Staged Self-Assembly and Polyomino Context-Free Grammars

Andrew Winslow*

Department of Computer Science, Tufts University,
awinslow@cs.tufts.edu

Abstract. Previous work by Demaine et al. (2012) developed a strong connection between smallest context-free grammars and staged self-assembly systems for one-dimensional strings and assemblies. We extend this work to two-dimensional polyominoes and assemblies, comparing staged self-assembly systems to a natural generalization of context-free grammars we call *polyomino context-free grammars (PCFGs)*.
We achieve nearly optimal bounds on the largest ratios of the smallest PCFG and staged self-assembly system for a given polyomino with n cells. For the ratio of PCFGs *over* assembly systems, we show that the smallest PCFG can be an $\Omega(n/\log^3 n)$-factor larger than the smallest staged assembly system, even when restricted to square polyominoes. For the ratio of assembly systems over PCFGs, we show that the smallest staged assembly system is never more than a $O(\log n)$-factor larger than the smallest PCFG and is sometimes an $\Omega(\log n/\log\log n)$-factor larger.

1 Introduction

In the mid-1990s, the Ph.D. thesis of Erik Winfree [22] introduced a theoretical model of self-assembling nanoparticles. In this model, which he called the *abstract tile assembly model (aTAM)*, square particles called *tiles* attach edgewise to each other if their edges share a common *glue* and the bond strength is sufficient to overcome the kinetic energy or *temperature* of the system. The products of these systems are *assemblies*: aggregates of tiles forming via crystal-like growth starting at a *seed tile*. Surprisingly, these tile systems have been shown to be computationally universal [22,5], self-simulating [11,12], and capable of optimally encoding arbitrary shapes [18,1,21].

In parallel with work on the aTAM, a number of variations on the model have been proposed and investigated. These variations change a number of features of the original aTAM, for instance allowing glues to repulse [10,17,19], or adding labels to each tile to produce patterned assemblies [13,6,20]. For a more thorough treament of the aTAM and its variants, see the recent surveys of Patitz [16] and Doty [9].

One well-studied variant called the *hierarchical* [4] or *two-handed assembly model (2HAM)* [7] eliminates the seed tile and allows tiles and assemblies to

* Supported in part by National Science Foundation grant CBET-0941538.

attach in arbitrary order. This model was shown to be capable of (theoretically) faster assembly of squares [4] and simulation of aTAM systems [2], including capturing the seed-originated growth dynamics. A generalization of the 2HAM model proposed by Demaine et al. [7] is the *staged assembly model*, which allows the assemblies produced by one system to be used as reagents (in place of tiles) for another system, yielding systems divided into sequential assembly *stages*. They showed that such sequential assembly systems can replace the role of glues in encoding complex assemblies, allowing the construction of arbitrary shapes efficiently while only using a constant number of glue types, a result impossible in the aTAM or 2HAM.

To understand the power of the staged assembly model, Demaine et al. [8] studied the problem of finding the smallest system producing a one-dimensional assembly with a given sequence of labels on its tiles, called a *label string*. They proved that for systems with a constant number of glue types, this problem is equivalent to the well-studied problem of finding the smallest context-free grammar whose language is the given label string, also called the *smallest grammar problem* (see [15,3]). For systems with unlimited glue types, they proved that the ratio of the smallest context-free grammar *over* the smallest system producing an assembly with a given label string of length n (which they call *separation*) is $\Omega(\sqrt{n/\log n})$ and $O((n/\log n)^{2/3})$ in the worst case.

In this paper we consider the two-dimensional version of this problem: finding the smallest staged assembly system producing an assembly with a given *label polyomino*. For systems with constant glue types and no cooperative bonding, we achieve separation of grammars *over* these systems of $\Omega(n/(\log\log n)^2)$ for polyominoes with n cells (Sect. 6.1), and $\Omega(n/\log^3 n)$ when restricted to rectangular (Sect. 6.2) or square (Sect. 6.3) polyominoes with a constant number of labels. Adding the restriction that each step of the assembly process produces a single product, we achieve $\Omega(n/\log^3 n)$ separation for general polyominoes with a single label (Sect. 6.1). For the separation of staged assembly systems *over* grammars, we achieve bounds of $\Omega(\log n/\log\log n)$ (Sect. 4) and, constructively, $O(\log n)$ (Sect. 5). For all of these results, we use a simple definition of context-free grammars on polyominoes that generalizes the deterministic context-free grammars (called *RCFGs*) of [8].

When taken together, these results give a nearly complete picture of how smallest context-free grammars and staged assembly systems compare. For some polyominoes, staged assembly systems are exponentially smaller than context-free grammars ($O(\log n)$ vs. $\Omega(n/\log^3 n)$). On the other hand, given a polyomino and grammar deriving it, one can construct a staged assembly system that is a (nearly optimal) $O(\log n)$-factor larger and produces an assembly with a label polyomino replicating the polyomino.

2 Staged Self-Assembly

An instance of the staged tile assembly model is called a *staged assembly system* or *system*, abbreviated *SAS*. A SAS $\mathcal{S} = (T, G, \tau, M, B)$ is specified by five parts: a *tile set* T of square *tiles*, a *glue function* $G : \Sigma(G)^2 \rightarrow \{0, 1, \ldots, \tau\}$, a *temperature* $\tau \in \mathbb{N}$, a directed acyclic *mix graph* $M = (V, E)$, and a *start bin function* $B : V_L \rightarrow T$ from the *leaf vertices* $V_L \subseteq V$ of M with no incoming edges.

Each tile $t \in T$ is specified by a 5-tuple (l, g_n, g_e, g_s, g_w) consisting of a label l taken from an alphabet $\Sigma(T)$ (denoted $l(t)$) and a set of four non-negative integers in $\Sigma(G) = \{0, 1, \ldots, k\}$ specifying the *glues* on the sides of t with normal vectors $\langle 0, 1\rangle$ (north), $\langle 1, 0\rangle$ (east), $\langle 0, -1, \rangle$ (south), and $\langle -1, 0\rangle$ (west), respectively, denoted $g_{\boldsymbol{u}}(t)$. In this work we only consider glue functions with the constraints that if $G(g_i, g_j) > 0$ then $g_i = g_j$, and $G(0, 0) = 0$.

A *configuration* is a partial function $C : \mathbb{Z}^2 \rightarrow T$ mapping locations on the integer lattice to tiles. Any two locations $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ in the domain of C (denoted $\mathrm{dom}(C)$) are *adjacent* if $||p_2 - p_1|| = 1$ and the *bond strength* between any pair of tiles $C(p_1)$ and $C(p_2)$ at adjacent locations is $G(g_{p_2 - p_1}(C(p_1)), g_{p_1 - p_2}(C(p_2)))$. A *configuration* is a *τ-stable assembly* or an *assembly at temperature* τ if $\mathrm{dom}(C)$ is connected on the lattice and, for any partition of $\mathrm{dom}(C)$ into two subconfigurations C_1, C_2, the sum of the bond strengths between tiles at pairs of locations $p_1 \in \mathrm{dom}(C_1)$, $p_2 \in \mathrm{dom}(C_2)$ is at least τ. Any pair of configurations C_1, C_2 are equivalent if there exists a vector $\boldsymbol{v} = \langle x, y\rangle$ such that $\mathrm{dom}(C_1) = \{p + \boldsymbol{v} \mid p \in \mathrm{dom}(C_2)\}$ and $C_1(p) = C_2(p + \boldsymbol{v})$ for all $p \in \mathrm{dom}(C_1)$. Two τ-stable assemblies A_1, A_2 are said to *assemble* into a *superassembly* A_3 if there exists a translation vector $\boldsymbol{v} = \langle x, y\rangle$ such that $\mathrm{dom}(A_1) \cap \{p + \boldsymbol{v} \mid p \in A_2\} = \emptyset$ and A_3 defined by the partial functions A_1 and A_2' with $A_2'(p) = A_2(p + \boldsymbol{v})$ is a τ-stable assembly.

Each vertex of the mix graph M describes a *two-handed assembly process.* This process starts with a set of τ-stable *input assemblies* I. The set of *assembled assemblies* Q is defined recursively as $I \subseteq Q$, and for any pair of assemblies $A_1, A_2 \in Q$ with superassembly A_3, $A_3 \in Q$. Finally, the set of *products* $P \subseteq Q$ is the set of assemblies A such that for any assembly A', no superassembly of A and A' is in Q.

The mix graph $M = (V, E)$ of $\mathcal{S}$ defines a set of two-handed assembly processes (called *mixings*) for the non-leaf vertices of M (called *bins*). The input assemblies of the mixing at vertex v is the union of all products of mixings at vertices v' with $(v', v) \in E$. The start bin function B defines the lone single-tile product of each mixings at a leaf bin. The system $\mathcal{S}$ is said to *produce* an assembly A if some mixing of $\mathcal{S}$ has a single product, A. We define the size of $\mathcal{S}$, denoted $\mathcal{S}$, to be $|E|$, the number of edges in M. If every mixing in a $\mathcal{S}$ has a single product, then $\mathcal{S}$ is a *singular self-assembly system (SSAS).*

The results of Section 6.4 use the notion of a self-assembly system $\mathcal{S}'$ *simulating* a system $\mathcal{S}$ by carrying out the same sequence of mixings and producing a set of scaled assemblies. Formally, we say a system $\mathcal{S}' = (T', G', \tau, M', B')$ *simulates* a system $\mathcal{S} = (T, G, \tau, M, B)$ at *scale* b if there exist two functions f, g with the following properties:

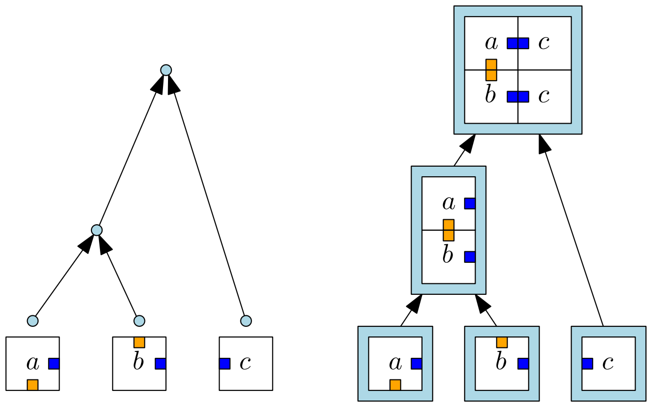


Fig. 1. A self-assembly system (SAS) consisting of a mix graph and tile types (left), and the assemblies produced by carrying out the algorithmic process of staged self-assembly (right).

(1) The function $f : (\Sigma(T') \cup \{\varnothing\})^{b^2} \to \Sigma(T) \cup \{\varnothing\}$ maps the labels of $b \times b$ regions of tiles (called *blocks*) to a label of a tile in T. The empty label $\varnothing$ denotes no tile.
(2) The function $g : S' \to V$ maps a subset S' of the vertices of the mix graph M' to vertices of the mix graph M such that g is an isomorphism between the subgraph induced by S' in M' and the graph M.
(3) Let $P(v)$ be the set of products of the bin corresponding to vertex v in a mix graph. Then for each vertex $v \in M$ with $v' = g^{-1}(v)$, $P(v) = \{f(p) \mid p \in P(v')\}$.

Intuitively, f defines a correspondence between the b-scaled macrotiles in $\mathcal{S}'$ simulating tiles in $\mathcal{S}$, and g defines a correspondence between bins in the systems. Property (3) requires that f and g do, in fact, define correspondence between what the systems produce.

The self-assembly systems constructed in Sections 5 and 6 produce only *mismatch-free assemblies*: assemblies in which every pair of incident sides of two tiles in the assembly have the same glue. A system is defined to be *mismatch-free* if every product of the system is mismatch-free.

3 Polyomino Context-Free Grammars

Here we describe polyominoes, a generalization of strings, and polyomino context-free grammars, a generalization of deterministic context-free grammars. These objects replace the strings and restricted context-free grammars (RCFGs) of Demaine et al. [8].

A *labeled polyomino* or *polyomino* $P = (S, L)$ is defined by a connected set of points S on the square lattice (called *cells*) containing $(0, 0)$ and a label function

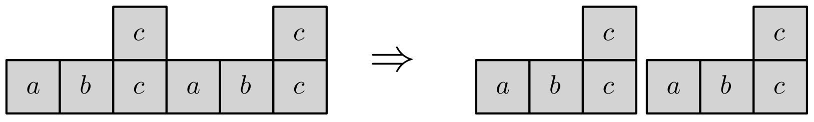
$L : S \to \Sigma(P)$ mapping each cell of P to a *label* contained in an alphabet $\Sigma(P)$. The *size* of P is the number of cells P contains and is denoted $|P|$. The label of the cell at lattice point (x, y) is denoted $L((x, y))$ and we define $P(x, y) = L((x, y))$ for notational convenience. We refer to the *label* or *color* of a cell interchangeably.

