smaller. However, linear self-assembly of double-helical DNA appeared to be limited, prompting the suggestion to use the self-assembly of two-dimensional [Win96] and branched [WYS98] DNA structures for DNA-based computation. These suggestions were predicated on the complex synthetic DNA structures invented by Seeman for DNA nanotechnology [See82,See98]; there is now over a decade of experimental work with these molecules, including the recent demonstration of two-dimensional (2D) crystals and their modification [WLWS98,LYK$^+$00,MSS99,LSS99].

Self-assembly and branched DNA structures may be used in combination with other DNA computing techniques. Reif has proposed using step-wise self-assembly to reduce errors [Rei99]; the circuit satisfaction problem has a particularly elegant implementation in his model. Jonoska has considered the self-assembly of branched DNA into flexible graph-like structures, with applications to NP-complete problems [JKS99]. An interesting observation is that pre-formed branched DNA structures can provide advantages for subsequent processing by restriction enzyme digestion [JKS98]. Even knottedness can be used for computation; DNA Borromean rings implement a logical `AND` gate [SWY$^+$98]. We are a long way from understanding the full power of branched DNA and self-assembly.

However, these proposals still use complex DNA structures and assemblies that are likely to pose at least as many technical difficulties as 2D self-assembly. It has been suggested that fixed-width or one-dimensional (1D) self-assembly may be an attractive and robust experimental system, with faster self-assembly and lower error rates than two-dimensional systems [LWRar]. In this article we explore the computational power of 1D self-assembly of branched DNA structures.

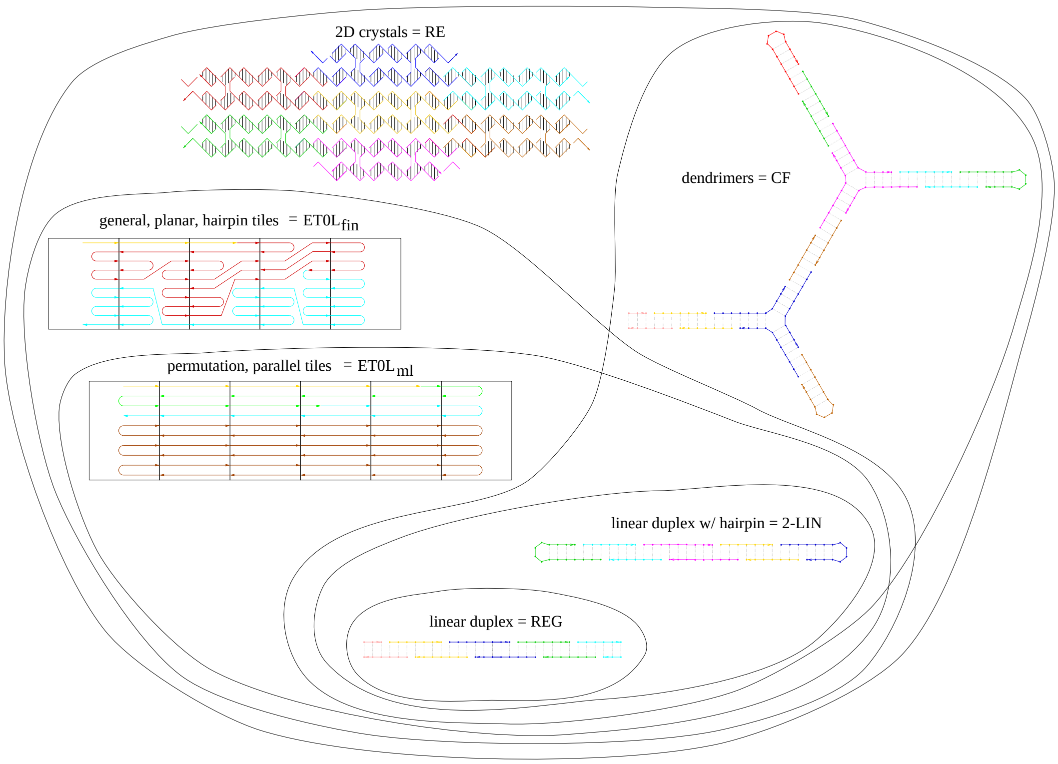

Fig. 1. The hierarchy of self-assembled languages, including the new results in this paper. Here REG is the family of regular languages, LIN is the family of linear languages (and k-LIN is the k-metalinear languages, where the axioms may have k non-terminal symbols), CF is the family of context-free languages, CS is the family of context-sensitive languages, and RE is the family of recursively enumerable languages.

2 DNA self-assembly and formal language theory

In DNA-based computing, a test tube of DNA oligonucleotides (equivalently, strands) is considered to represent a set of strings over an alphabet. A strand of DNA can be interpreted directly as a string over $\mathcal{D} = \{A, C, G, T\}$ (a DNA sequence) by reading its bases in the $5' \to 3'$ order. If S is a DNA sequence, then its Watson-Crick complement is written S'; e.g. $AGCTGCG' = CGCAGCT$. We will follow the convention that DNA strands may be taken to represent strings over a larger alphabet Σ by using a codebook $\mathcal{C} : \Sigma \to \mathcal{D}^N$; i.e., each symbol in Σ is represented by an N-base subsequence. It will always be assumed that for $\alpha \neq \beta$, occurrences of $\mathcal{C}(\alpha)$ and $\mathcal{C}(\beta)$ are guaranteed not to overlap in the DNA strands under consideration. Thus, a tube of DNA strands can be considered to represent a set of strings over the alphabet Σ. Because DNA strands can be circular, a test tube may also contain circular strings over $\mathcal{D}$ and Σ, represented using the prefix symbol $\circ$ as described later. Finally, note that the codebook defines a many-to-one relationship of $\mathcal{D}^*$ to Σ^*, because DNA subsequences that are not part of any codeword $\mathcal{C}(\alpha)$ are simply ignored. For example, using a codebook where $\mathcal{C}(a) = ACT, \mathcal{C}(b) = GAC$, both the strings $ACTGACGAC$ and $GTACTTTGGACGTGAC$ code for abb.

Formal languages, central to understanding computation on strings, provide a natural formalism for DNA based computing. There is a close correspondence between generative grammars and the self-assembly and ligation of DNA molecules. Both are processes that generate new strings from previous ones according to well-defined rules. As an example, we informally describe the self-assembly model of Winfree et al. [WYS98]. A multi-strand DNA molecule is represented as a graph, called a DNA *complex*, where each node (labelled from $\mathcal{D}$) represents a nucleotide, directed *backbone* edges represent covalent phospho-diester bonds,

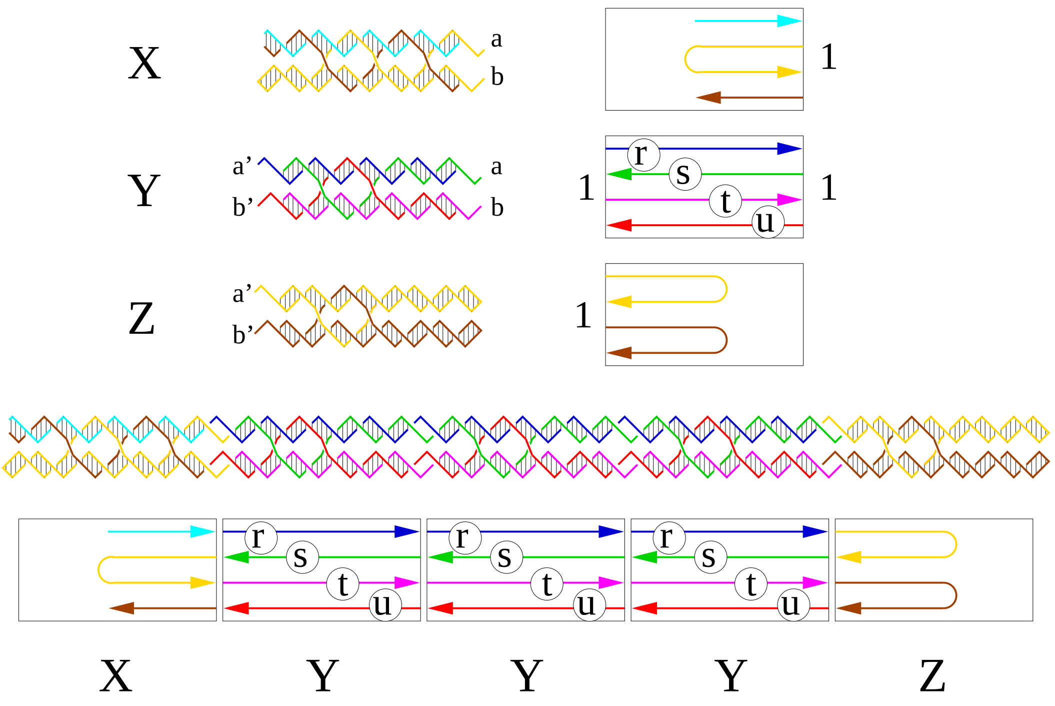

Fig. 2. Three DPE molecules that assemble to construct a language outside of context-free. Both molecular schematics and string tile diagrams are shown. Sticky-end sequences are indicated by a, b and their Watson-Crick complements a' and b'; the matching relationship is given by the binding label 1 on the string tiles. The coding sequences in the DNA strands are indicated by the edge labels r, s, t, and u in the string tiles.

and undirected *basepair* edges represent Watson-Crick base pairing. Self-assembly at "*temperature*" $\mathcal{T}$ starts with (an infinite supply of) a finite set of DNA complexes. Two DNA complexes with complementary sticky ends (of length $\geq \mathcal{T}$) can be joined to make a new complex. A DNA complex that cannot participate in further assembly is called a *maximal (or terminal) complex.* The set of strands remaining after ligation of all nicks in all terminal complexes is a DNA sequence language over $\mathcal{D}$, encoding (via the codebook $\mathcal{C}$) a language over Σ. For example, in the linear duplex assembly of Figure 1, the DNA sequences after ligation are the two strands $\{red\ yellow\ blue\ green\ cyan, cyan'\ green'\ blue'\ yellow'\ red'\}$, where color names indicate sequences.

Throughout this text we are interested only in the maximal, not the intermediate assemblies. That is, we assume all reactions go to completion, exhausting the supply of their reactants. Our motivation is partly to avoid treatment of concentrations and kinetics and equilibria, as would be necessary to study intermediates, and partly because the study of maximal assemblies allows for more elegant mathematics.

We are interested in computational structure-function relationships: what classes of DNA complexes (i.e. structures) give rise to what classes of languages (i.e. functions)? As is illustrated in Figure 1, the first natural classes of DNA self-assembly to be studied reproduced much of the Chomsky hierarchy for formal languages.