Define a *polyomino context-free grammar (PCFG)* to be a quadruple $G = (\Sigma, \Gamma, S, \Delta)$. The set Σ is a set of *terminal symbols* and the set Γ is a set of *non-terminal symbols*. The symbol $S \in \Gamma$ is a special *start symbol*. Finally, the set Δ consists of *production rules*, each of the form $N \to (R_1, (x_1, y_1)) \ldots (R_j, (x_j, y_j))$ where $N \in \Gamma$ and is the left-hand side symbol of only this rule, $R_i \in N \cup T$, and each (x_i, y_i) is a pair of integers. The *size* of G is defined to be the total number of symbols on the right-hand sides of the rules of Δ.

A polyomino P can be derived by starting with S, the start symbol of G, and repeatedly replacing a non-terminal symbol with a set of non-terminal and terminal symbols. The set of valid replacements is Δ, the production rules of G, where a non-terminal symbol N with lower-leftmost cell at (x, y) can be replaced with a set of symbols R_1 at $(x+x_1, y+y_1)$, R_2 at $(x+x_2, y+y_2)$, $\ldots$, R_j at $(x+x_j, y+y_j)$ if there exists a rule $N \to (R_1, (x_1, y_1))(R_2, (x_2, y_2)) \ldots (R_j, (x_j, y_j))$. Additionally, the set of terminal symbol cells derivable starting with S must be connected and pairwise disjoint.

The polyomino P derived by the start symbol of a grammar G is called the *language of* G, denoted $L(G)$, and G is said to *derive* P. In the remainder of the paper we assume that each production rule has at most two right-hand side symbols (equivalent to binary normal form for 1D CFGs), as any PCFG can be converted to this form with only a factor-2 increase in size. Such a conversion is done by iteratively replacing two right-hand side symbols R_i, $R_{i'}$ with a new non-terminal symbol Q, and adding a new rule replacing Q with R_i and $R_{i'}$.

Intuitively, a polyomino context-free grammar is a recursive decomposition of a polyomino into smaller polyominoes. Because each non-terminal symbol is the left-hand side symbol of at most one rule, each non-terminal corresponds to a subpolyomino of the derived polyomino. Then each production rule is a decomposition of a subpolyomino into smaller subpolyominoes (see Figure 2).



$$N \to (R, (0, 0))(R, (3, 0))$$

Fig. 2. Each production rule in a PCFG generating a single shape is a decomposition of the left-hand side non-terminal symbol's polyomino into the right-hand side symbols' polyominoes.

In this interpretation, the smallest grammar deriving a given polyomino is equivalent to a decomposition using the fewest *distinct* subpolyominoes in the decomposition. As for the smallest CFG for a given string, the smallest PCFG for a given polyomino is deterministic and finding such a grammar is NP-hard. Moreover, even approximating the smallest grammar is NP-hard [3], and achieving optimal approximation algorithms remains open [14].

In Section 5 we construct self-assembly systems that produce assemblies whose label polyominoes are scaled versions of other polyominoes, with some amount of "fuzz" in each scaled cell. A polyomino $P' = (S', L')$ is said to be a (c, d)-*fuzzy replica* of a polyomino $P = (S, L)$ if there exists a vector $\langle x_t, y_t \rangle$ with the following properties:

1. For each block of cells $\mathcal{S}'_{(i,j)} = \{(x, y) \mid x_t + di \leq x < x_t + d(i + 1), y_t + dj \leq y < y_t + d(j + 1)\}$ (called a *supercell*), $\mathcal{S}'_{(i,j)} \cap S' \neq \varnothing$ if and only if $(i, j) \subseteq S$.
2. For each supercell $\mathcal{S}'_{(i,j)}$ containing a cell of P', the subset of *label cells* $\{(x, y) \mid x_t + di + (d - c)/2 \leq x < x_t + d(i + 1) + (d - c)/2, y_t + dj + (d - c)/2 \leq y < y_t + d(j + 1) + (d - c)/2\}$ consists of c^2 cells of P', with all cells having identical label, called the *label of the supercell* and denoted $\mathcal{L}_{(i,j)}$.
3. For each supercell $\mathcal{S}'_{(i,j)}$, any cell that is not a label cell of $\mathcal{S}'_{(i,j)}$ has a common *fuzz label* in L'.
4. For each supercell $\mathcal{S}'_{(i,j)}$, the label of the supercell $\mathcal{L}'_{(i,j)} = P(i, j)$.

Properties (1) and (2) define how sets of cells in P' replicate individual cells in P, and the labels of these sets of cells and individual cells. Property (3) restricts the region of each supercell not in the label region to contain only cells with a common fuzz label. Property (4) requires that each supercell's label matches the label of the corresponding cell in P.

4 SAS over PCFG Separation Lower Bound

This result uses a set of shapes we call n-*staggler*s, an example is seen in Figure 3. The shapes consist of $\log n$ bars of dimensions $n/\log n \times 1$ stacked vertically atop each other, with each bar horizontally offset from the bar below it by some amount in the range $-(n/\log n - 1), \ldots, n/\log n - 1$. We use the shorthand that $\log n = \lfloor \log n \rfloor$ for conciseness. Every sequence of $\log n - 1$ integers, each in the range $[-(n/\log n - 1), n/\log n - 1]$, encodes a unique staggler and by the pidgeonhole principle, some n-staggler requires $\log((2n/\log n - 1)^{\log n - 1} = \Omega(\log^2 n)$ bits to specify.

Lemma 1. *Any n-staggler can be derived by a PCFG of size $O(\log n)$.*

Proof. A set of $O(\log n)$ production rules deriving a bar (of size $\Theta(n/\log n) \times 1$) can be constructed by repeatedly doubling the length of the bar, using an additional $\log n$ rules to form the bar's exact length. The result of these production rules is a single non-terminal B deriving a complete bar.

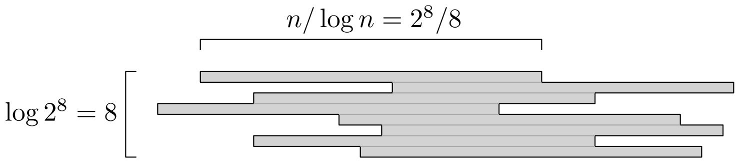


Fig. 3. The 2^8-staggler specified by the sequence $-18, 13, 9, -17, -4, 12, -10$.

Using the non-terminal B, a stack of k bars can be described using a production rule $N \to (B,(x_1,0))(B,(x_2,1))\ldots(B,(x_k,k-1))$, where the x-coordinates $x_1, x_2, \ldots, x_k$ encode the offsets of each bar relative to the bar below it. An equivalent set of $k-1$ production rules in binary normal form can be produced by creating a distinct non-terminal for T_i each stack of the first i bars, and a production rule $T_i \to (T_{i-1},(0,0))(B,(x_i,i))$ encoding the offset of the topmost bar relative to the stack of bars beneath it.

In total, $O(\log n)$ rules are used to create B, the non-terminal deriving a bar, and $O(\log n)$ are used to create the stack of bars, one per bar. So the n-staggler can be constructed using a PCFG of size $O(\log n)$.

Lemma 2. *For every n, there exists an n-staggler P such that any SAS or SSAS producing an assembly with label polyomino P has size $\Omega(\log^2 n/\log\log n)$.*

Proof. The proof is information-theoretic. Recall that more than half of all n-stagglers require $\Omega(\log^2 n)$ bits to specify. Now consider the number of bits contained in a SAS $\mathcal{S}$. Recall that $|\mathcal{S}|$ is the number of edges in the mix graph of $\mathcal{S}$. Any SAS can be encoded naively using $O(|\mathcal{S}|\log|\mathcal{S}|)$ bits to specify the mix graph, $O(|T|\log|T|)$ bits to specify the tile set, and $O(|\mathcal{S}|\log|T|)$ bits to specify the tile type at each leaf node of the mix graph. Because the number of tile types cannot exceed the size of the mix graph, $|T| \leq |\mathcal{S}|$. So the total number of bits needed to specify $\mathcal{S}$ (and thus the number of bits of information contained in $\mathcal{S}$) is $O(|\mathcal{S}|\log|\mathcal{S}| + |T|\log|T| + |\mathcal{S}|\log|\mathcal{S}|) = O(|\mathcal{S}|\log|\mathcal{S}|)$. So some n-staggler requires a SAS $\mathcal{S}$ such that $O(|\mathcal{S}|\log|\mathcal{S}|) = \Omega(\log^2 n)$ and thus $|\mathcal{S}| = \Omega(\log^2 n/\log\log n)$.

Theorem 1. *The separation of SASs and SSASs over PCFGs is $\Omega(\log n/\log\log n)$.*

Proof. By the previous two lemmas, more than half of all n-stagglers require SASs and SSASs of size $\Omega(\log^2 n/\log\log n)$ and all n-stagglers have PCFGs of size $O(\log n)$. So the separation is $\Omega(\log n/\log\log n)$.

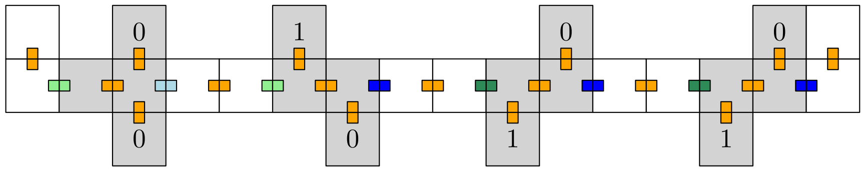
We also note that scaling the n-staggler by a c-factor produces a shape which is derivable by a CFG of size $O(\log n + \log c)$. That is, the result still holds for n-stagglers scaled by any amount polynomial in n. For instance, the $O(n)$-factor of the construction of Theorem 2.

At first it may not be clear how PCFGs achieve smaller encodings. After all, each rule in a PCFG G or mixing in SAS $\mathcal{S}$ specifies either a set of right-hand side symbols or set of input bins to use and so has up to $O(\log|G|)$ or $O(\log|\mathcal{S}|)$

bits of information. The key is the coordinate describing the location of each right-hand side symbol. These offsets have up to $O(\log n)$ bits of information and in the case that G is small, say $O(\log n)$, each rule has a number of bits *linear* in the size of the PCFG!

5 SAS over PCFG Separation Upper Bound

Next we show that the separation lower bound of the last section is nearly large as possible by giving an algorithm for converting any PCFG G into a SSAS $\mathcal{S}$ with system size $O(|G| \log n)$ such that $\mathcal{S}$ produces an assembly that is a fuzzy replica of the polyomino derived by G. Before describing the full construction, we present approaches for efficiently constructing general binary counters and for simulating glues using geometry.



Increment 0011_b by 1, yielding 0100_b.

Fig. 4. A binary counter row constructed using single-bit constant-sized assemblies. Dark blue and green glues indicate 1-valued carry bits, light blue and green glues indicate 0-valued carry bits.

The *binary counter row assemblies* used here are a generalization of those by Demaine et al. [7] consisting of constant-sized bit assemblies, and an example is seen in Figure 4. Our construction achieves $O(\log n)$ construction of arbitrary ranges of rows and increment values, in contrast to the contruction of [7] that only produces row sets of the form $0, 1, \ldots, 2^{2^m} - 1$ that increment by 1. To do so, we show how to construct two special cases from which the generalization follows easily.

Lemma 3. *Let i, j, n be integers such that $0 \leq i \leq j < n$. There exists a SSAS of size $O(\log n)$ with a set of bins that, when mixed, assemble a set of $j - i + 1$ binary counter rows with values $i, i+1, \ldots, j$ incremented by 1.*

Proof. Representing integers as binary strings, consider the prefix tree induced by the binary string representations of the integers i through j, which we denote $T_{(i,j)}$. The prefix tree $T_{(0,2^m-1)}$ is a complete tree of height m, and the prefix tree $T_{(i,j)}$ with $0 \leq i \leq j \leq 2^m - 1$ is a subtree of $T_{(0,2^m-1)}$ with $j - i + 1$ leaf nodes See Figure 5 for an example with $m = 4$.

Now let $n = 2^m - 1$. If $T_{(0,n)}$ is drawn with leaves in left-to-right order by increasing integer values, then the leaves of the subtree $T_{(i,j)}$ appear contiguously.

So the subtree $T_{(i,j)}$ has at most $2 \log n$ internal nodes with one child forming the leftmost and rightmost paths in $T_{(i,j)}$. Furthermore, if one removes these two paths from $T_{(i,j)}$, the remainder of $T_{(i,j)}$ is a forest of complete trees with at most two trees of each height and $2 \log n$ trees total.

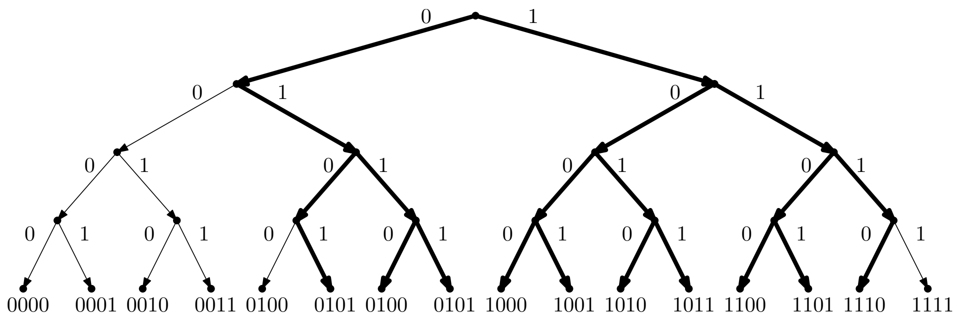


Fig. 5. The prefix tree $T_{(0,15)}$ for integers 0 to $2^4 - 1$ represented in binary. The bold subtree is the prefix subtree $T_{(5,14)}$ for integers 5 to 14.