- Self-assembly of duplex DNA by single sticky-end adhesion generates regular languages by forming linear DNA complexes [WYS98].
- Self-assembly of hairpin and duplex DNA by single sticky-end adhesion generates 2-metalinear languages by forming linear DNA complexes (a modest generalization of [Eng99]).
- Self-assembly of hairpin, duplex, and 3-arm DNA by single sticky-end adhesion generates context-free languages by forming dendrimer DNA complexes [WYS98].

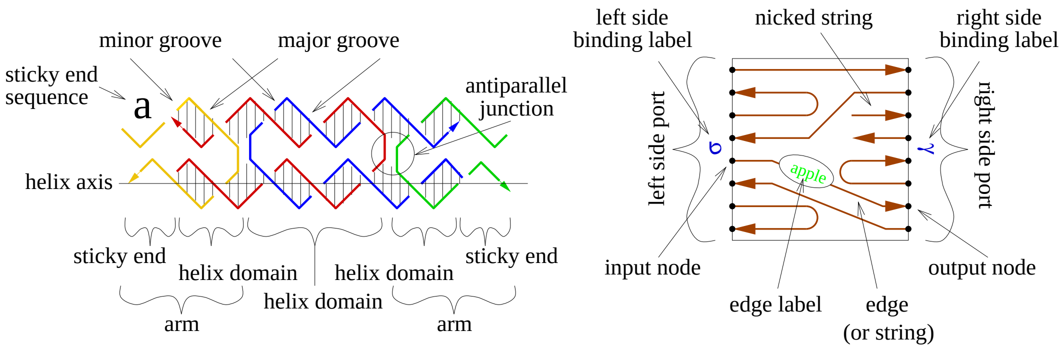

Fig. 3. Terminology for DNA multi-crossover structures and string tile diagrams. Arrowheads indicate the 3′ end of the DNA. Note that in the molecular schematic (showing a DAO molecule), the major/minor goove also indicates the 3′ and 5′ ends: the 3′ ends point away from the center of the narrow (minor) groove. The non-crossing strands in an antiparallel junction are antiparallel; in a parallel junction, they would be parallel. Hairpin strands begin and end on the same side of the tile or molecule.

– Self-assembly of DX units by double sticky-end adhesion, at a critical temperature that allows discrimination between single and double matches, can generate recursively enumerable languages by forming 2D DNA complexes [Win96,WYS98].

In this paper, we show two new classes:

– Self-assembly of DNA multi-crossover units by multiple simultaneous sticky-end adhesion generates $ET0L_{fin}$ languages by forming linear multi-helix DNA complexes.
– The above case, restricted to units where in each internal tile all coding strands cross from one side to the other side, generates $ET0L_{ml}$ languages.

Note that the context-sensitive languages have not yet arisen in DNA self-assembly.

3 Motivating examples

We develop the basic ideas by example, before introducing formal notation in Section 5.

3.1 A non-context-free language

We begin by considering linear assembly of double-crossover (DX) molecules [FS93], allowing hairpins on the arms. There are five types of DX molecules, classified according to whether the strands that don't cross at the junction are parallel (P) or anti-parallel (A) with each other, whether there are an even (E) or odd (O) number of half-turns between junctions, and whether the narrow (N) or wide (W) groove is in excess on the inside between the junctions: thus they are called DAE, DAO, DPE, DPON, DPOW.

If DPE units assemble into a linear array where each unit joins its neighbors via both of *two* sticky ends, then the resulting language of DNA sequences can be beyond context-free. Specifically, Figure 2 shows how to achieve $L = \{r^n s^n t^n u^n | n \geq 1\} \notin \mathcal{L}_{CF}$. The hairpins on the arms of X and Z allow the strands to turn around and return again through the tiles. Long-range correlations are possible due to the fact that the final DNA strand snakes through the assembly several times. The terminal DNA complexes assembled in this reaction are of the form XY^*Z, but the DNA strands encode sequences of the form $r^n s^n t^n u^n$. Thus a regular language of units yields a non-context-free language of DNA sequences.

The logic of language generation by self-assembly can be hard to see when complex multi-helical DNA structures are drawn; the situation is clarified by using *(linear) string tiles*, as shown in Figures 2 and 3 and described informally here (we give formal definitions in Section 5). The left and right sides are called

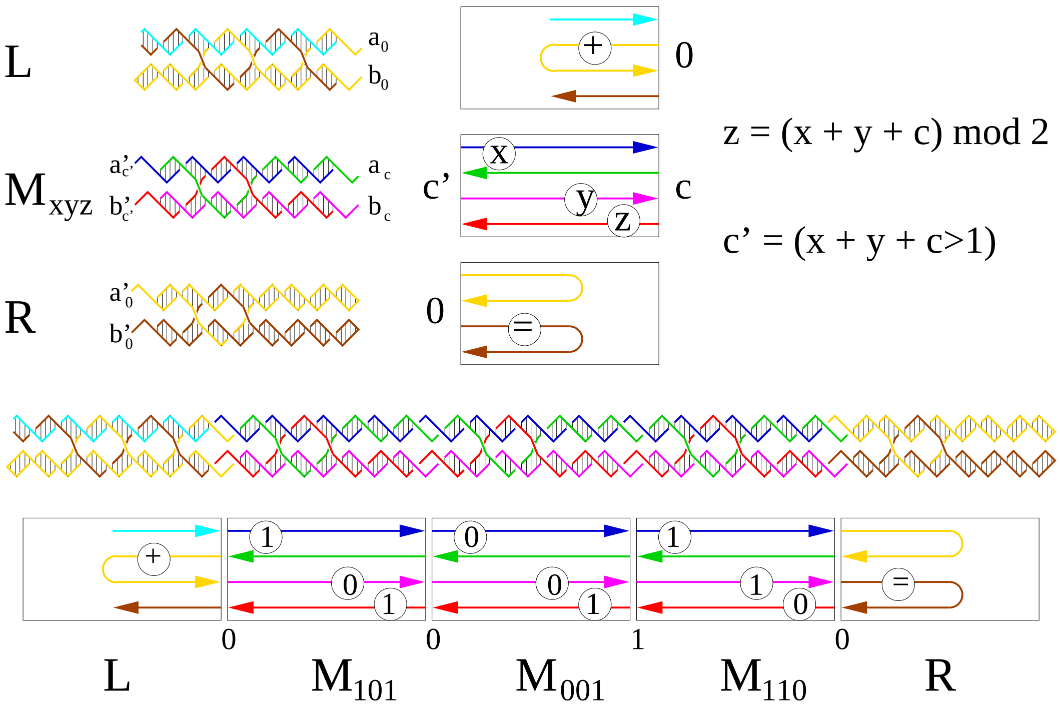

Fig. 4. Ten DPE molecules that assemble to construct a binary addition table. Here, x, y, z, c, and c' are binary variables; there is a one tile for every combination of x, y, z values. Thus, there are only four sticky-end sequences, a_0, a_1, b_0, b_1, and their Watson-Crick complements. Likewise, there are only two binding labels, 0 and 1. Matching binding labels are indicated below the sides of joined tiles in the assembly.

the *ports*. Each port may be labelled by a symbol or color, called the binding label, to indicate how tiles may be joined to each other. The ports also have several input and output nodes representing the $3'$ and $5'$ ends, respectively, of strands that can be ligated to strands in adjacent tiles. The input and output nodes within a tile are connected by edges (drawn as arrows) representing the DNA strands of the tile. A missing edge (drawn as an arrow that connects to or from nothing) indicates a nick in the DNA strand. Each edge is labelled by a string over the final alphabet to indicate the coding sequence on that strand. Tiles with matching binding labels may be joined (like dominoes); assemblies which permit no further additions are called maximal assemblies. When joined, the edges in maximal assemblies form either paths or cycles; the strings labelling the edges may be concatenated to form linear or circular strings, respectively. The collection of all such concatenated linear (circular) strings, for all maximal assemblies made from a given set of tiles, is called the linear (circular) language generated by the tile set.

3.2 Parallel tiles that generate an addition table

A more interesting example, building on [Rei99,LYK$^+$00], is generating a table of all addition input/output triples. ([LWRar] gives another implementation of this example, based on a preliminary draft of this paper.) As shown in Figure 4, the basic unit is still a DPE double crossover unit, each with a pair of sticky ends on the left and on the right. The sequences for these sticky ends are such that for any two units which bump into each other, either both sticky ends match (and the units may be joined) or both sticky ends don't match (and the units may not be joined). *Thus, a temperature that allows discrimination between a partial match and a total match is not necessary.*

How does this system work? The two possible sticky-end pairs represent the two possible carry-bit states during bitwise addition. Starting from the right, each new unit adds a new bit to **x** and a new bit

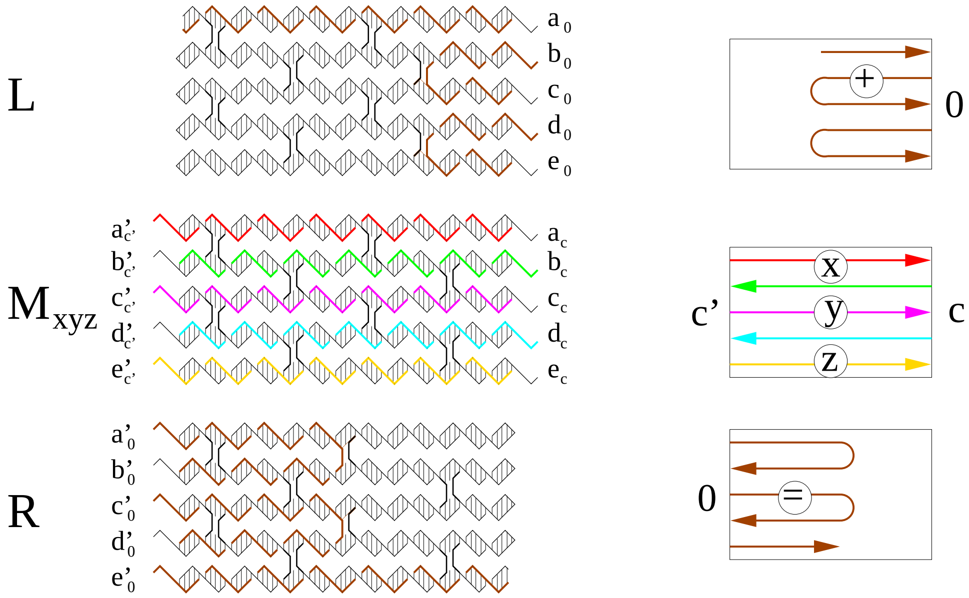

Fig. 5. Ten penta-crossover molecules that assemble to construct a binary addition table.

to $\mathbf{y}$ (thus there are always 4 possibilities) and the appropriate new bit to $\mathbf{z}$ (as a function of the previous carry and $\mathbf{x}$ and $\mathbf{y}$), terminating with the sticky-end pair for the appropriate carry bit (again as a function of the previous carry and $\mathbf{x}$ and $\mathbf{y}$). With the capping units, the final strand through the maximal assembly zigs first through $\mathbf{x}$, then zags through $\mathbf{y}$, and finally (in reverse order) through $\mathbf{z}$. The generated language is thus

$$L_{ADDREV} = \{\text{``}\mathbf{x} + \mathbf{y} = \mathbf{z}\text{''} : |\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| \text{ and } \#\mathbf{z}^R = \#\mathbf{x} + \#\mathbf{y}\}$$

where $\mathbf{z}^R$ gives the string $\mathbf{z}$ in reverse-order and $\#\mathbf{x}$ gives the integer represented by the binary string $\mathbf{x}$. We use quotation marks to emphasize that the contained symbols are just symbols; i.e., "+" is a symbol and not a mathematical operator in this context.