Note that a complete subtree of the prefix tree corresponds to a set of all possible 2^h suffixes of length h, where h is the height of the subtree. The leaves of such a subtree then correspond to the set of strings of length l with a specific prefix of length $l - h$ and any suffix of length h. For the assemblies we use the same geometry-based encoding of each bit as [7], and a distinct set of glues used for each bit of the assembly encoding both the bit index and carry bit value from the previous bit.

Left and right bins. We build a mix graph (seen in Figure 6) consisting of two disjoint paths of bins (called *left bins* and *right bins*) that are used to iteratively assemble partial counter rows i and j by the addition of distinct constant-sized assemblies for each bit. The partial rows are used to produce the assemblies in the subtree and missing bit bins (described next). In the suffix trees, the bit strings of these assemblies are progressively longer subpaths of the leftmost and rightmost paths in the subtree of binary strings of the integers i to j.

Subtree bins. Assemblies in subtree bins correspond to assemblies encoding prefixes of binary counter row values. However, unlike left and right bins that encode prefixes of only a single value, subtree bins encode prefixes of many binary counter values between i the j – namely a set of values forming a maximal complete subtree of the subtree of binary strings of integers from i to j, hence the name *subtree bins*. For example, if $i = 12$ and $j = 16$, then the set of binary strings for values 12 (01100_b) to 15 (01111_b) have a common prefix 011_b. In this case a subtree bin containing an assembly encoding the three bits 011 would be created. Since there are at most $2 \log n$ such complete subtrees, the number of subtree bins is at most this many. Creating each bin only requires a single mixing step of combining an assembly from a left or right bin with a single bit assembly, for example adding a 1-bit assembly to the left bin assembly encoding the prefix 01_b.

Missing bit bins. To add the bits not encoded by the assemblies in the subtree bins, we create sets of four constant-sized assemblies in individual *missing bit bins*. Since the assemblies in subtree bins encode bit string prefixes of sets of values forming complete subtrees, completing these prefixes with *any* suffix forms a bit string whose value is between i and j. This allows complete non-determinism in the bit assemblies that attach to complete the counter row, provided they properly handle carry bits. For every bit index missing in *some* subtree bin assembly, the four assemblies encoding the four possibilities for the input and carry values are assembled and placed into separate bins. When all bins are mixed, subtree assemblies mix non-deterministically with all possible assemblies from missing bit bins, producing all counter rows whose binary strings are found in the subtree. In total, up to $4 \log n$ missing bit bins are created, and each contains a constant-sized assembly and so requires constant work to produce.

The total number of total bins is clearly $O(\log n)$. Consider mixing the left and right bins containing completed counter rows for i and j, all subtree bins, and all missing bit bins. Any assembly produced by the system must be a complete binary counter row, as all assemblies are either already complete rows (left and right bins) or are partial assemblies (subtree bins and missing bit bins) that can be extended towards the end of the bit string by missing bit bin assemblies, or towards the start of the bit string by missing bit and then subtree bin assemblies.

The second counter generalization is incrementing by non-unitary values:

Lemma 4. *Let k, n be integers such that $0 \leq k \leq n$ and $n = 2^m$. There exists a SSAS of size $O(\log n)$ with a set of bins that, when mixed, assemble a set of 2^m binary counter rows with values $0, 1, \ldots, 2^m - 1$ incremented by k.*

Proof. For each row, the incremented value of the bth bit of the row depends on three values: the previous value of the bth bit, the carry bit from the $(b-1)$st addition, and the bth bit of k. The resulting output is a pair of bits: the resulting value of the bth bit and the bth carry bit (seen in Table 1).

Create a set of four $O(1)$-tile subassemblies for each bit of the counter, selecting from the first or second half of the combinations in Table 1, resulting in $4 \log n$ assemblies total. Each subassembly handles a distinct combination of the bth bit value of the previous row, $(b-1)$st carry bit, and bth bit value of k by encoding each possibility as a distinct glue. When mixed in a single bin, these subassemblies combine in all possible combinations and producing all counter rows from 0 to $2^m - 1$.

Lemma 5. *Let i, j, k, n be integers such that $0 \leq i \leq j < n$ and $0 \leq k \leq n$. There exists a SSAS of size $O(\log n)$ with a set of bins that, when mixed, assemble a set of $j - i + 1$ binary counter rows with values $i, i+1, \ldots, j$ incremented by k.*

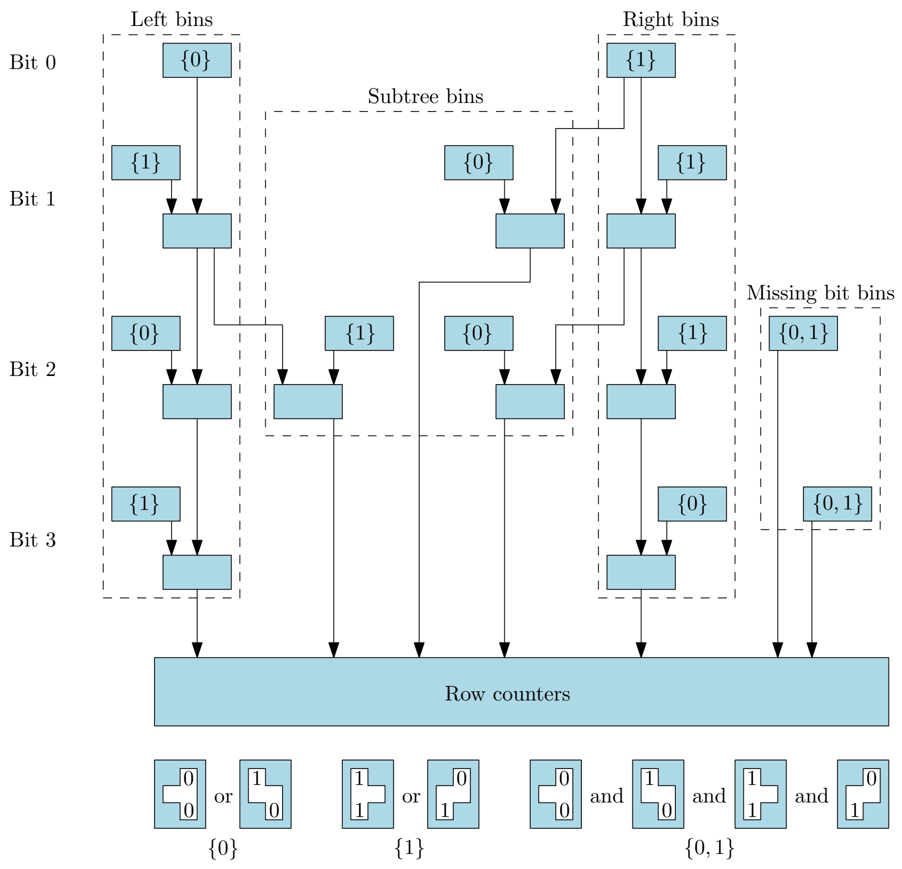


Fig. 6. The mix graph constructed for the prefix subtree $T_{(5,14)}$ seen in Figure 5.

Proof. Combine the constructions used in the proofs of Lemmas 3 and 4 by using mixing sequences as in the proof of Lemma 3 and sets of four subassemblies encoding input, carry, and increment bit values as in the proof of Lemma 4.

Theorem 8 of Demaine et al. [7] describes how to reduce the number of glues used in a system by replacing each tile with a large *macrotile* assembly, and encoding the tile's glues via unique geometry on the macrotile's sides. We prove a similar result for labeled tiles, used for proving Theorems 2, 3, and 7.

Lemma 6. *Any mismatch-free* $\tau = 1$ *SAS (or SSAS)* $\mathcal{S} = (T, G, \tau, M)$ *can be simulated by a SAS (or SSAS)* $\mathcal{S}'$ *at* $\tau = 1$ *with* $O(1)$ *glues, system size* $O(\Sigma(T)|T| + |\mathcal{S}|)$*, and* $O(\log |G|)$ *scale.*

Proof. The proof is constructive. Produce a set of north *macroglue assemblies* for the glue set: $O(\log |G|) \times O(1)$ assemblies, each encoding the integer label of a glue i via a sequence of bumps and dents along the north side of the assembly

Input bits			Output bits	
bth bit of k	bth bit	$(b-1)$st carry bit	bth bit	bth carry bit
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1. All bit combinations for a binary adder incrementing n by k.

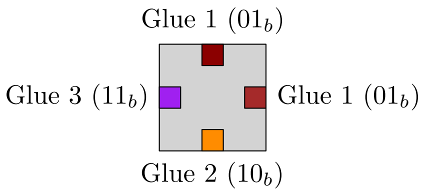
representing the binary sequence of bits for i, as seen in Figure 7. All north macroglue assemblies share a pair of common glues: an *inner glue* on the west end of the south side of the assembly (green in Figure 7) and an *outer glue* on the west end of the north side of the assembly (blue in Figure 7). The null glue also has the sequence of bumps and dents (encoding 0), but lacking the outer glue. Repeating this process three more times yields sets of east, west, and south macroglue assemblies.

For each label $l \in \Sigma(T)$, repeat the process of producing the macroglue assemblies once using a tile set exclusively labeled l. Also produce a square $\Theta(\log |G|) \times \Theta(\log |G|)$ *core assembly*, with a single copy of the inner glue on the counterclockwise end of each face. Use the macroglue and core assemblies to produce a set of *macrotiles*, one for each $t \in T$, consisting of a core assembly whose tiles have the label of t, and four glue assemblies encode the four glues of t and whose tiles have the label of t. Extend the mix graph M' of $\mathcal{S}'$ by carrying out the mixings of M but starting with the equivalent macrotiles. Define the simulation function f to map each macrotile to the label found on the macrotile, and the function g to take the portion of M' and g to be the portion of the mix graph carrying out the mixings of $\mathcal{S}$.

The work done to produce the glue assemblies is $O(\Sigma(T)|G|)$, to produce the core assemblies is $O(\Sigma(T) \log\log |G|)$, and to produce the macrotiles is $O(|T|)$. Carrying out the mixings of $\mathcal{S}$ requires $O(|\mathcal{S}|)$ work. Since each macrotile is used in at least one mixing simulating a mixing in $\mathcal{S}$, $|T| \leq |\mathcal{S}|$. Additionally, $|G| \leq 4|T|$. So the total system size is $O(\Sigma(T)|G| + \Sigma(T) \log\log |G| + |T| + |\mathcal{S}|) = O(\Sigma(T)|T| + |\mathcal{S}|)$.

Armed with these tools, we are ready to convert PCFGs into SSASs. Recall that in Section 4 we showed that in the worst case, converting a PCFG into a SSAS (or SAS) *must* incur an $\Omega(\log n / \log\log n)$-factor increase in system size. Here we achieve a $O(\log n)$-factor increase.

Theorem 2. *For any polyomino P with $|P| = n$ derived by a PCFG G, there exists a SSAS $\mathcal{S}$ with $|\mathcal{S}| = O(|G| \log n)$ producing an assembly with label polyomino P', where P' is a $(O(\log n), O(n))$-fuzzy replica of P.*



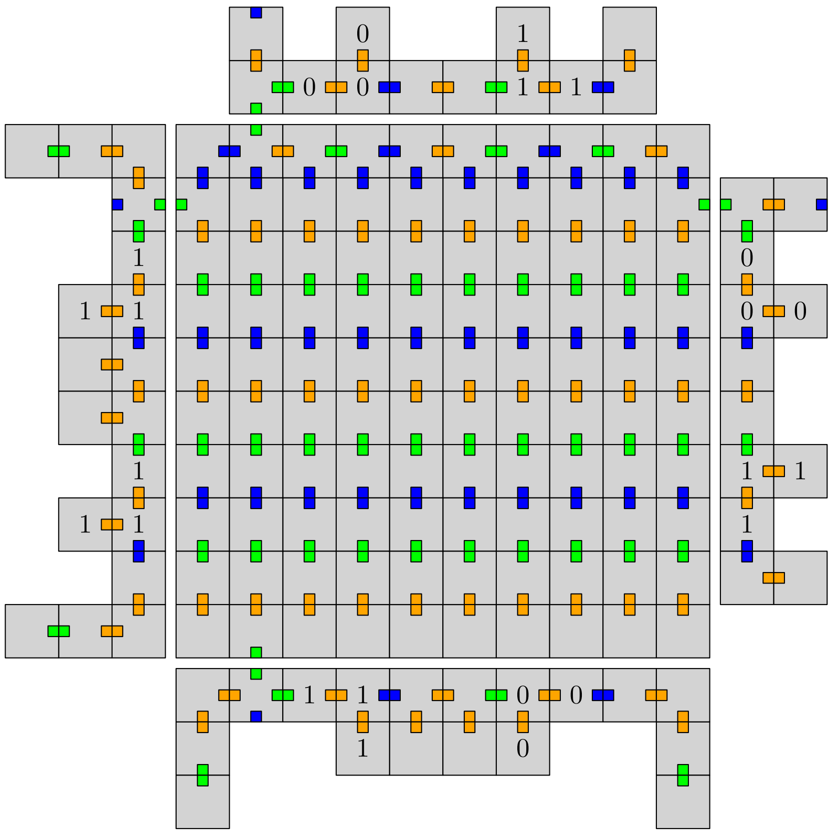


Fig. 7. Converting a tile in a system with 7 glues to a macrotile with $O(\log |G|)$ scale and 3 glues. The gray label of the tile is used as a label for all tiles in the core and macroglue assemblies, with the 1 and 0 markings for illustration of the glue bit encoding.

Proof. We combine the macrotile construction of Lemma 6, the generalized counters of Lemma 5, and a macrotile assembly invariant that together enable efficient simulation of each production rule in a PCFG by a set of $O(\log n)$ mixing steps.

Macrotiles. The macrotiles used are extended versions of the macrotiles in Lemma 6 with two modifications: a secondary, *resevoir macroglue* assembly on each side of the tile in addition to a primary *bonding macroglue*, and a thin *cage* of dimensions $\Theta(n) \times \Theta(\log n)$ surrounding each resevoir macroglue (see Figure 8).

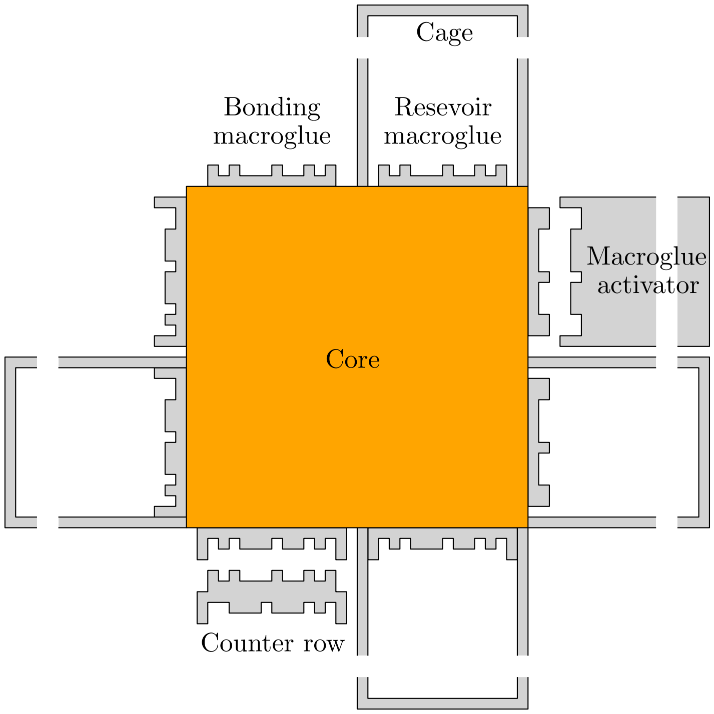


Fig. 8. A macrotile used in converting a PCFG to a SAS, and examples of value maintenance and offset preparation.

Mixing a macrotile with a set of bins containing counter row assemblies constructed by Lemma 5 causes completed (and incomplete) counter rows to attach to the macrotile's macroglues. Because each macroglue's geometry matches the geometry of exactly one counter row, a partially completed counter row that attaches can only be completed with bit assemblies that match the macroglue's value. As a result, mixing the bin sets of Lemma 5 with an assembly consisting of macrotiles produces the same set of products as mixing a completed set of binary counter rows with the assembly.

An attached counter row effectively causes the macroglue's value to change, as it presents geometry encoding a new value and covers the macroglue's previous value. The cage is constructed to have height sufficient to accomodate up to n counter rows attached to the reservoir macroglue, but no more.

Because of the cage, no two macrotiles can attach by their bonding macroglues unless the macroglue has more than n counter rows attached. Alternatively, one can produce a thickened counter row with thickness sufficient to extend beyond the cage. We call such an assembly a *macroglue activator*, as it "activates" a bonding macroglue to being able to attach to another promoted macroglue on another macrotile. Notice that a macroglue activator will never attach to a bonding macroglue's resevoir twin, as the cage is too small to contain the activator.

An invariant. Counter rows and activators allow precise control of two properties of a macrotile: the identities of the macroglues on each side, and whether these glues are activated. In a large assembly containing many macroglues, the ability to change and activate glues allows precise encoding of how an assembly can attach to others. In the remainder of the construction we maintain the invariant that every macrotile has the same glue identity on all four sides, and any macrotile assembly consists of macrotiles with glue identities forming a contiguous interval, e.g. 4, 5, 6, 7. Intervals are denoted $[i, i']$, e.g. $[g_4, g_7]$.

By Lemma 5, a set of row counters incrementing the glue identities of *all* glues on a macrotile can be produced using $O(\log n)$ work. Activators, by virtue of being nearly rectangular with $O(\log n)$ cells of bit geometry can also be produced using $O(\log n)$ work.

Production rule simulation. Consider a PCFG with non-terminal N and production rule $N \to (R_1, (x_1, y_1))(R_2, (x_2, y_2))$ and a SSAS with two bins containing assemblies A_1, A_2 with the label polyominoes of A_1 and A_2 being fuzzy replicas of the polyominoes derived by R_1 and R_2. Also assume A_1 and A_2 are assembled from the macrotiles just described, including the invariant that the identities of the glues on A_1 and A_2 are identical on all sides of a macrotile and contiguous across the assembly, i.e. the identities of the glues are $[i_1, j_1]$ and $[i_2, j_2]$ on assemblies A_1 and A_2, respectively.

Select two cells c_{R_1}, c_{R_2}, in the polyominoes derived by R_1 and R_2 adjacent in polyomino derived by N. Define the glue identities of the two macrotiles forming the supercells mapped to c_{R_1} and c_{R_2} to be g_1 and g_2. Then the glue sets on A_1 and A_2 can be decomposed into three subsets $[i_1, g_1 - 1]$, $[g_1]$, $[g_1 + 1, j_1]$ and $[i_2, g_2 - 1]$, $[g_2]$, $[g_2 + 1, j_2]$, respectively. We change these glue values in three steps:

1. Construct two sets of row counters that increment i_1 through g_1 by $j_1 - i_1 + 1$ and i_2 through g_2 by $g_2 - i_2 + 1$, and mix them in separate bins with A_1 and A_2 to produce two new assemblies A'_1 and A'_2. Assemblies A'_1 and A'_2 have glues $[g_1 + 1, g_1 + j_1 - i_1 + 1]$ and $[g_2, g_2 + j_2 - i_2]$, respectively, and the macroglues with values g_1 and g_2 now have values $g'_1 = g_1 + (g_1 - i_1) + j_1 + 1$ and $g'_2 = g_2$, i.e. the glues of A'_1 and A'_2 are $[g'_1 - (j_1 - i_1), g'_1]$ and $[g'_2, g'_2 + j_2 - i_2]$.
2. Construct a set of row counters that increment the values of all glues on A'_2 by $g'_2 - g'_1 + 1$ if this value is positive, and mix the counters with A'_2 to produce A''_2. Then the macroglue with value g'_2 now has value $g''_2 = g'_1 + 1$ and the glue values of A'_1 and A''_2 are $[g'_1 - (j_1 - i_1), g'_1]$ and $[g''_2, g''_2 + j_2 - i_2]$.
3. Construct a pair of macroglue activators with values g'_1 and g''_2 that attach to the pair of macroglue sides matching the two adjacent sides of cells c_{R_1} and c_{R_2}. Mix each activator with the corresponding assembly A'_1 or A''_2.

Mixing A'_1 and A''_2 with the pair of activated macroglues causes them to bond in exactly one way to form a superassembly A_3 whose label polyomino is a fuzzy replica of the polyomino derived by N. Moreover, the glue values of the macrotiles in A_3 are $[g'_1-(j_1-i_1), g''_2+j_2-i_2]$, maintaining the invariant. Because each macrotile has a resevoir macroglue on each side, any bonding macroglue with an activator already attached has a resevoir macroglue that accepts the matching row counter, so each mixing has a single product and specifically no row counter products.

System scale The PCFG P contains at most n production rules. Also, each step shifts glue identities by at most n (the number of distinct glues on the macrotile), so the largest glue identity on the final macrotile assembly is n^2. So we produce macrotiles with core assemblies of size $O(\log n) \times O(\log n)$ and cages of size $O(n)$. Assembling the core assemblies, cages, and initial macroglue assemblies of the macrotiles takes $O(|P| \log n + \log n + \log n) = O(|P| \log n)$ work, dominated by the core assembly production. Simulating each production rule of the grammar takes $O(\log n)$ work spread across a constant number of $O(\log n)$-sized sequences of mixings to produce sets of row counters and macroglue activators.

Applying Lemma 6 to the construction (creating macrotiles of macrotiles) gives a constant-glue version of Theorem 2:

Theorem 3. *For any polyomino P with $|P| = n$ derived by a PCFG G, there exists a SSAS $\mathcal{S}'$ using $O(1)$ glues with $|\mathcal{S}'| = O(|G| \log n)$ producing an assembly with label polyomino P', where P' is a $(O(\log n \log\log n), O(n \log\log n))$-fuzzy replica of P.*

Proof. The construction of Theorem 2 uses $O(\log n)$ glues, namely for the counter row subconstruction of Lemma 5. With the exception of the core assemblies, all tiles of S have a common fuzz (gray) label, so creating macrotile versions of these tiles and carrying out all mixings involving these macrotiles and *completed* core assemblies is possible with $O(1 \cdot |T| + |\mathcal{S}|) = O(|\mathcal{S}|)$ mixings and scale $O(\log\log n)$. Scaled core assemblies of size $\Theta(n \log\log n) \times \Theta(n \log\log n)$ can be constructed using constant glues and $O(\log(n \log\log n)) = O(\log n)$ mixings, the same number of mixings as the unscaled $\Theta(n) \times \Theta(n)$ core assemblies of Theorem 2. So in total, this modified construction has system size $O(|S|) = O(|G| \log n)$ and scale $O(\log\log n)$. Thus it produces an assembly with label polyomino that is a $(O(\log n \log\log n), O(n \log\log n))$-fuzzy replica of P.

The results in this section and Section 4 achieve a "one-sided" correspondence between the smallest PCFG and SSAS encoding a polyomino, i.e. the smallest PCFG is approximately an *upper bound* for the smallest SSAS (or SAS). Since the separation upper bound proof (Theorem 2) is constructive, the bound also yields an algorithm for converting a a PCFG into a SSAS.

6 PCFG over SAS and SSAS Separation Lower Bound

Here we develop a sequence of PCFGs over SAS and SSAS separation results, all within a polylogarithmic factor of optimal. The results also hold for polynomially scaled versions of the polyominoes, which is used to prove Theorem 7 at the end of the section. This scale invariance also surpasses the scaling of the fuzzy replicas in Theorems 2 and 3, implying that this relaxation of the problem statement in these theorems was not unfair.

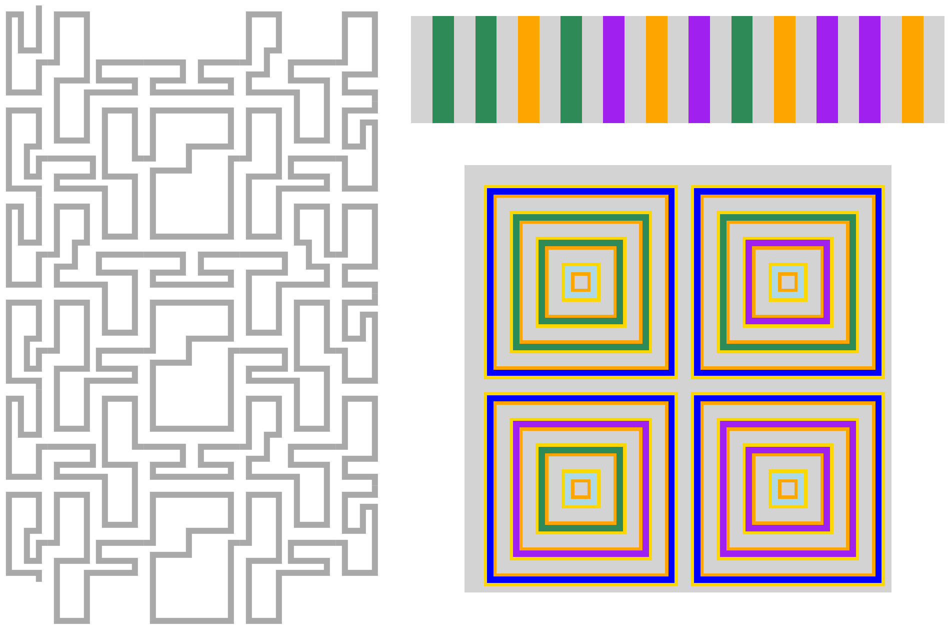


Fig. 9. Two-bit examples of the weak (left), end-to-end (upper right), and block (lower right) binary counters used to achieve separation of PCFGs over SASs and SSASs in Section 6.