There are two potential drawbacks to the scheme illustrated in Figure 4. First, we note that [FS93] found that parallel variants (DPE, DPON, DPOW) of short double crossover molecules are less stable than the antiparallel (DAE, DAO) variants. Long arms in our DPE should stabilize the parallel structures, but it is worth investigating whether the string tiles can be implemented using only antiparallel structures. Second, the reversed output string may indicate a limitation to the string tile approach. As we see in the next example, it does not.

In fact, we can avoid both the reversed $\mathbf{z}$ string and the parallel DX molecules by using larger multi-crossover units, thus generating exactly

$$L_{ADD} = \{\text{``}\mathbf{x} + \mathbf{y} = \mathbf{z}\text{''} : |\mathbf{x}| = |\mathbf{y}| = |\mathbf{z}| \text{ and } \#\mathbf{z} = \#\mathbf{x} + \#\mathbf{y}\}.$$

Figure 5 shows string tiles built from antiparallel quintuple-axis DNA molecules. Note that we are ignoring the black strands, which will not code for anything according to the codebook, and thus are ignored in the final language. In this system, each adhesion event now involves multiple sticky ends, but again no sensitive discrimination is required – either all sticky ends match, or none do.

3.3 Hairpin tiles for CNF-SAT

We now show a use for tiles (other than cap tiles) whose coding strands involve hairpins: they allow CNF-SAT problems to be solved in $O(1)$ biosteps using linear arrays of DNA multi-crossover units. The

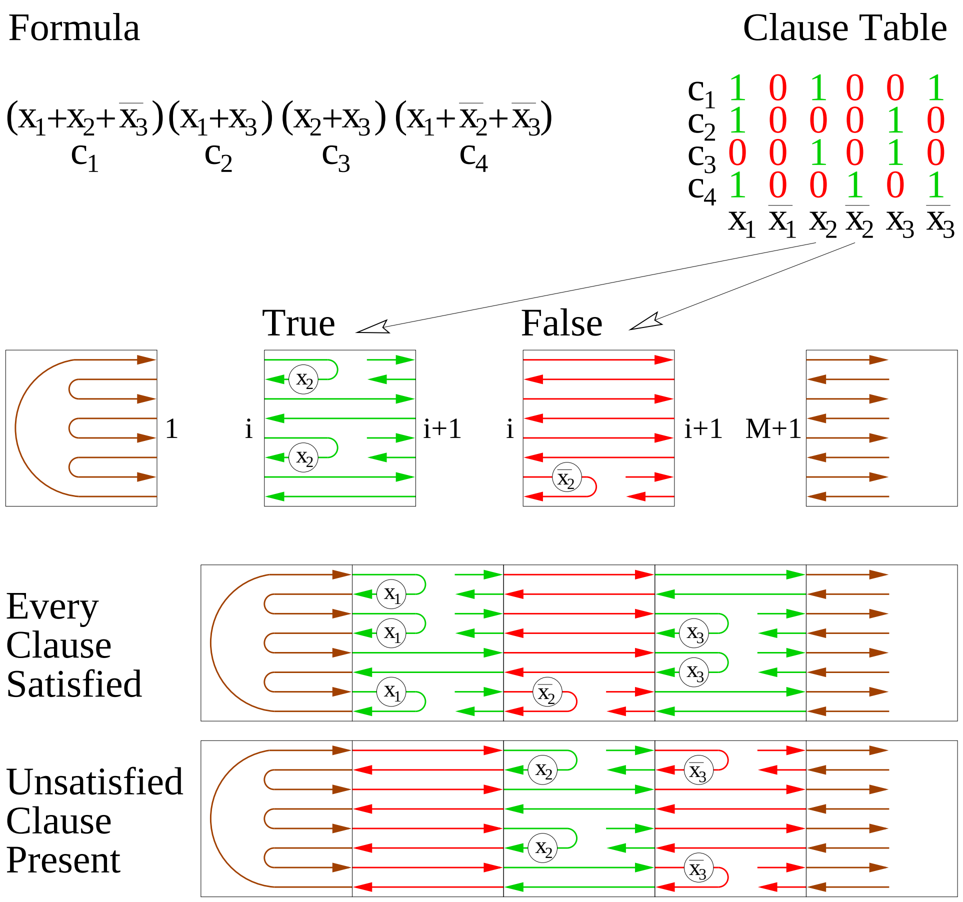

Fig. 6. Solving an N-clause M-variable CNF-SAT problem by linear assembly of $2M + 2$ width-N strings tiles. Shown are the cap tiles and the tiles for x_i and for $\overline{x_i}$, $i = 2$. A circular strand indicates the solution to the problem. Maximal assemblies are all of length $M + 2$.

solution we present here, using linear self-assembly, may (or may not) be faster and more robust than the two-dimensional self-assembly of [Win96,LLar], which is also sufficient to solve this problem in $O(1)$ biosteps.

The main idea is as follows: A CNF-SAT problem of N clauses and M variables is solved using and initial set of $2M + 2$ hairpin tiles of width N, which assemble to form all 2^M distinct tile assemblies of length $M+2$. To isolate a solution to the problem, one additional operation is required: after assembly and ligation of the DNA, we select for circular strands (for example, by 2D gel electrophoresis or by exonuclease digestion). The formula is satisfiable iff a circular strand is present, as it gives a solution to the CNF-SAT problem.

More specifically, each clause C_i is a disjunction of literals from $\{x_1, \ldots, x_M, \overline{x_1}, \ldots, \overline{x_M}\}$. The entire CNF formula, then, can be represented as a clause table C where each row signifies a clause, and each column signifies a literal. Each tile will represent a column of entries in C. Thus there are two tiles for each variable j, "True" (green) and "False" (red). "True" has a hairpin in each row i such that the literal x_j appears in clause i, and "False" has a hairpin in each row i such that the literal $\overline{x_j}$ appears in clause i. In any terminal assembly made from the tiles shown in Figure 6, the i^{th} row (helix axis) has a hairpin in the j^{th} column (tile) iff the assignment of "True" or "False" to variable x_j has satisfied clause i. Therefore, the strand starting in the left cap tile is circular iff every clause is satisfied. The actual satisfying assignment is deduced from the intervening sequence. (By construction, no other strands in the assembly can be circular.)

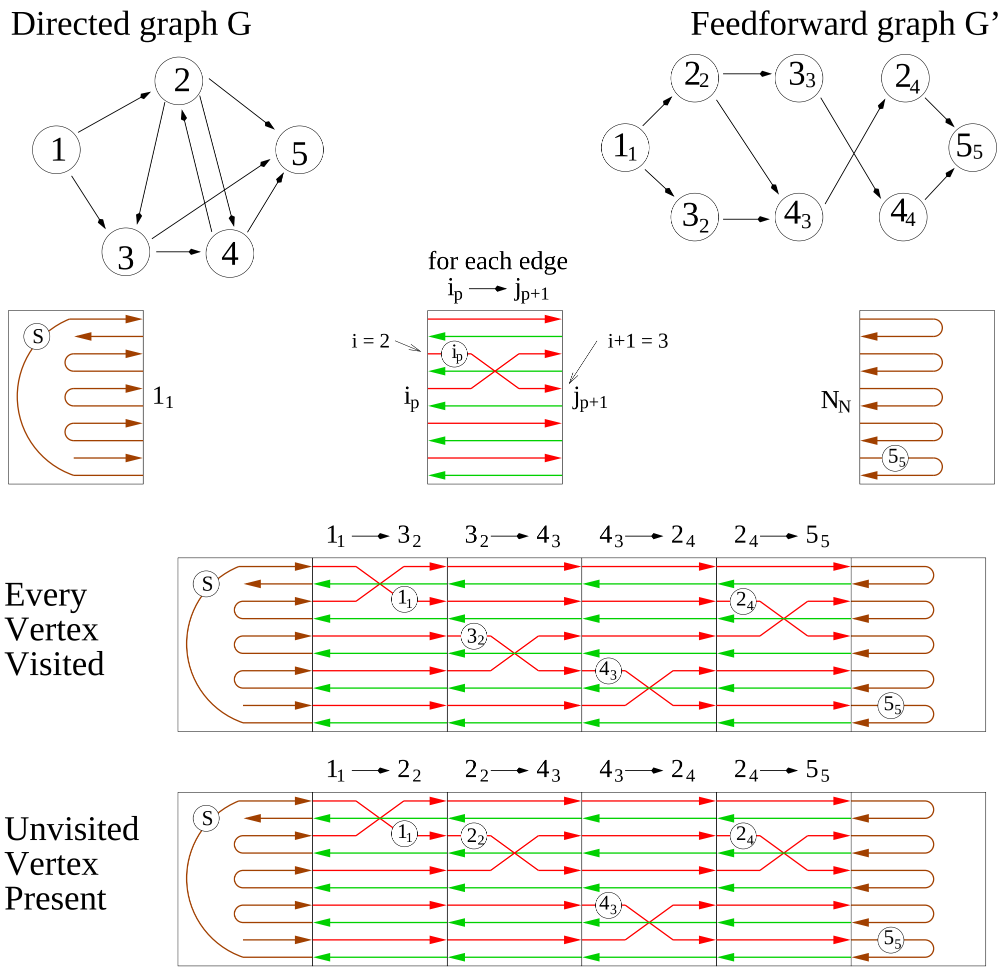

Fig. 7. $O(N^2)$ width-N strings tiles to solve an N-vertex Directed Hamiltonian Path Problem. Maximal assemblies are all of length $N+1$.

3.4 Permutation tiles for DHPP

As another example of solving an NP-complete problem in $O(1)$ biosteps using linear string tiles, we solve the Directed Hamiltonian Path Problem (DHPP) using permutation tiles – that is, tiles without hairpins, but where the strands may be re-routed as they cross the tile (that is, their order is permuted).