6.1 General shapes

In this section we describe an efficient system for assembling a set of shapes we call *weak counters*. An example of a rows in the original counter and macrotile weak counter are shown in Figure 10. These shapes are macrotile versions of the doubly-exponential counters found in [7] with three modifications:

1. Each row is a single path of tiles, and any path through an entire row uniquely identifies the row.
2. Adjacent rows do not have adjacent pairs of tiles, i.e. they do not touch.
3. Consecutive rows attach at alternating (east, west, east, etc.) ends.

Figure 11 shows three consecutive counter rows attached in the final assembly. Each row of the doubly-exponential counter consists of small, constant-sized

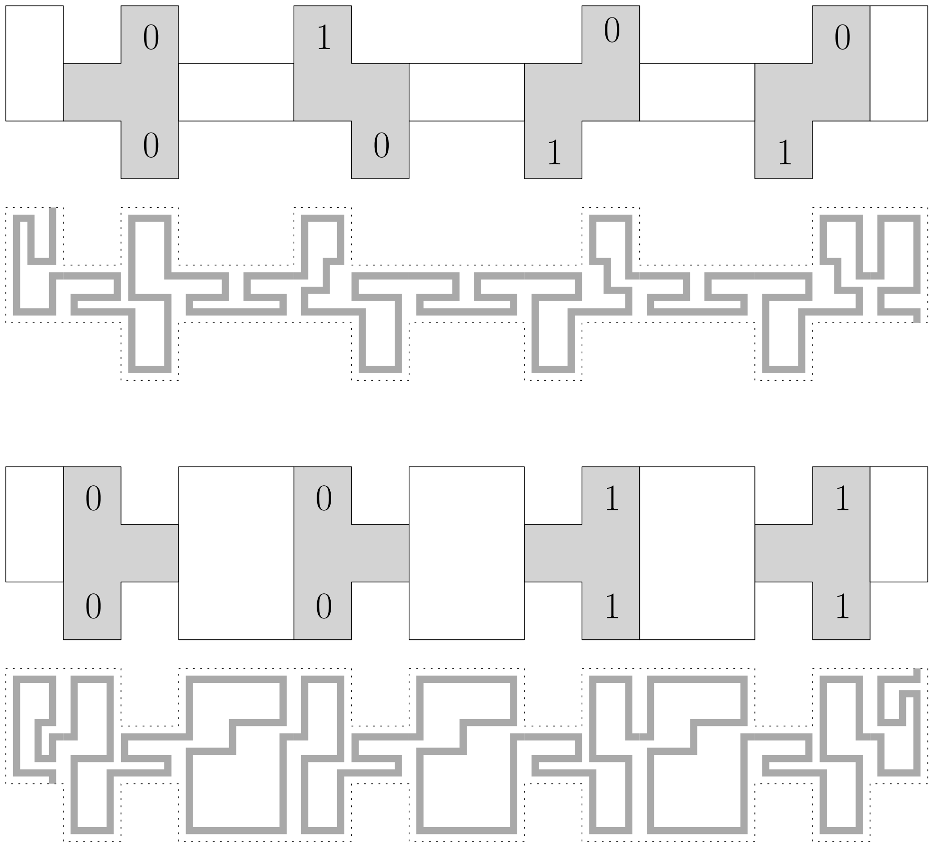


Fig. 10. Zoomed views of increment (top) and copy (bottom) counter rows described in [7] and the equivalent rows of a weak counter.

assemblies corresponding to 0 or 1 values, along with a 0 or 1 carry bit. We implement each assembly as a unique path of tiles and assemble the counter as in [7], but using these path-based assemblies in place of the original assemblies. We also modify the glue attachments to alternate on east and west ends of each row. Because the rows alternate between incrementing a bit string, and simply encoding it, alternating the attachment end is trivial. Finally, note that adjacent rows only touch at their attachment, but the geometry encoded into the row's path prevents non-consecutive rows from attaching.

Lemma 7. *There exists a $\tau = 1$ SAS of size $O(b)$ that produces a 2^b-bit weak counter.*

Proof. The counter is an $O(1)$-scaled version of the counter of Demaine et al [7]. They show that such an assembly is producible by a system of size $O(b)$.

Lemma 8. *For any PCFG G deriving a 2^b-bit weak counter, $|G| = \Omega(2^{2^b})$.*

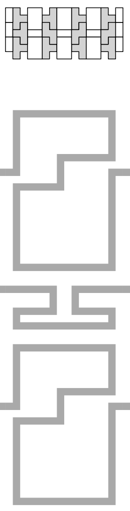


Fig. 11. Adjacent attached rows of the counter described in [7] (top) and the equivalent rows in the weak counter (bottom).

Proof. Define a *minimal row spanner* of row $\mathcal{R}_i$ to be a non-terminal symbol N of G with production rule $N \to (B, (x_1, y_1))(C, (x_2, y_2))$ such that the polyomino derived by N contains a path between a pair of easternmost and westernmost tiles of the row and the polyominoes derived by B and C do not. We claim that each row (trivially) has at least one minimal row spanner and each non-terminal of G is a minimal row spanner of at most one unique row.

First, suppose by contradiction that a non-terminal N is a minimal row spanner for two distinct rows. Because N is connected and two non-adjacent rows are only connected to each other via an intermediate row, N must be a minimal row spanner for two adjacent rows $\mathcal{R}_i$ and $\mathcal{R}_{i+1}$. Then the polyominoes of B and C each contain tiles in both $\mathcal{R}_i$ and $\mathcal{R}_{i+1}$, as otherwise either C or B is a minimal row spanner for $\mathcal{R}_i$ or $\mathcal{R}_{i+1}$.

Without loss of generality, assume B contains a tile at the end of $\mathcal{R}_i$ not adjacent to $\mathcal{R}_{i+1}$. But B also contains a tile in $\mathcal{R}_{i+1}$ and (by definition) is connected. So B contains a path between the east and west ends of row $\mathcal{R}_i$, and thus N is a not a minimal row spanner for r_i. So N is a minimal row spanner for at most one row.

Next, note that the necessarily-serpentine path between a pair of easternmost and westernmost tiles of a row in a minimal row spanner uniquely encodes the row it spans. So the row spanned by a minimal row spanner is unique.

Because each non-terminal of G is a minimal row spanner for at most one unique row, G must have at least 2^{2^b} non-terminal symbols and total size $\Omega(2^{2^b})$.

Theorem 4. *The separation of PCFGs over $\tau = 1$ SASs for single-label polyomines is $\Omega(n/(\log \log n)^2)$.*

Proof. By the previous two lemmas, there exists a SAS of size $O(b)$ producing a b-bit weak counter, and any PCFG deriving this shape has size $\Omega(2^{2^b})$. The assembly itself has size $n = \Theta(2^{2^b} b)$, as it consists of 2^{2^b} rows, each with b subassemblies of constant size. So the separation is $\Omega((n/b)/b) = \Omega(n/(\log \log n)^2)$.

In [7], the $O(\log \log n)$-sized SAS constructing a $\log n$-bit binary counter repeatedly doubles the length of each row (i.e. number of bits in the counter) using $O(1)$ mixings per doubling. Achieving such a technique in a SSAS seems impossible, but a simpler construction producing a b-bit counter with $O(b)$ work can be done by using a unique set of $O(1)$ glues for each bit of the counter. In this case, mixing these reusable elements along with a previously-constructed pair of first and last counter rows creates a single mixing assembling the entire counter at once. Modifying the proof of Theorem 4 to use this construction gives a similar separation for SSASs:

Corollary 1. *The separation of PCFGs over $\tau = 1$ SSASs for single-label polyominoes is $\Omega(n/\log^2 n)$.*

6.2 Rectangles

For the weak counter construction, the lower bound in Lemma 8 depended on the poor connectivity of the weak counter polyomino. This dependancy suggests that such strong separation ratios may only be achievable for special classes of "weakly connected" or "serpentine" shapes. Restricting the set of shapes to rectangles or squares while keeping an alphabet size of 1 gives separation of at most $O(\log n)$, as any rectangle of area n can be derived by a PCFG of size $O(\log n)$.

But what about rectangles with a constant-sized alphabet? In this section we achieve surprisingly strong separation of PCFGs over SASs and SSASs for rectangular constant-label polyominoes, nearly matching the separation achieved for single-label general polyominoes.

The construction The polyominoes constructed resemble binary counters whose rows have been arranged in sequence horizontally, and we call them *b-bit end-to-end counters*. Each row of the counter is assembled from tall, thin macrotiles (called *bars*), each containing a *color strip* of orange, purple, or green. The color strip is coated on its east and west faces with gray geometry tiles that encode the bar's location within the counter.

Each row of the counter has a sequence of green and purple *display bars* encoding a binary representation of the row's value and flanked by orange *reset bars* (see Figure 13). An example for $b = 2$ bits can be seen in Figure 12.

Each bar has dimensions $O(1) \times 3(\log_2 b + b + 2)$, sufficient for encoding two pieces of information specifying the location of the bar within the assembly. The *row bits* specify which row the bar lies in (e.g. the 7th row). The *subrow bits* specify where within the row the bar lies (e.g. the 4th bit). The subrow value starts at 0 on the east side of a reset bar, and increments through the display

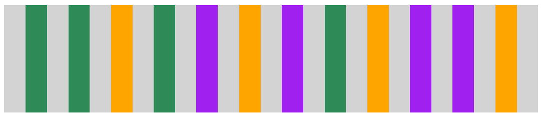


Fig. 12. The rectangular polyomino used to show separation of PCFGs over SASs when constrained to constant-label rectangular polyominoes. The green and purple color strips denote 0 and 1 bits in the counter.

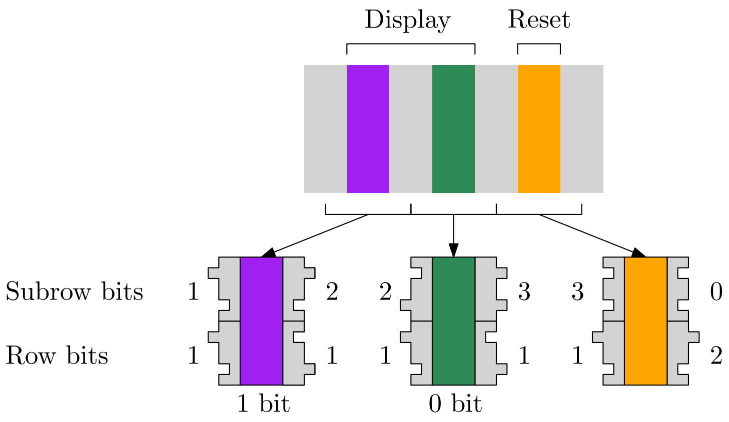


Fig. 13. The implementation of the vertical bars in row 2 (01_b) of an end-to-end counter.

bars until reaching $b+1$ on the west end of the next reset bar. Bars of all three types with row bits ranging from 0 to $2^b - 1$ are produced.

Efficient assembly The counter is constructed using a SAS of size $O(b)$ in two phases. First, sequences of $O(b)$ mixings are used to construct five families of bars: 1. reset bars, 2. 0-bit display bars resulting from a carry, 3. 0-bit display bars without a carry, 4. 1-bit display bars resulting from a carry, 5. 1-bit display bars without a carry, The mixings product five bins, each containing *all* of the bars in the family. These five bins are then combined into a final bin where the bars attach to form the $\Theta(2^b) \times \Theta(b)$ rectangular assembly. The five families are seen in Figure 14.

Efficient $O(b)$ assembly is achieved by careful use of the known approach of non-deterministic assembly of single-bit assemblies as done in [7]. Assemblies encoding possible input bit and carry bit value combinations for each row bit and subrow bit are constructed and mixed together, and the resulting products are every valid set of input and output bit strings, i.e. every row of a binary counter assembly.

As a warmup, consider the assembly of all reset bars. For these bars, the west subrow bits encode b and the east subrow bits encode 0. The row bits encode a value i on the west side, and $i+1$ on the east side, for all i between 0 and $2^b - 1$.