First, as a polynomial time preprocessing step, we convert the directed N-vertex graph G into a feed-forward version G' with up to N^2 vertices. The vertex at layer position p of G' corresponding to vertex i in G is labelled i_p. As shown in Figure 7, we have a permutation tile for each edge $i_p \to j_{p+1}$ in the new graph; this tile has a single pair of non-parallel arrows from left input i (respectively $i+1$) to right output $i+1$ (respectively i). Binding labels are such that each maximal assembly of tiles corresponds to a path from 1_1 to N_N in G'. Thus, starting at S in a tile assembly representing a particular length-N path through G, the strand can advance from row i to row $i+1$ only if vertex i is visited at some point. If an unvisited vertex is present, there is a row the strand cannot advance beyond, and consequently it must terminate at the $3'$ nick. (Note that the other strands may form a circle.) If there is no unvisited vertex, in which case the assembly represents a Hamiltonian path, the strand containing S is circular. Again, we isolate circular strands, and from those strands we further must extract strands containing S.

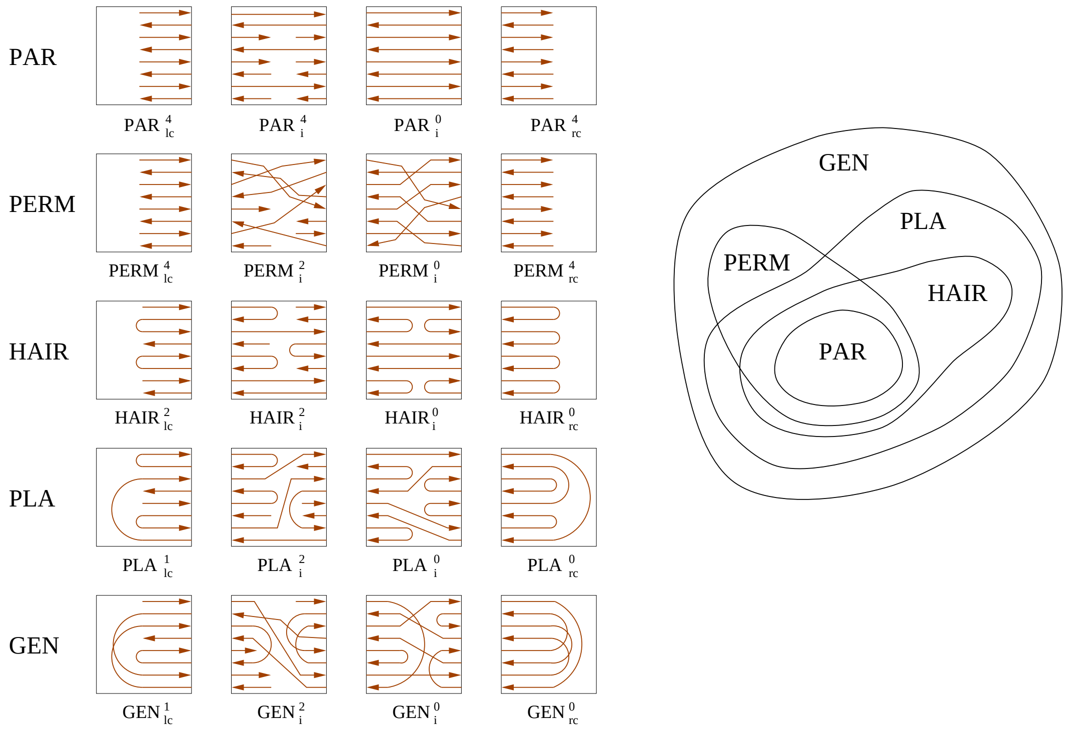

Fig. 8. The five tile classes: parallel tiles (PAR), permutation tiles (PERM), hairpin tiles (HAIR), planar tiles (PLA), and general tiles (GEN). The left (lc) and right (rc) cap tiles of each class are shown, and the number of nicks is indicated in the superscript. (left) Example tiles. (right) Tile class inclusion diagram.

Note that DHPP could also be solved with hairpin tiles as was CNF-SAT, and conversely CNF-SAT could be solved with permutation tiles; this is left as an exercise to the reader.

4 Constructibility of string tiles

The CNF-SAT and DHPP examples made use of string tiles for which no explicit DNA structures were given. Can we construct DNA molecules for the string tiles in question? We approach this first by informally defining several classes of (simpler and then more complex) string tiles, and then demonstrating a procedure to build complex string tiles from simpler ones.

4.1 Classes of string tiles

As illustrated in Figure 8, we classify linear string tiles into *parallel tiles*, in which the i^{th} input node on one side is connected to the i^{th} output node on the other side; *permutation tiles*, in which nodes must be connected to nodes on the opposite side only, but in any order; *hairpin tiles*, which are like parallel tiles with the additional possibility that an input node may be connected to an adjacent output node on the same side; *planar tiles* in which the strands do not cross, as drawn on the tiles; and *general tiles*, in which any connections are allowed. These classes are subdivided into left and right *cap tiles*, which have one unlabelled side, and into tiles with a given number of nicks (i.e., internal $3'$ and $5'$ ends, measured in pairs).

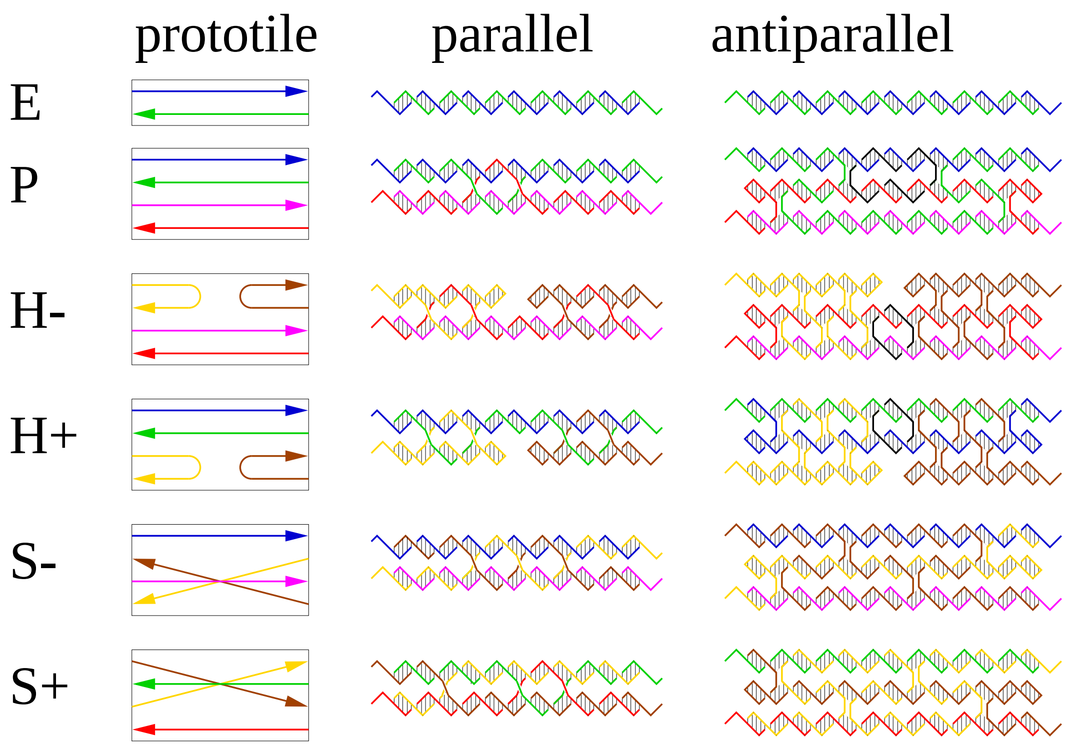

Fig. 9. Prototile sets (using only parallel junctions or only antiparallel junctions) from which all general linear string tiles can be built.

4.2 Criteria for constructibility; prototiles

We might ask at this point, can general tiles of significant complexity actually be made out of DNA? To be a useful implementation of a string tile, a proposed DNA complex must

- be geometrically compatible with DNA molecular structure,
- have strand routing identical to that in the string tile,
- be "rigid" as a molecule, so that each helical domain remains parallel to the other helical domains,
- self-assemble reliably from the individual strands, when mixed according to some protocol.

For example, consider the two sets of tiles proposed for generating the addition table. In both sets, each double helix domain is connected to each neighboring domain by at least two junctions, ensuring that the axes will be parallel in the molecule and that the molecule will be rigid. The first set (Figure 4) was implemented with DPE molecules, which have been characterized in the laboratory [FS93], so they are known to self-assemble from the four component strands (although with questionable reliability). The second set (Figure 5) was implemented with hypothetical quintuple-axis molecules, which have not been experimentally demonstrated yet, although they are likely to be feasible (N. Seeman, private communication). In fact, it is unlikely that the internal tiles would spontaneously self-assembly from their 10 component strands, unless the long black strand were broken into several shorter strands; but this is a question to be answered by experiment. Direct assembly of tiles from component strands poses an even greater difficulty for larger string tiles.

Therefore, we pursue an approach where larger string tiles are assembled from a small set of *prototiles*, which consist of (or are very similar to) molecules that have already been characterized experimentally. In fact, we give two possible implementations for each prototile, one using parallel junctions (essentially DPE molecules [FS93]) and one using anti-parallel junctions (essentially triple-crossover molecules

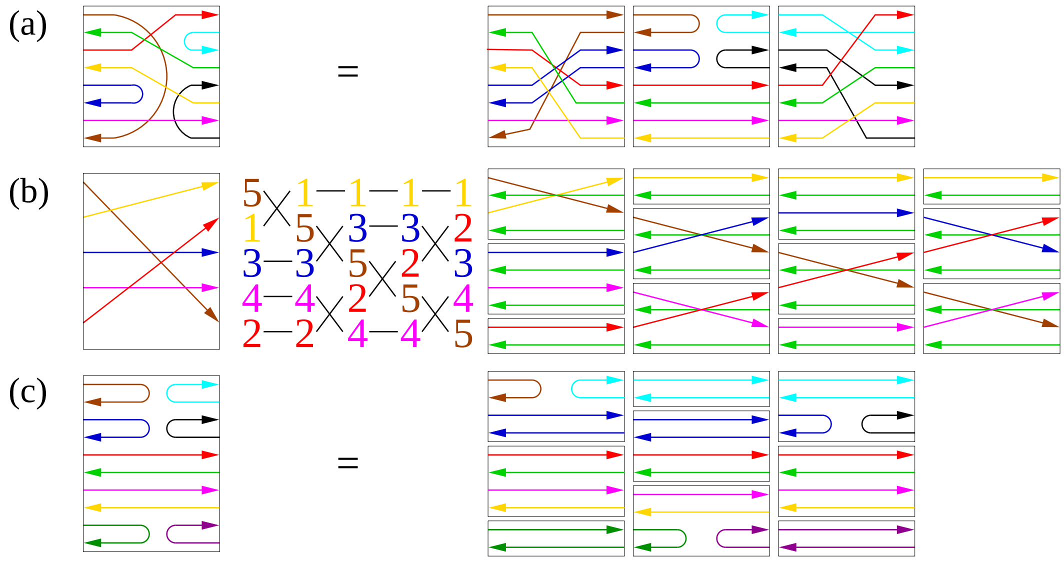

Fig. 10. (a) A general string tile with k hairpins can be created from two permutation tiles and a hairpin tile with k hairpins. (b) A permutation tile can be built from the $E, P, S+, S-$ prototiles. Here we show the construction for the rightward arrows; a similar construction routes the leftward arrows. (c) A hairpin tile with k hairpins can be construction from the $E, P, H+, H-$ prototiles.

[LYK$^+$00,LWRar]). Thus, the constructions below show that either parallel or anti-parallel junctions alone are sufficient for implementing all general tiles. The six prototiles are shown in Figure 9; note that they are all rigid.

4.3 Constructions

We construct a general tile in three steps. First (Figure 10a), we observe that every general tile is the composition of a permutation tile, a hairpin tile, and a permutation tile.

Second (Figure 10b), we show that permutation tile can be built from the $E, P, S+, S-$ prototiles. To see how to arrange the prototiles, we make a list of the desired destination for the left input nodes and sort this list using the Even-Odd Transposition Sort, which is guaranteed to finish within N rounds [Knu73]. Wherever we performed a swap, we place an $S+$ prototile; where we didn't swap, we place a P prototile; and we use the E prototile for untested positions. This correctly routes the rightward arrows, without affecting the leftward arrows. A similar procedure can be done to route the leftward arrows without altering the rightward arrows. Once the proper arrangement of the prototiles is determined, the actual DNA molecules can be made with unique sticky ends for the prototile in each position; each of these prototiles can be assembled from their component strands in separate reactions, and then mixed to form the entire string tile.

Third (Figure 10c), we note that hairpins tiles of the form required (hairpins within a helix axis, but not between them) can be built from the $E, P, H+, H-$ prototiles. Three columns of prototiles are always sufficient.

5 Generative power of string tiles

We use a formal language theory approach to analyze the generative power of string tile assembly. We first give a general definition of string tiles, and then specialize to the fixed-width linear string tiles used in this paper.

5.1 Preliminaries

λ represents the empty string. From a given alphabet Φ we construct a distinct *barred* alphabet $\bar{\Phi} = \{\bar{\sigma} : \sigma \in \Phi\}$. Thus, $\Phi \cap \bar{\Phi} = \emptyset$ and, treating $^{-}$ as an operator, $\bar{\bar{\sigma}} = \sigma$.

A *circular string* over Σ is a finite set $c \subset \Sigma^+$ such that $(ab \in c \Rightarrow ba \in c)$ and $(x, y \in c \Rightarrow \exists a, b : x = ab$ and $y = ba)$. I.e. c consists of all circular permutations of a given string. If $x \in c$, the shorthand $\circ x$ denotes c.

A *directed graph* with edges labeled over Σ is a pair $g = (V, E)$ where $E \subseteq V \times \Sigma^* \times V$. We use the notation V_g and E_g to refer to the vertices V and edges E of g, respectively. Two graphs g_1 and g_2 are *disjoint* if $V_{g_1} \cap V_{g_2} = \emptyset$.

A *maximal path* π in graph g is a list $\pi = v_1 v_2 \cdots v_{k+1}$ where for $1 \leq i \leq k$, $(v_i, s_i, v_{i+1}) \in E_g$, and in-deg$(v_1)$=0, out-deg$(v_{k+1})$=0, and $v_i = v_j \Rightarrow i = j$. In that case, $word(\pi) = s_1 s_2 \cdots s_k$.

A *cycle* π in graph g is a list $\pi = v_1 v_2 \cdots v_{k+1}$ where for $1 \leq i \leq k$, $(v_i, s_i, v_{i+1}) \in E_g$ and $v_1 = v_{k+1}$, and $v_i = v_j \Rightarrow i = j$ for $1 \leq i, j \leq k$. In that case, $cword(\pi) = \circ s_1 s_2 \cdots s_k$.

Let g_1 and g_2 be disjoint graphs over Σ and let $A_1 = \{a_{1,1}, \ldots, a_{1,n}\} \subseteq V_{g_1}$, $A_2 = \{a_{2,1}, \ldots, a_{2,m}\} \subseteq V_{g_2}$, $B_1 = \{b_{1,1}, \ldots, b_{1,m}\} \subseteq V_{g_1}$, and $B_2 = \{b_{2,1}, \ldots, b_{2,n}\} \subseteq V_{g_2}$ be disjoint ordered subsets. Then the *join* of g_1 and g_2 using A_1, A_2, B_1, and B_2 is

$$g_1 \;{}_{A_1,B_1}{+}_{A_2,B_2}\; g_2 = (V_{g_1} \cup V_{g_2}, E_{g_1} \cup E_{g_2} \cup \bigcup_{i=1}^{n} \{(A_{1,i}, \lambda, B_{2,i})\} \cup \bigcup_{i=1}^{m} \{(A_{2,i}, \lambda, B_{1,i})\}).$$

That is, we add unlabeled edges from nodes in A_1 and A_2 to the respective nodes in B_2 and B_1.

5.2 String Tiles

A *port* (over the alphabet Φ) of graph g is a triple $p = (\sigma, I, O)$ where

- $\sigma \in \Phi \cup \bar{\Phi} \cup \{\lambda\}$ is called the *binding label*,
- $I \subseteq V$ is an ordered set of *input nodes* with in-deg 0,
- $O \subseteq V$ is an ordered set of *output nodes* with out-deg 0,
- $I \cap O = \emptyset$,
- $\sigma = \lambda$ iff $I = O = \emptyset$, in which case the port is said to be *empty*.

We use the notation σ_p, I_p, and O_p to refer to the binding label σ, input nodes I, and output nodes O of p, respectively. Two ports P_1 and P_2 are *disjoint* if I_{P_1}, O_{P_1}, I_{P_2}, and O_{P_2} are mutually disjoint.

A (k-sided) *string tile* over Σ, Φ is a pair $t = (P, G)$ where

- G is a directed graph over Σ with all nodes having in-deg ≤ 1 and out-deg ≤ 1,
- $P = \{p_1, p_2, \ldots, p_k\}$ is a set of mutually disjoint ports over Φ of G.

Here, Σ is the alphabet for string labels (and thus the alphabet for the language generated by the tiles) while Φ is the alphabet for the binding labels (which are relevant only for the self-assembly process). We use the notation P_t and G_t to refer to the ports P and graph G of t, respectively. When t is understood, σ_n, I_n^i and O_n^i refer to the binding label, the i^{th} input node and i^{th} output node of the port indexed by n, respectively. Two string tiles are *disjoint* if their graphs are disjoint. Note that the connected components of G are paths and cycles.

A string tile t is *of width* w if $|I_p| = |O_p| = w$ for all ports $p \in P_t$. A string tile is *uniform* if it is of width w for some w. A string tile t is *primitive* if $\bigcup_{p \in P_t} I_p \cup O_p = V_{G_t}$; i.e. every vertex is in a port. Note that in a primitive string tile, all edges are of the form (I_n^i, s, O_m^j). If a tile is not primitive, it is *composite*. See Figure 11a.

Let t_1 and t_2 be disjoint string tiles over Σ, Φ and let p_1 and p_2 be ports of t_1 and t_2 respectively. Then we say that t_1 and t_2 are *p_1-p_2-compatible* if $\bar{\sigma}_{p_1} = \sigma_{p_2} \neq \lambda$, $|I_{p_1}| = |O_{p_2}|$, and $|O_{p_1}| = |I_{p_2}|$. For p_1-p_2-compatible tiles t_1 and t_2, the *p_1-p_2-composition* is

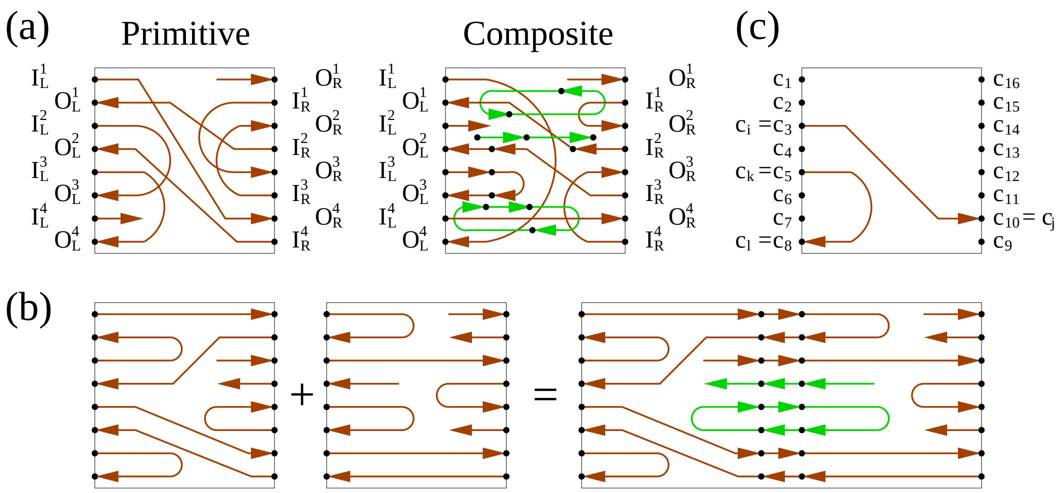

Fig. 11. (a) Diagram of a linear string tile (width 4). Note the order of the input and output nodes. The edge labels are not shown; nor are the binding labels. The half-edges indicate nodes that *aren't* involved in edges. (b) Illustration of the composition of tiles. (c) Numbering used in definition of planar tiles.

$$t_1 \;{}_{p_1}{+}_{p_2}\; t_2 = (P_{t_1} \cup P_{t_2} \setminus \{p_1, p_2\}, G_{t_1} \;{}_{O_{p_1}, I_{p_1}}{+}_{O_{p_2}, I_{p_2}}\; G_{t_2}).$$

Note that it is easily verified that $t_1 \;{}_{p_1}{+}_{p_2}\; t_2$ is indeed a string tile. If p_1 and p_2 are understood, as they are for all linear string tiles (below), then they are omitted as subscripts for notational convenience. See Figure 11b.

5.3 Classes of tiles

A *linear* string tile is a 2-sided string tile such that one port (the *left* port, by convention indexed by L) is labelled from $\Phi \cup \{\lambda\}$ and the other port (the *right* port, indexed by R) is labelled from $\bar{\Phi} \cup \{\lambda\}$.