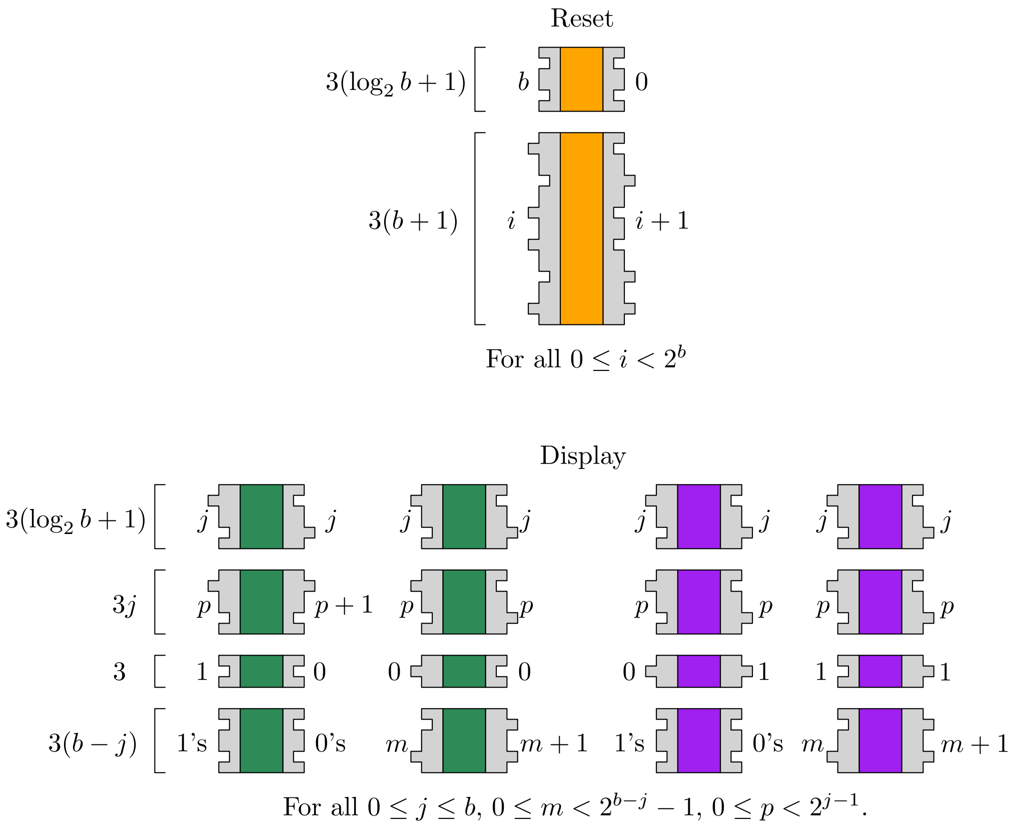


Fig. 14. The decomposition of bars used assemble a b-bit end-to-end counter.

Constructing all such bars using $O(b)$ work is straightforward. For each of the $\log_2 b + 1$ subrow bits, create an assembly where the west and east bits are 1 and 0 respectively, *except* for the most significant bit (bit $\log_2 b + 1$), where the west and east bits are both 0.

For the row bits we use the same technique as in [7] and extended in Lemma 4: create a constant-sized set of assemblies for each bit that encode input and output value and carry bits. For bits 1 through $b-1$ (zero-indexed) create four assemblies corresponding to the four combinations of value and carry bits, for bit 0 create two assemblies corresponding to value bits (the carry bit is always 1), for bit b create three assemblies corresponding to all combinations except both value and carry bits valued 1, and for bit $b+1$ create a single assembly with both bits valued 0. Give each bit assembly a unique south and north glue encoding its location within the bar and carry bit value, and give all bit assemblies a common orange color strip. Mixing these assemblies produces all reset bars, with subrow west and east values of b and 0, and row values i and $i+1$ for all i from 0 to $2^b - 1$.

In contrast to producing reset bars, producing display bars is more difficult. The challenge is achieving the correct color strip relative to the subrow and row values. Recall that the row value i locates the bar's row and the subrow value j locates the bar within this row. So the correct color strip for a bar is green if the jth bit of i is 0, and purple if the jth bit of i is 1.

We produce four families of display bars, two for each value of the jth bit of of i. Each subfamily is produced by mixing a subrow assembly encoding j on both east and west ends with three component assemblies of the row value: the *least significant bits (LSB) assembly* encoding bits 1 through $j-1$ of i, the *most significant bits (MSB) assembly* encoding bits $j+1$ through b of i, and the constant-sized jth bit assembly. This decomposition is seen in the bottom half of Figure 14.

The four families correspond to the four input and carry bit values of the jth bit. These values determine what collections of subassemblies should appear in the other two components of the row value. For instance, if the input and carry bit values are both 1, then the LSB assembly must have all 1's on its west side (to set the jth carry bit to 1) and all 0's on its east side. Similarly, the MSB assembly must have some value p encoded on its west side and the value $p+1$ encoded on its east side, since the jth bit and and jth carry bit were both 1, so the $(j+1)$st carry bit is also 1.

Notice that each of the four families has b subfamilies, one for each value of j. Producing all subfamilies of each family is possible in $O(b)$ work by first recursively producing a set of b bins containing successively larger sets of MSB and LSB assemblies for the family. Then each subfamily can be produced using $O(1)$ amortized work, mixing one of b sets of LSB assembly subfamilies, one of b sets of MSB assemblies, and the jth bit assembly together. For instance, one can produce the set of b sets of MSB assemblies encoding pairs of values p and $p+1$ on bits $b-1$ through b, $b-2$ through b, etc. by producing the set on bits k through b, then adding four assemblies to this bin (those encoding possible pairs of inputs to the $(k-1)$st bit) to produce a similar set on bits $k-1$ through b.

Lemma 9. *There exists a $\tau = 1$ SAS of size $O(b)$ that produces a b-bit end-to-end counter.*

Proof. This follows from the description of the system. The five families of bars can each be produced with $O(b)$ work and the bars can be combined together in a single mixing to produce the counter. So the system has total size $O(b)$.

Lemma 10. *For any PCFG G deriving a b-bit end-to-end counter, $|G| = \Omega(2^b)$.*

Proof. Let G be a PCFG deriving a b-bit end-to-end counter. Define a *minimal row spanner* to be a non-terminal symbol N with production rule $N \rightarrow (B,(x_1,y_1))(C,(x_2,y_2))$ such that the polyomino derived by N (denoted p_N) horizontally spans the color strips of all bars in row $\mathcal{R}_i$ including the reset bar at the end of the row, while the polyominoes derived by B and C (denoted

p_B and p_C) do not. Consider the bounding box D of these color strips (see Figure 15).

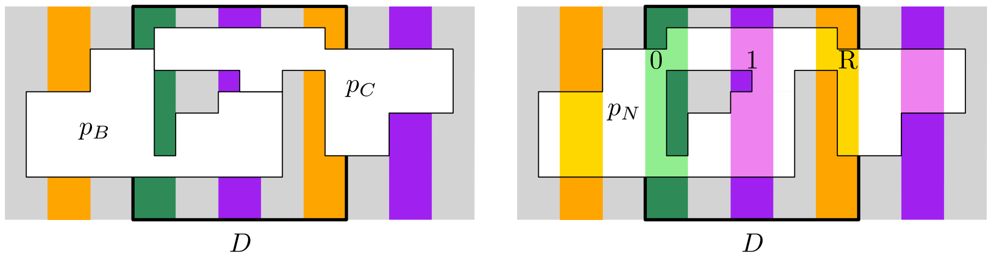


Fig. 15. A schematic of the proof that a non-terminal is a minimal row spanner for at most one unique row. (Left) Since p_B and p_C can only touch in D, their union non-terminal N must be a minimal row spanner for the row in D. (Right) The row's color strip sequence uniquely determines the row spanned by N (01_b).

Without loss of generality, p_B intersects the west boundary of D but does not reach the east boundary, while p_C intersects the east boundary but does not reach the west boundary, so any location at which p_B and p_C touch must lie in D. Then any row spanned by p_N and not spanned by p_B or p_C must lie in D, since spanning it requires cells from both p_B and p_C. So p_N is a minimal row spanner for at most one row: row $\mathcal{R}_i$.

Because the sequence of green and purple display bars found in D is distinct and separated by display bars in other rows by orange reset bars, each minimal row spanner spans a unique row $\mathcal{R}_i$. Then since each non-terminal is a spanner for at most one unique row, G must have 2^b non-terminal symbols and $|G| = \Omega(2^b)$.

Theorem 5. *The separation of PCFGs over $\tau = 1$ SASs for constant-label rectangles is $\Omega(n/\log^3 n)$.*

Proof. By construction, a b-bit end-to-end counter has dimensions $\Theta(2^b b) \times \Theta(b)$. So $n = \Theta(2^b b^2)$ and $b = \Theta(\log n)$. Then by the previous two lemmas, the separation is $\Omega((n/b^2)/b) = \Omega(n/\log^3 n)$.

We also note that a simple replacement of orange, green, and purple color strips with distinct horizontal sequences of black/white color substrips yields the same result but using fewer distinct labels.

6.3 Squares

The rectangular polyomino of the last section has exponential aspect ratio, suggesting that this shape requires a large PCFG because it approximates a patterned one-dimensional assemblies reminiscent of those in [8]. Creating a polyomino with better aspect ratio but significant separation is possible by extending the polyomino's labels vertically. For a square this approach gives a separation of PCFGs over SASs of $\Omega(\sqrt{n}/\log n)$, non-trivial but far worse than the rectangle.

The construction In this section we describe a polyomino that is square but contains an exponential number of distinct subpolyominoes such that each subpolyomino has a distinct "minimal spanner", using the language of the proof of Lemma 10. These subpolyominoes use circular versions of the vertical bars of the construction in Section 6.2 arranged concentrically rather than adjacently. We call the polyomino a *b-bit block counter*, and an example for $b = 2$ is seen in Figure 16.

Each *block* of the counter is a $\Theta(b^2) \times \Theta(b^2)$ square subpolyomino encoding a sequence of b bits via a sequence of concentric rectangular *rings* of increasing size. Each ring has a *color loop* encoding the value of a bit, or the start or end of the bit sequence (the interior or exterior of the block, respectively). The color loop actually has three subloops, with the center loop's color (green, purple, light blue, or dark blue in Fig. 16) indicating the bit value or sequence information, and two surrounding loops (light or dark orange in Fig. 16) indicating the interior and exterior sides of the loop.

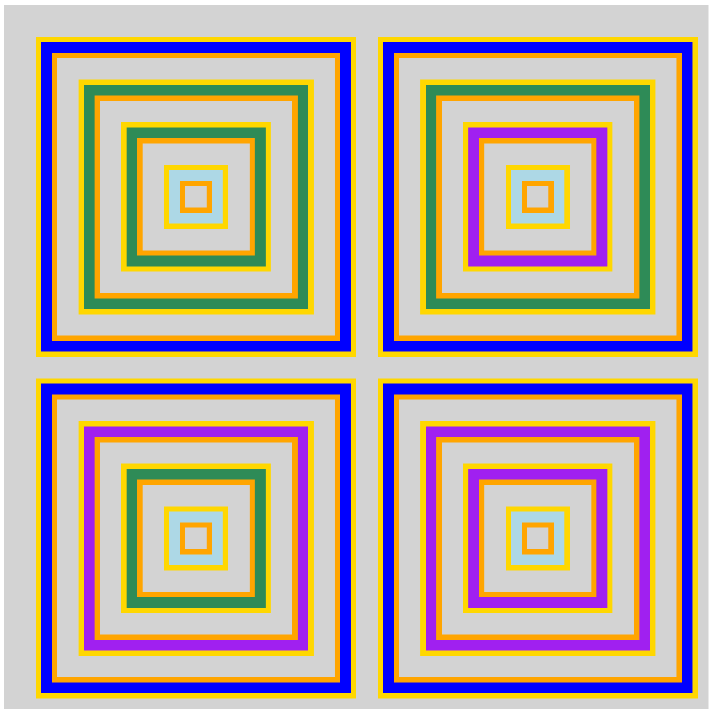


Fig. 16. The square polyomino used to show separation of PCFGs over SASs when constrained to constant-label square polyominoes. The green and purple color subloops denote 0 and 1 bits in the counter, while the light and dark blue color subloops denote the start and end of the bit string. The light and dark orange color subloops indicate the interior and exterior of the other subloops.

Efficient assembly of blocks Though each counter block is square, they are constructed similarly to the end-to-end counter rows of Section 6.2 by assembling the vertical *bars* of each ring together into horizontal stacks of assemblies. Horizontal *slabs* are added to "fill in" the remaining portions of each block.

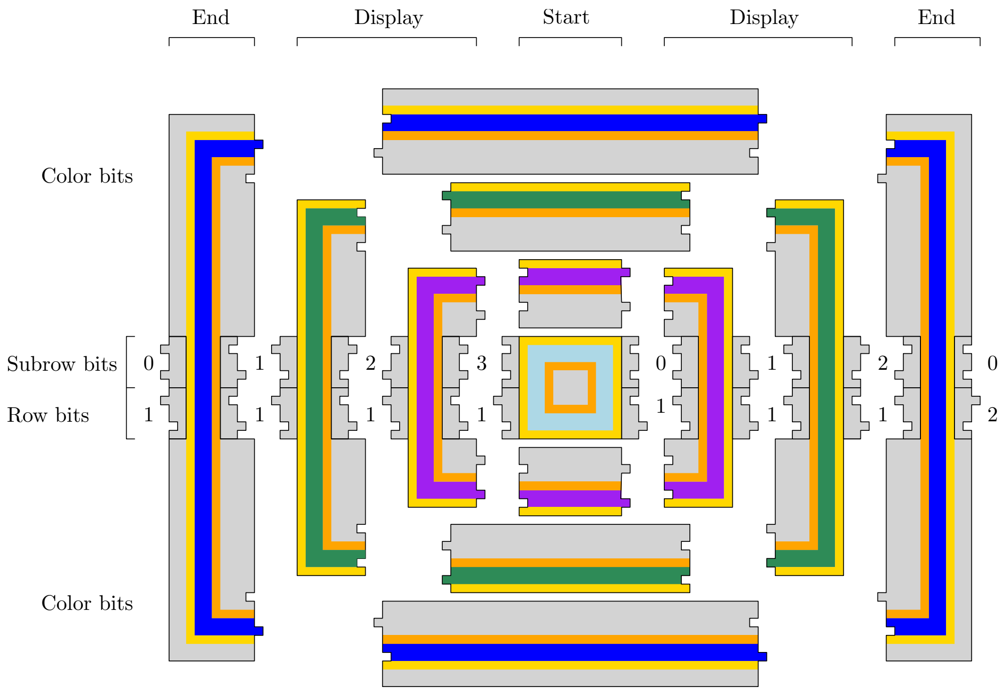


Fig. 17. The implementation of rings in each block of the block counter.