If t is a linear string tile, then it is called a *complete* tile if $\sigma_L = \lambda = \sigma_R$; it is called a *left cap* tile if $\sigma_L = \lambda \neq \sigma_R$; it is called a *right cap* tile if $\sigma_L \neq \lambda = \sigma_R$; and it is called an *internal* tile if $\sigma_L \neq \lambda \neq \sigma_R$. The intuition here is that ports with binding label λ cannot be extended.

A linear string tile with n input nodes of degree 0 and m output nodes of degree 0 is said to be n,m*-nicked*; if $n = m$, the tile is said to have n *nicks*. Note that a width w string tile always has $n = m$.

Let SET be a set of uniform linear string tiles, then SET^n is the subset of tiles with n or fewer nicks; SET_{lc} is the subset of left cap tiles; SET_{rc} is the subset of right cap tiles; SET_i is the subset of internal tiles. This notation is illustrated in Figure 8.

Let the *general tiles*, GEN, be the set of all uniform primitive linear string tiles. The *parallel tiles*, PAR, the *permutation tiles*, $PERM$, the *hairpin tiles*, $HAIR$, and the *planar tiles*, PLA, are subsets of GEN, defined as follows (and see Figures 8 and 11c). Let $t \in GEN$ be of width w. Then

- $t \in PAR$ iff $(I^i_n, s, O^j_m) \in E_{G_t} \Rightarrow i = j$ and $(n,m) \in \{(L,R),(R,L)\}$,
- $t \in PERM$ iff $(I^i_n, s, O^j_m) \in E_{G_t} \Rightarrow (n,m) \in \{(L,R),(R,L)\}$,
- $t \in HAIR$ iff $(I^i_n, s, O^j_m) \in E_{G_t} \Rightarrow$ either $i = j$ and $(n,m) \in \{(L,R),(R,L)\}$
  or $i - j \in \{\pm 1\}$ and $(n,m) \in \{(L,L),(R,R)\}$,
- $t \in PLA$ iff for $i < k$, $(c_i, s, c_j), (c_k, s', c_l) \in E_{G_t} \Rightarrow$ either $i < k, l < j$ or $i, j < k, l$ or $l < i, j < k$, where $(c_1, c_2, \ldots, c_{4W})$ is the counterclockwise list of port nodes $(I^1_L, O^1_L, I^2_L, O^2_L, \ldots, I^2_R, O^2_R, I^1_R, O^1_R)$.

5.4 Assemblies and languages

Let T be a finite set of width w linear string tiles. The string over T, $\alpha = t_1 t_2 t_3 \cdots t_n$, with $n \geq 1$ and $t_i \in T$ for all $1 \leq i \leq n$, is an *assembly* over T if either $n = 1$ or t_i and t_{i+1} are compatible for all $1 \leq i < n$. Furthermore, α is a *maximal assembly* over T if α is not a proper substring of any other assembly over T. $\mathcal{A}(T)$ is the set of all maximal assemblies over T. Note that $\mathcal{A}(T)$ is always a regular language.

Each assembly α *induces* a single tile $t_\alpha = t'_1 + t'_2 + t'_3 + \cdots + t'_n$, where t'_i is a unique isomorphic copy of t_i (required for distinctness). Note that for $n > 1$, this tile is always *not* a primitive tile!

A tile t (and thus an assembly α) *induces* a set of linear strings

$$L(t) = \{word(\pi) : \pi \text{ is a maximal path in } G_t\}$$

and a set of circular strings (only possible if the tile is composite)

$$C(t) = \{cword(\pi) : \pi \text{ is a cycle in } G_t\}.$$

Then, for T a finite set of linear string tiles, the linear and circular languages generated by T are, respectively,

$$L(T) = \bigcup_{\alpha \in \mathcal{A}(T)} L(t_\alpha)$$

and

$$C(T) = \bigcup_{\alpha \in \mathcal{A}(T)} C(t_\alpha).$$

Thus, we arrive at the family of languages generated by a (possibly infinite) set of linear string tiles, SET:

$$\mathcal{L}_{ST}(SET) = \{L(T) : T \subset SET \text{ is finite }\}$$

and

$$\mathcal{C}_{ST}(SET) = \{C(T) : T \subset SET \text{ is finite }\}.$$

ST stands for "String Tile". If $SET1, SET2$, and $SET3$ are sets of tiles, then, for notational convenience, we refer to the family of languages $\mathcal{L}_{ST}(SET1_{lc} \cup SET2_i \cup SET3_{rc})$ as $\mathcal{L}_{ST}(SET1, SET2, SET3)$.

5.5 Scattered linear grammars for parallel string tiles

We will show that the family of languages generated by parallel tiles is a subclass of the languages generated by scattered context grammars. A *scattered context grammar* (see [RS97], page 128) is a quadruple $G = (\Sigma_N, \Sigma_T, P, S)$. Σ_N are the non-terminal symbols and Σ_T are the terminal symbols. S is a finite set of axiom strings over $\Sigma = \Sigma_N \cup \Sigma_T$ and P is a finite set of vector productions of the form

$$\langle A_1, A_2, \ldots, A_n \rangle \to \langle x_1, x_2, \ldots, x_n \rangle$$

where $A_i \in \Sigma_N$ and $x_i \in \Sigma^*$ and $n > 0$. A single derivation step, using the above production, transforms the string

$$\eta_1 A_1 \eta_2 A_2 \eta_3 \cdots \eta_n A_n \eta_{n+1} \to \eta_1 x_1 \eta_2 x_2 \eta_3 \cdots \eta_n x_n \eta_{n+1}$$

where $\eta_i \in \Sigma^*$. That is, we are performing synchronized application of context-free productions. The language generated by G is $L(G) = \{x : s \to^* x \text{ and } s \in S\}$ where $\to^*$ is the symmetric, transitive closure of $\to$.

We define a *scattered n-metalinear grammar* to be a scattered context grammar restricted to axiom strings of the form

$$x_1 A_1 x_2 A_2 x_3 \cdots x_n A_n x_{n+1}$$

and to productions of the form

$$\langle A_1, A_2, \ldots, A_n \rangle \to \langle x_1 B_1 y_1, x_2 B_2 y_2, \ldots, x_n B_n y_n \rangle$$

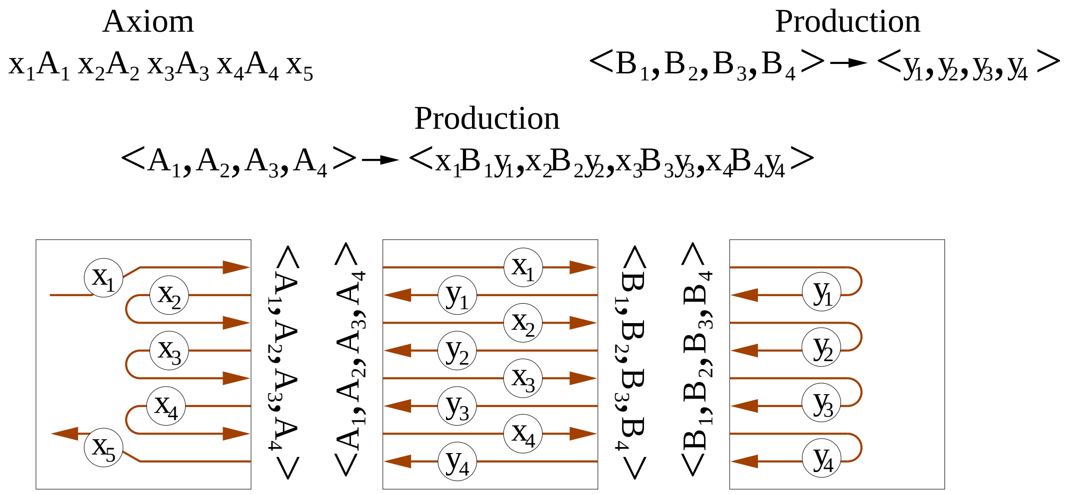

Fig. 12. The correspondence between a scattered grammar and parallel string tiles with $HAIR^1_{lc}$ and $HAIR^0_{rc}$.

and

$$\langle B_1, B_2, \ldots, B_n\rangle \rightarrow \langle y_1, y_2, \ldots, y_n\rangle$$

where $A_i, B_i \in \Sigma_N$ and $x_i, y_i \in \Sigma_T^*$. That is, we are requiring synchronized application of linear productions to a string with n non-terminals. $\mathcal{L}_{SM}$ is the family of languages generated by a scattered n-metalinear grammars, for some n.

Figure 12 shows a one-to-one correspondence of axioms to left cap tiles, productions to parallel tiles and right cap tiles, such that the language generated by the grammar is identical to the (linear) language generated by the tiles. In this way, it is straightforward to prove that

Theorem 1. $\mathcal{L}_{SM} = \mathcal{L}_{ST}(HAIR^1, PAR^0, HAIR^0)$.

5.6 Parallel Normal Form for Permutation Tiles

We will sketch the main ideas needed to prove that the languages generated by permutation tiles are the same as those generated by parallel tiles (although fewer permutation tiles may be required).

We use a two-step process for converting a finite set of permutation tiles (with general caps) into a finite set of parallel tiles (with hairpin caps) that generate the same language (Figure 13). Let $T_0 \subset GEN_{lc} \cup PERM_i \cup GEN_{rc}$; we will define $T_1 \subset GEN^1_{lc} \cup PERM^0_i \cup GEN^0_{rc}$ and $T_2 \subset HAIR^1_{lc} \cup PAR^0_i \cup HAIR^0_{rc}$, as sketched in Figure 13, by creating new maximal assemblies from the original ones, and collecting the new tiles to form T_1 and T_2. Such an approach can easily guarantee that every string in the original language is still in the language generated by the new tiles; it must also be shown that no additional strings are generated.

Suppose without loss of generality that the tiles in T_0 are all width w, and that all maximal assemblies induce complete tiles (i.e., they have cap tiles on each end).

Step 1: For every assembly $\alpha \in \mathcal{A}(T_0)$, create a new assembly of width-$(w+1)$ tiles for each maximal path in t_α. All the edges in the original tiles are present in the new tiles, but we keep only the edge labels on the chosen maximal path (colored green); edge labels on the remaining edges (brown) are replaced by λ. Additionally, we add λ-labelled edges from the left cap's bottom port to the start of the maximal path (yellow), and from the end of the maximal path back to the left caps' bottom port (cyan). All remaining nicks are closed with λ-labelled edges (magenta). Finally, the binding labels are augmented by the color pattern of the ports. The union of all such tiles created for all maximal paths for all assemblies in $\mathcal{A}(T_0)$ is the set T_1. Note that T_1 must be a finite set, because there are a finite number of possible width-$(w+1)$ tiles with the allowed edge and binding labels.

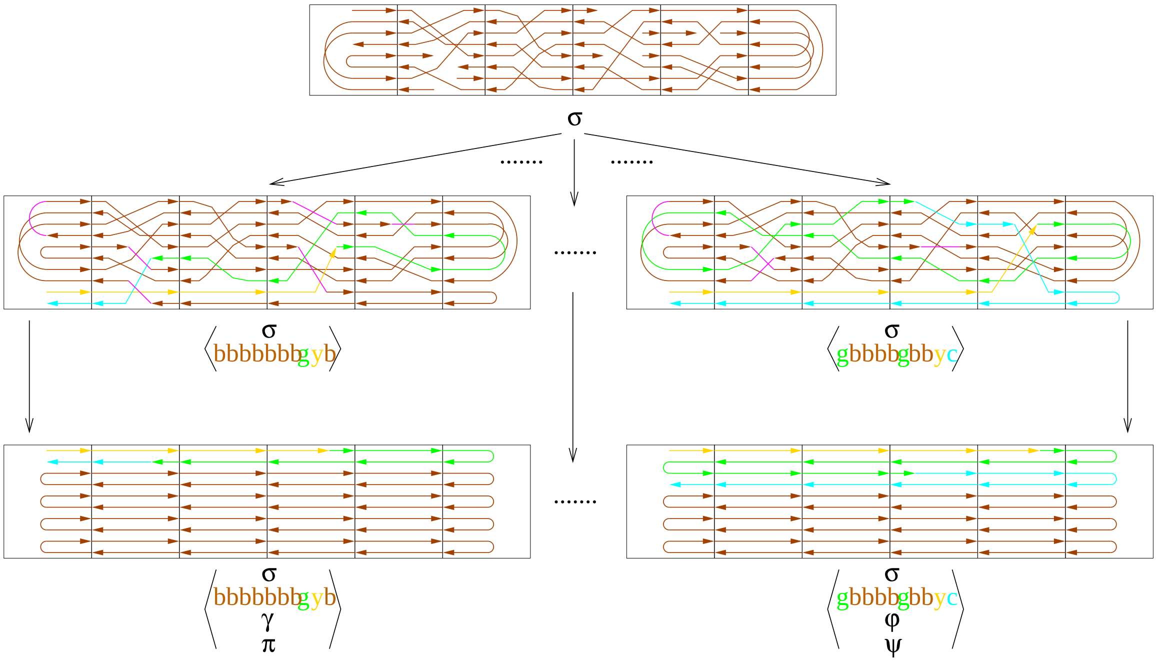

Fig. 13. The correspondence between permutation tiles and parallel tiles. At the top is an original assembly in $\mathcal{A}(T_0)$; in the middle are the corresponding assemblies in $\mathcal{A}(T_1)$, one assembly per maximal path in the original assembly; and at the bottom are the assemblies in $\mathcal{A}(T_2)$, where the chosen path has been moved to the upper rows. σ is the binding label joining the two central tiles in the original assembly. $< \sigma, bbbbbbbgyb >$ is the binding label in step 1 augmented by the color pattern of the ports. γ is the permutation applied to the right port of the tile on the left, and and π is applied to the left port of the tile on the right.

To see that $\mathcal{L}_{ST}(T_0) \subseteq \mathcal{L}_{ST}(T_1)$, note that for every word in $\mathcal{L}_{ST}(T_0)$ there is an assembly in $\mathcal{A}(T_0)$ that contains that word on a maximal path, and thus there is an assembly in $\mathcal{A}(T_1)$ that contains the same word on a maximal path.

To see that $\mathcal{L}_{ST}(T_1) \subseteq \mathcal{L}_{ST}(T_0)$, note that every assembly in $\mathcal{A}(T_1)$ induces a complete tile with a unique maximal path (and perhaps many circles). The augmented binding labels ensure that this path is colored yellow-green-cyan, and thus that it corresponds to a maximal path in the corresponding assembly in $\mathcal{A}(T_0)$ obtained by replacing each tile by one that had been used to create it.

Step 2: For every assembly $\alpha \in \mathcal{A}(T_1)$, create a new assembly of width-$(w+1)$ tiles by permuting the input and output nodes on each original tile so that in the new assembly, all internal tiles are parallel tiles and the maximal path is layered at the top. Augment each port's binding label to include the permutations used for its input and output node lists. Finally, since the brown edges are all λ-labelled and involved in cycles, we replace the brown edges in the cap tiles so as to obtain hairpin cap tiles. Again, the union of all such tiles created for all assemblies in $\mathcal{A}(T_1)$ is the set T_2. Again, T_2 must be a finite set, because there are a finite number of possible tiles. The argument that $\mathcal{L}_{ST}(T_1) = \mathcal{L}_{ST}(T_2)$ is similar to the one given above.

Thus, we have sketched the proof for:

Theorem 2. $\mathcal{L}_{ST}(HAIR^1, PAR^0, HAIR^0) = \mathcal{L}_{ST}(GEN, PERM, GEN)$.

5.7 Hairpin Normal Form for General Tiles

A similar normal form, using hairpin tiles, can be found for general tiles. The argument has three steps. First, hairpin tiles are a normal form for planar tiles. Second, planar tiles are shown sufficient to generate

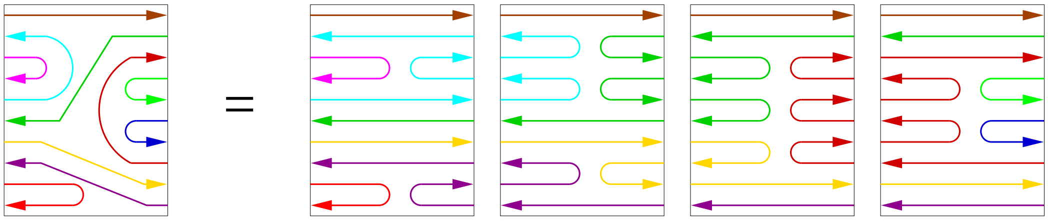

Fig. 14. Planar tiles can be built from hairpin tiles.

the output languages produced by Hennie Machines. Third, a Hennie Machine can be found that outputs the language generated by any given set of linear string tiles.

Figure 14 illustrates the first step, which is given without proof:

Theorem 3. $\mathcal{L}_{ST}(HAIR^1, HAIR^0, HAIR^0) = \mathcal{L}_{ST}(HAIR^1, PLA^0, HAIR^0)$.

For the second step, we will use Turing Machines (TM) equipped with a 2-way read/write input tape and a 1-way write-only output tape. Let Q be the (finite) set of head states, Σ_I be the (finite) set of input tape symbols, and Σ_O be the (finite) set of output tape symbols. Then the state transition table for a TM consists of entries of the form $q\sigma \to q'\sigma' Ds$ where $q, q' \in Q$, $\sigma, \sigma' \in \Sigma_I$, $D \in \{L, R, H_A, H_R\}$, and $s \in \Sigma_O^*$; every pair in $Q \times \Sigma_I$ appears exactly once on the left-hand-side in the state transition table. The action H_A is to halt and accept; the action H_R is to halt and reject; L and R indicate moving left and right, respectively. A TM has the *k-visit* property, and hence is a *finite-visit* TM, if for no input does the machine enter any tape cell more than k times.

Related models include *Hennie Machines* (HM), which are finite-visit TMs whose use of tape space is bounded by a linear function of the input string length (i.e. they are linear bounded automata); and *two-way generalized sequential machines* ($2gsm$), which are Hennie Machines with a read-only input tape that is never accessed beyond the ends of the input string.

If a TM m computing on input x enters the accepting halt state with y on the output tape, then we say that $y = m(x)$. The *output language* of a TM m is $L(m) = \{y : y = m(x)\}$. The family of all output languages of finite-visit TMs is

$$\mathcal{L}_{TM_{fin}} = \{L(m) : m \text{ is a finite-visit } TM \}.$$

Likewise,

$$\mathcal{L}_{HM} = \{L(m) : m \text{ is a HM } \}$$

and

$$\mathcal{L}_{2gsm_{fin}} = \{L(g) : g \text{ is a finite-visit } 2gsm \}.$$

By padding input strings with extra blank symbols, it is easy to see that $\mathcal{L}_{TM_{fin}} = \mathcal{L}_{HM}$. A corollary of our argument will be that, additionally, $\mathcal{L}_{TM_{fin}} = \mathcal{L}_{2gsm_{fin}}$.

Given a HM m, we must define a finite set of tiles $T \subset HAIR^1_{lc} \cup PLA^0_i \cup HAIR^0_{rc}$ that generate the same language; i.e. $L(m) = L_{ST}(T)$. For each $x \in \Sigma_I^*$, we draw the execution of m computing on x as shown in Figure 15, including a (yellow) path from the leftmost tile to the beginning of the input where the head starts, and a path (cyan) from where the head halts back to the bottom of the leftmost tile, filling up all space in between. Each tile represents a single tape cell, and contains information about every visit to that cell by the Turing Machine head. The binding labels give the state of the Turing Machine head for each entry to and exit from the cell, thus ensuring that any maximal assembly represents a valid execution of the Turing Machine. The union of all tiles so created is T. Using a similar argument to that given for parallel normal form, we see that $L_{ST}(T) = L(m)$.

Theorem 4. $\mathcal{L}_{HM} \subseteq \mathcal{L}_{ST}(HAIR^1, PLA^0, HAIR^0)$.

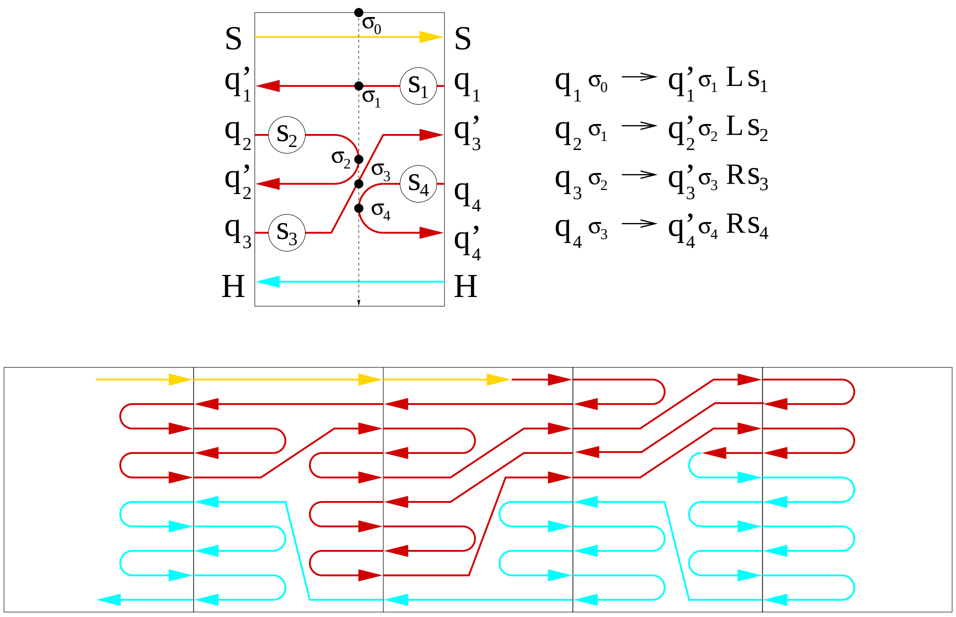

Fig. 15. Planar tiles that simulate a finite-visit Turing Machine. (top) The general form of tiles, showing start edges (yellow), action edges (red), and halt edges (cyan). Although formally not part of the tile, we label the cell with tape symbols σ_i along the center, indicating what must be on the tape during each visit of the Turing Machine head. Each action edge corresponds to an entry in the TM state transition table, as shown on the right. (bottom) An assembly containing the history of a Turing Machine computation.