The bars are identical to those found in Section 6.2 with three modifications (seen in Figure 17). First, each bar has additional height according to the value of the subrow bits (8 tiles for every increment of the bits). Second, each bar has four additional layers of tiles on the side (east or west) facing the interior of the block, with *color bits* at the north and south ends of the side encoding three values: 11_b (if the center color subloop is purple, a 1-bit), 00_b (if the center color subloop is green, a 0-bit), or 01_b (if the center color subloop is dark blue, the end of the bit sequence). The additional layers are used to fill in gaps between adjacent rings left by protruding geometry, and the bit values are used to control the attachment of the horizontal slabs of each ring.

Third, the reset bars used in Section 6.2 are replaced with two kinds of bars: *start bars* and *end bars*, seein in Figure 19. End bars form the outermost rings of each block, and the start bars form the square cores of each block. Both start and end bars "reset" the subrow counters, and the east end bars increment the row value.

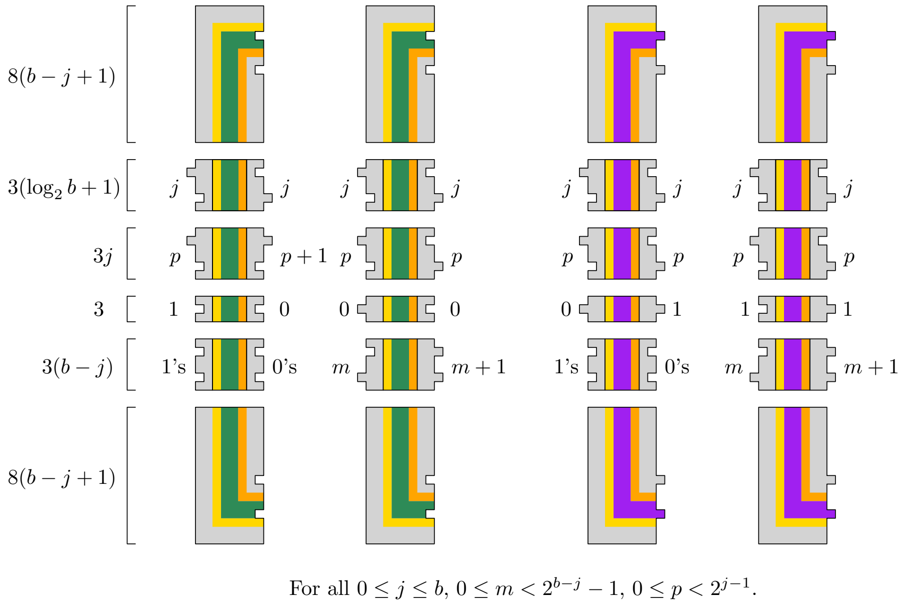


Fig. 18. The decomposition of vertical display bars used to assemble blocks in the b-bit block counter. Only the west bars are shown, with east bars identical but color bits and color loops reflected.

Recall that the vertical bars of the end-to-end counter in Section 6.2 were constructed using $O(b)$ total work by amortizing the constructing subfamilies of MSB and LSB assemblies for each subrow value j. We use the same trick here for these assemblies as well as the new assemblies on the north and south ends of each bar containing the color bits. In total there are twelve families of vertical bar assemblies (four families of west display bars, four families of east display bars, and two families each of start and end bars), and each is assembled using $O(b)$ work.

Finally, the horizontal slabs of each ring are constructed as six families, each using $O(b)$ work, as seen in Figure 20.

Efficient assembly of the counter Once the families of vertical bars and horizontal slabs are assembled into blocks, we are ready to arrange them into a completed counter. Each row of the counter has $\sqrt{2^b} = 2^{b/2}$ blocks. So assuming b is even, the $b/2$ least significant bits of the westmost block of each row are 0's, and of the eastmost block are 1's. Before mixing the vertical bar families together, we "cap" the east end bar of each block at the east end of a row by constructing a set of thin assemblies (right part of Figure 21) and mixing them with the family of east end bars.

After this modification to the east end bar family, mixing all vertical bar families results in $2^{b/2}$ assemblies, each forming most of a row of the block

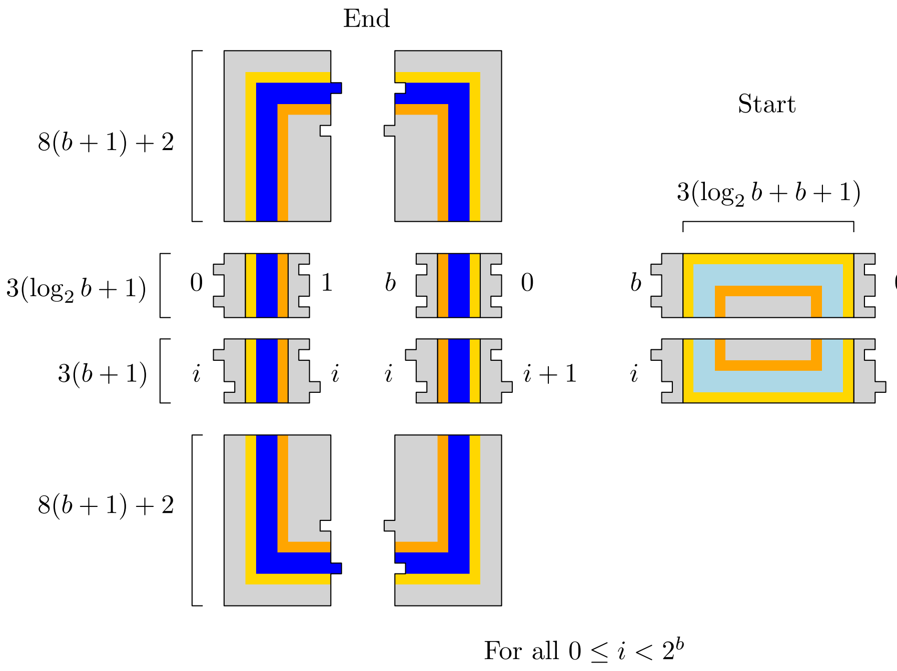


Fig. 19. The decomposition of vertical start and end bars used to assemble blocks in the b-bit block counter.

counter. Mixing these assemblies with the families of horizontal slabs results in a completed set of block counter rows, each containing $2^{b/2}$ square assemblies with dimensions $\Theta(b^2) \times \Theta(b^2)$, forming $2^{b/2}$ rectangles with dimensions $\Theta(2^{b/2}b^2) \times \Theta(b^2)$.

To arrange the rows vertically into a complete block counter, a vertically-oriented version of the end-to-end counter of Section 6.2 with geometry instead of color strips (left part of Fig. 21) is assembled and used as a "backbone" for the rows to attach into a combined assembly. This modified end-to-end counter (see Figure 22) has subrow values from 0 to $b/2$, for the $b/2$ most signficant bits of the row value of each block, and row values from 0 to $2^{b/2}$. Modified versions of reset bars with height (width in the horizontal end-to-end counter) $\Theta(b^2)$ are used to bridge across the geometry-less portions of the west sides of the blocks, as well as the always-zero $b/2$ least significant bits of the block's row value and subrow $\log_2 b$ bits.

This modified end-to-end counter can be assembled using $O(b)$ work as done for the original end-to-end counter, since the longer reset bars only add $O(\log(b^2)) = O(\log b)$ work to the assembly process. After the vertical end-to-end counter has been combined with the blocks to form a complete block counter, a horizontal end-to-end counter is attached to the top of the assembly to produce a square assembly.

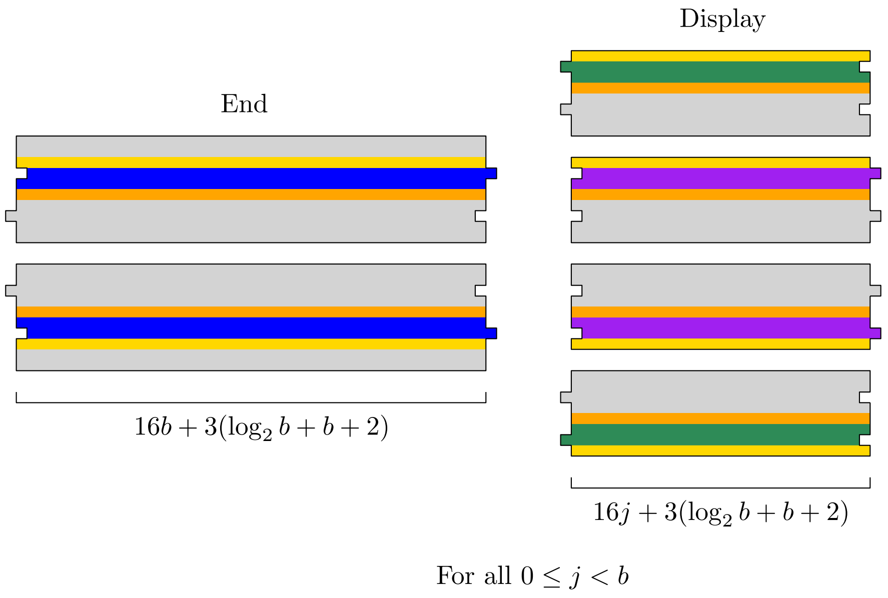


Fig. 20. The decomposition of horizontal slabs of each ring the b-bit block counter.

Lemma 11. *For even b, there exists a $\tau = 1$ SAS of size $O(b)$ that produces a b-bit block counter.*

Proof. The construction described builds families of vertical bars and horizontal slabs that are used to assemble each the rings forming all blocks in the counter. There are a constant number of families, and each family can be assembled using $O(b)$ work. The vertical and horizontal end-to-end counters can also be assembled using $O(b)$ work each by Lemma 9. Then the b-bit block counter can be assembled by a SAS os size $O(b)$.

We now consider a lower bound for any PCFG G deriving the counter, using a similar approach as Lemma 10.

Lemma 12. *For any PCFG G deriving a b-bit block counter, $|G| = \Omega(2^b)$.*

Proof. Define a *minimal block spanner* as to be a non-terminal symbol N in G with production rule $N \to (B, (x_1, y_1))(C, (x_2, y_2))$ such that the polyomino derived by N (denoted p_N) contains a path from a gray cell outside the color loop of the end ring of the counter to a gray cell inside the start color loop of the counter, and the polyominoes derived by B and C (denoted p_B and p_C) do not.

First we show that any minimal block spanner is a spanner for at most one block. Assume by contradiction and that N is a minimal block spanner for two blocks $\mathcal{B}_i$ and $\mathcal{B}_j$ and that p_B contains a gray cell inside the start color loop of $\mathcal{B}_i$. Then B must be entirely contained in the color loop of the end ring of $\mathcal{B}_i$, as

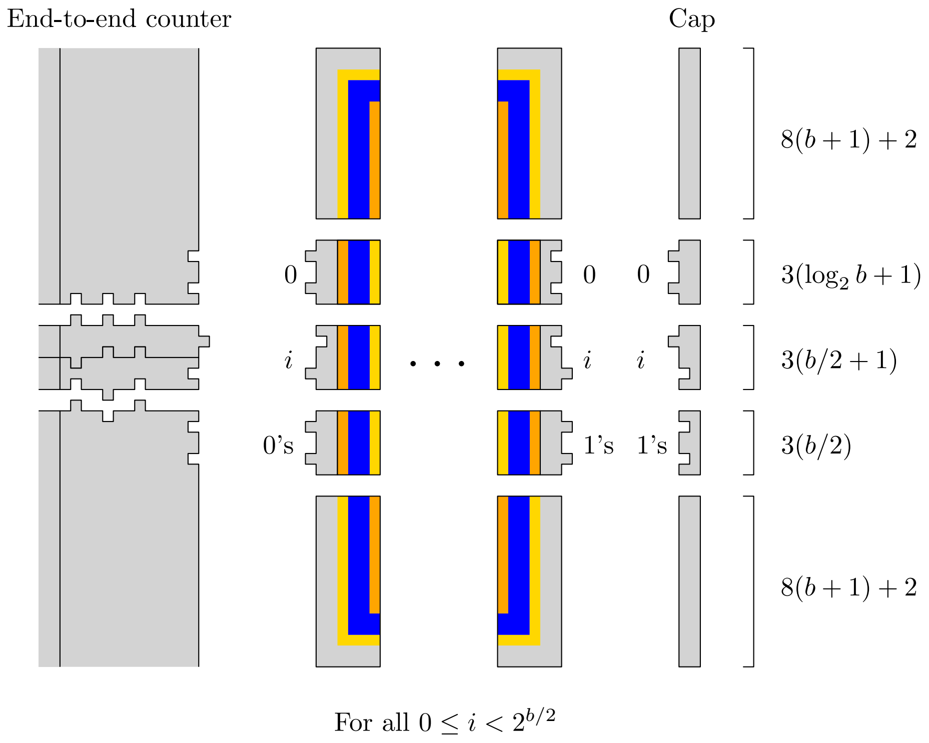


Fig. 21. (Left) The interaction of a vertical end-to-end counter with the westernmost block in each row. (Right) The cap assemblies built to attach to the easternmost block in each row.

otherwise N is not a minimal block spanner for $\mathcal{B}_i$. Similarly, C must then be entirely contained in the color loop of the end ring of $\mathcal{B}_j$. Since no pair of color loops from distinct blocks have adjacent cells, p_N is not a connected polyomino and so G is not a valid PCFG.