For the third step of the argument, we need to find a Hennie Machine that outputs the same language as any given set of general tiles. Let $T \subset GEN$ be a finite set (without loss of generality, assume that all maximal assemblies of T have cap tiles on both ends) of width-w linear string tiles over Σ, Φ. Our HM will have input alphabet $\Sigma_I = \hat{T} = T \times \{L_i : 0 \leq i \leq w\} \times \{R_i : 0 \leq i \leq w\}$ and the output alphabet $\Sigma_O = \Sigma$. L_i and R_i are used only to select which path to read in assemblies containing multiple nicks. The HM proceeds in two phases: first it checks that the input represents a valid maximal assembly, then it reads the word off one of the maximal paths. The (finite) information about the tile types, edges, and labels is contained in the HM's finite state logic. The HM first checks that the first input symbol represents a left cap tile; then it scans right (producing no output) so long as each successive tile is compatible with the preceding one; either it arrives finally at a right cap tiles, or else it halts in the rejecting state, thus producing no output. In the former case, the HM then scans back to the left until it finds the first symbol (t, L_i, R_j) where O^i_L or O^j_R is nicked. (If it finds no such tile, it halts in the reject state.) The HM then simply follows the chosen edges from its beginning until its end, producing as output the edge labels, then halting in the accept state. Note that because symbols can contain L_0 and R_0 but ports are numbered starting from 1, the HM can copy any maximal path from any assembly. Also note that this HM uses the input tape for reads only, and hence is a finite-visit $2gsm$. Thus, we have:

Theorem 5. $\mathcal{L}_{ST}(GEN, GEN, GEN) \subseteq \mathcal{L}_{2gsm_{fin}}$.

Altogether,

Theorem 6. $\mathcal{L}_{ST}(HAIR^1, HAIR^0, HAIR^0) = \mathcal{L}_{ST}(GEN, GEN, GEN) = \mathcal{L}_{HM} = \mathcal{L}_{2gsm_{fin}}$.

6 Conclusions and Open Questions

We can now fit the languages generated by linear string tiles into known language classes. $ET0L$ systems [Roz73,RV78,RV80,EH99] are the most convenient well-studied model. Diagrams very similar to string tiles came up in the study of transductions by finite-visit machines [EH99]. Of particular interest are the metalinear $ET0L$ systems, which generate the languages in $ET0L_{ml}$, and the $ET0L$ systems of finite index, which generate the language in $ET0L_{fin}$. It is straightforward to show that scattered metalinear grammars are a normal form for metalinear $ET0L$ systems, and thus $\mathcal{L}_{SM} = ET0L_{ml}$. In [ERS80] it was proved that $\mathcal{L}_{2gsm_{fin}} = ET0L_{fin}$. Furthermore, it was shown in [RV80] that the language $\{a^n b^n : n \geq 1\}^*$ is in $ET0L_{fin} \setminus ET0L_{ml}$. Indeed, there is a simple set of width-2 planar tiles that generate this language, and we can conclude that no set of parallel tiles can do so. Thus Figure 1 is justified.

We have given a full characterization only of the language classes generated by finite sets of tiles. However, complexity issues remain to be investigated – how many tiles are necessary to generate a specific language? The parallel normal form theorem potentially uses exponentially (in w) more parallel tiles than permutation tiles. This question has obvious relevance to using string tiles to solve NP-complete problems, such as CNF-SAT or DHPP.

Our definition of string tiles allows edges to be labeled by the empty string, corresponding to tiles with DNA containing no coding sequence. Thus, the resulting ligated DNA strands may have very long regions coding for no information. How do our language classes change if we insist on only λ-free string tiles?

Are the circular languages significantly different from the linear languages? Unlike linear DNA strands, circular DNA strands can be knotted with themselves and with other circular strands (although this is not part of the current formal model); can knottedness increase the computational power? Careful routing of strands using string tiles augments the computational power of linear DNA assembly; for 2D or 3D assembly, although string tiles cannot increase the language class beyond RE, can string tiles be used for more efficient computation?

Do the constructions and results presented in this paper point to better practical implementations for DNA-based computing? It is hard to say at this point, although the following calculation is illustrative. Consider a 40 variable CNF-SAT problem, with 160 clauses. In our construction, 80 string tiles must be prepared, each assembled from 240 prototiles (this number could be reduced substantially with an improved construction). These tiles would be 240×75 nm, with 160 sticky ends on each side. At a prototile concentration of $20 \mu M$, one ml of solution would hold 12×10^{15} prototiles, thus 4×10^{13} tiles and 1×10^{12} maximal assemblies – just sufficient for 1X coverage of variable assignments. If self-assembly of these monsters were reliable (40 sticky-end sets would have to be sufficiently distinct) and roughly as fast as oligonucleotide hybridization, the maximal assemblies would form in a few minutes. The assembly containing the satisfying strand would still have to survive ligation at each of up to 160×40 nicks, at (on a good day) 90% yield for each nick. On the one hand, we're excited to see a new approach for DNA based computing by self-assembly, which may have payoff for simple examples like generating an addition table; on the other hand, significant practical applications at this point seem rather far off.

Acknowledgements. The authors are indebted to Joost Engelfriet and Hendrik Jan Hoogeboom for their guidance through the maze of results on the output languages of various sorts of transducers.

References

[Adl94] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.

[EH99] J. Engelfriet and H. J. Hoogeboom. Two-way finite state transducers and monadic second-order logic. In *Lecture Notes in Computer Science*, volume 1644, pages 311–320. Springer Verlag, 1999.

[Eng99] Tony Eng. Linear DNA self-assembly with hairpins generates the equivalent of linear context-free grammars. In Rubin and Wood [RW99].

[ERS80] J. Engelfriet, G. Rozenberg, and G Slutzki. Tree transducers, L systems, and two-way machines. *J. Comp. and Syst. Sc.*, 20:150–202, 1980.

[FS93] Tsu-Ju Fu and Nadrian C. Seeman. DNA double-crossover molecules. *Biochemistry*, 32:3211–3220, 1993.

[JKS98] Nataša Jonoska, Stephen A. Karl, and Masahico Saito. Three dimensional DNA structures in computing. In Lila Kari, Harvey Rubin, and David H. Wood, editors, *Proceedings of the* 4^{th} *DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, pages 189–200, preliminary, 1998.

[JKS99] Nataša Jonoska, Stephen A. Karl, and Masahico Saito. Creating 3-dimensional graph structures with DNA. In Rubin and Wood [RW99], pages 123–135.

[Knu73] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed).* Addison-Wesley, 1973.

[LLar] Michail G. Lagoudakis and Thomas H. LaBean. 2D DNA self-assembly for satisfiability. In Winfree and Gifford [WGar].

[LSS99] Furong Liu, Ruojie Sha, and Nadrian C. Seeman. Modifying the surface features of two-dimensional DNA crystals. *Journal of the American Chemical Society*, 121(5):917–922, 1999.

[LWRar] Thomas H. LaBean, Erik Winfree, and John H. Reif. Experimental progress in computation by self-assembly of DNA tilings. In Winfree and Gifford [WGar].

[LYK+00] Thomas H. LaBean, Hao Yan, Jens Kopatsch, Furong Liu, Erik Winfree, John H. Reif, and Nadrian C. Seeman. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *Journal of the American Chemical Society*, 122:1848–1860, 2000.

[MSS99] Chengde Mao, Weiqiong Sun, and Nadrian C. Seeman. Designed two-dimensional DNA Holliday junction arrays visualized by atomic force microscopy. *Journal of the American Chemical Society*, 121(23):5437–5443, 1999.

[Rei99] John Reif. Local parallel biomolecular computing. In Rubin and Wood [RW99], pages 217–254.

[Roz73] Grzegorz Rozenberg. Extension of tabled 0L-systems and languages. *Intern. J. Comp. Inform. Sci.*, 2:311–336, 1973.

[RS97] Grzegorz Rozenberg and Arto Salomaa. *Handbook of formal languages*, volume 2. Springer-Verlag, New York, 1997.

[RV78] G. Rozenberg and D. Vermeir. On ETOL systems of finite index. *Information and Control*, 38:103–133, 1978.

[RV80] G. Rozenberg and D. Vermeir. On metalinear ETOL systems. *Fundamenta Informaticae*, pages 15–36, 1980.

[RW99] Harvey Rubin and David Harlan Wood, editors. *DNA Based Computers III: DIMACS Workshop, June 23-25, 1997*, volume 48 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, Providence, RI, 1999. American Mathematical Society.

[See82] Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99(2):237–247, 1982.

[See98] Nadrian C. Seeman. DNA nanotechnology: novel DNA constructions. *Annual Review of Biophysics and Biomolecular Structure*, 27:225–248, 1998.

[SWY+98] N. C. Seeman, H. Wang, X. P. Yang, F. R. Liu, C. D. Mao, W. Q. Sun, L. Wenzler, Z. Y. Shen, R. J. Sha, H. Yan, M. H. Wong, P. Sa-Ardyen, B. Liu, H. X. Qiu, X. J. Li, J. Qi, S. M. Du, Y. W. Zhang, J. E. Mueller, T. J. Fu, Y. L. Wang, and J. H. Chen. New motifs in DNA nanotechnology. *Nanotechnology*, 9(3):257–273, 1998.

[WGar] Erik Winfree and David K. Gifford, editors. *Proceedings of the* 5^{th} *DIMACS Meeting on DNA Based Computers, held at MIT, June 14-15, 1999*, Providence, RI, to appear. American Mathematical Society.

[Win96] Erik Winfree. On the computational power of DNA annealing and ligation. In Richard J. Lipton and Eric B. Baum, editors, *DNA Based Computers: DIMACS Workshop, April 4, 1995*, volume 27, pages 199–221, Providence, RI, 1996. American Mathematical Society.

[WLWS98] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.

[WYS98] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Laura F. Landweber and Eric B. Baum, editors, *DNA Based Computers II: DIMACS Workshop, June 10-12, 1996*, volume 44, Providence, RI, 1998. American Mathematical Society.