Next we show that the block spanned by N is unique, i.e. N cannot be reused as a minimal spanner for multiple blocks. See Figure 23. Let N be a minimal spanner for a block $\mathcal{B}_i$ and p be a path of cells in p_N starting at a gray cell contained in the start ring of $\mathcal{B}_i$ and ending at a gray cell outside the end ring of $\mathcal{B}_i$. Consider a traversal of p, maintaining a stack containing the color loops crossed during the traversal. Crossing a color loop from interior to exterior (a sequence of dark orange, then green, purple, or blue, then light orange cells) adds the center subloop's color to the stack, and traversing from exterior to interior removes the topmost element of the stack.

We claim that the sequence of subloop colors found in the stack after traversing an end ring from interior to exterior encodes a unique sequence of display rings and thus a unique block. To see why, first consider that the color loop of every ring forms a simple closed curve. Then the Jordan curve theorem implies

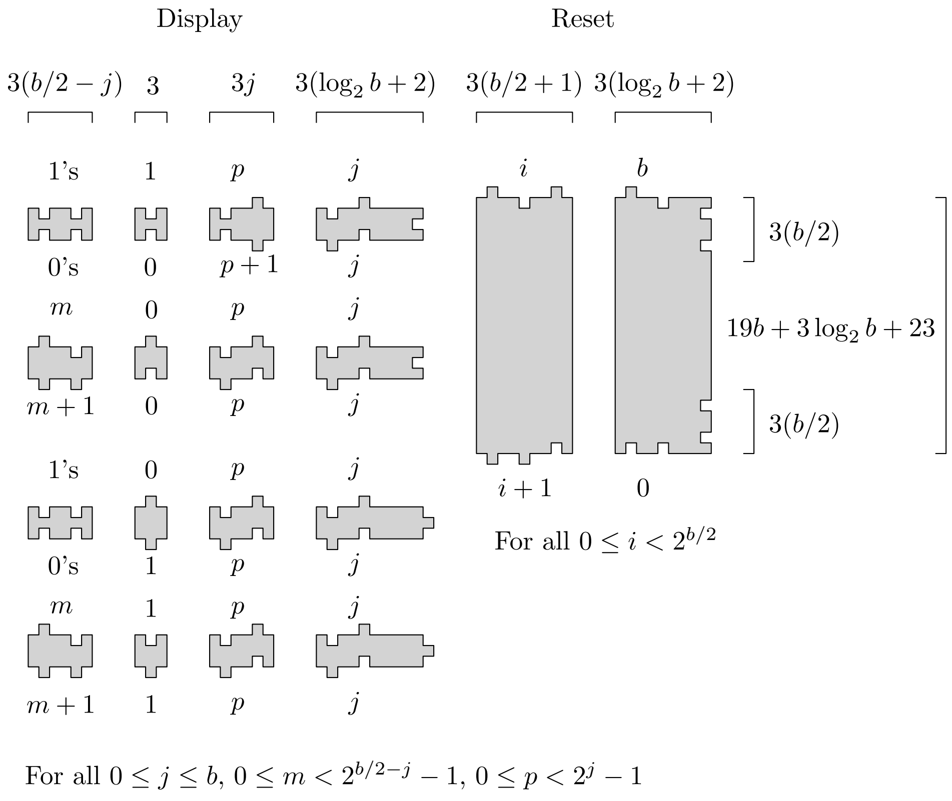


Fig. 22. The decomposition of the bars of a vertically-oriented end-to-end counter used to combine rows of blocks in a block counter.

that entering or leaving each region of gray cells between adjacent color loops requires traversing the color loop. Then by induction on the steps of p, the stack contains the set of rings *not containing* the current location on p in innermost to outermost order. So the stack state after exiting the exterior of the end ring uniquely identifies the block containing p and N is a minimal spanner for this unique block.

Since there are 2^b distinct blocks in a b-bit block counter, any PCFG that generates a counter has at least 2^b non-terminal symbols and size $\Omega(2^b)$.

Theorem 6. *The separation of PCFGs over $\tau = 1$ SASs for constant-label squares is $\Omega(n/\log^3 n)$.*

Proof. By construction, a b-bit block counter has size $\Theta(2^b b^2) = n$ and so $b = \Theta(\log n)$. By the previous two lemmas, the separation is $\Omega((n/b^2)/b) = \Omega(n/\log^3 n)$.

Unlike the previous rectangle construction, it does not immediately follow that a similar separation holds for 2-label squares. Finding a construction that

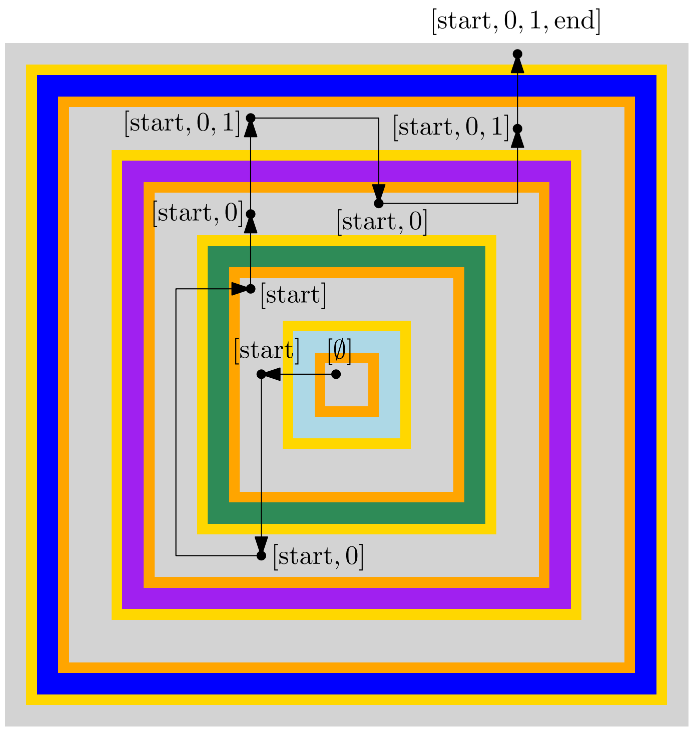


Fig. 23. A schematic of the proof that the block spanned by a minimal row spanner is unique. Maintaining a stack while traversing a path from the interior of the start ring to the exterior of the end ring uniquely determines the block spanned by any minimal block spanner containing the path.

achieves nearly-linear separation but only uses two labels remains an open problem.

6.4 Constant-glue constructions

Lemma 6 proved that any system $\mathcal{S}$ can be converted to a slightly larger system (both in system size and scale) that simulates $\mathcal{S}$. Applying this lemma to the constructions of Section 6 yields identical results for constant-glue systems:

Theorem 7. *All results in Section 6 hold for systems with* $O(1)$ *glues.*

Proof. Lemma 6 describes how to convert any SAS or SSAS $\mathcal{S} = (T, G, \tau, M)$ into a macrotile version of the system $\mathcal{S}'$ that uses a constant number of glues, has system size $O(\Sigma(T)|T| + |\mathcal{S}|)$, and scale factor $O(\log |G|)$. Additionally, the construction achieves matching labels on *all* tiles of each macrotile, including the glue assemblies. Because the labels are preserved, the polyominoes produced by each macrotile system $\mathcal{S}'$ simulating an assembly system $\mathcal{S}$ in Section 6 preserves the lower bounds for PCFGs (Lemmas 8, 10, and 12) of each construction. Moreover, the number of labels in the polyomino is constant and so $|\mathcal{S}'| = O(|T| + |\mathcal{S}|) = O(|\mathcal{S}|)$ and the system size of each construction remains the same. Finally, the scale of the macrotiles is $O(\log |G|) = O(\log |\mathcal{S}| = O(\log b)$, so n is increased by a $O(\log^2 b)$-factor, but since n was already exponential in b, it is still the case that $b = \Theta(\log n)$ and so the separation factors remain unchanged.

7 Conclusion

As the results of this work show, efficient staged assembly systems may use a number of techniques including, but not limited to, those described by local combination of subassemblies as captured by PCFGs. It remains an open problem to understand how the efficient assembly techniques of Section 5 and Section 6 relate to the general problem of optimally assembling arbitrary shapes.

Acknowledgements

We thank Benjamin Hescott and anonymous reviewers for helpful comments and feedback that greatly improved the presentation of the paper.

References

1. L. Adleman, Q. Cheng, A. Goel, and M.-D. Huang. Running time and program size for self-assembled squares. In *Proceedings of Symposium on Theory of Computing (STOC)*, 2001.
2. S. Cannon, E. D. Demaine, M. L. Demaine, S. Eisenstat, M. J. Patitz, R. T. Schweller, S. M. Summers, and A. Winslow. Two hands are better than one (up to constant factors): Self-assembly in the 2HAM vs. aTAM. In *Proceedings of International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 20 of *LIPIcs*, pages 172–184, 2013.
3. M. Charikar, E. Lehman, A. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and a. shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.
4. H.L. Chen and D. Doty. Parallelism and time in hierarchical self-assembly. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012.
5. M. Cook, Y. Fu, and R. Schweller. Temperature 1 self-assembly: determinstic assembly in 3D and probabilistic assembly in 2D. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011.
6. E. Czeizler and A. Popa. Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly. In D. Stefanovic and A. Turberfield, editors, *DNA 18*, volume 7433 of *LNCS*, pages 58–72. 2012.
7. E. D. Demaine, M. L. Demaine, S. Fekete, M. Ishaque, E. Rafalin, R. Schweller, and D. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.
8. E. D. Demaine, S. Eisenstat, M. Ishaque, and A. Winslow. One-dimensional staged self-assembly. *Natural Computing*, 2012.

9. D. Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12):78–88, 2012.
10. D. Doty, L. Kari, and B. Masson. Negative interactions in irreversible self-assembly. In Y. Sakakibara and Y. Mi, editors, *DNA 16*, volume 6518 of *LNCS*, pages 37–48. 2011.
11. D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. Intrinsic universality in self-assembly. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 5 of *LIPIcs*, pages 275–286, 2010.
12. D. Doty, J. H. Lutz, M. J. Patitz, R. T. Schweller, S. M. Summers, and D. Woods. The tile assembly model is intrinsically universal. In *Proceedings of Foundations of Computer Science (FOCS)*, pages 302–310, 2012.
13. M. Göös and P. Orponen. Synthesizing minimal tile sets for patterned dna self-assembly. In Y. Sakakibara and Y. Mi, editors, *DNA 16*, volume 6518 of *LNCS*, pages 71–82. 2011.
14. A. Jeż. Approximation of grammar-based compression via recompression. Technical report, arXiv, 2013.
15. E. Lehman. *Approximation Algorithms for Grammar-Based Data Compression*. PhD thesis, MIT, 2002.
16. M. J. Patitz. An introduction to tile-based self-assembly. In J. Durand-Lose and N. Jonoska, editors, *UCNC 2012*, volume 7445 of *LNCS*, pages 34–62. 2012.
17. M. J. Patitz, R. T. Schweller, and S. M. Summers. Exact shapes and turing universality at temperature 1 with a single negative glue. In L. Cardelli and W. Shih, editors, *DNA 17*, volume 6937 of *LNCS*, pages 175–189. 2011.
18. P. W. K. Rothemund and E. Winfree. The program-size complexity of self-assembled squares. In *Proceedings of Symposium on Theory of Computing (STOC)*, pages 459–468, 2000.
19. R. T. Schweller and M. Sherman. Fuel efficient computation in passive self-assembly. Technical report, arXiv, 2012.
20. S. Seki. Combinatorial optimization in pattern assembly. Technical report, arXiv, 2013.
21. D. Soloveichik and E. Winfree. Complexity of self-assembled shapes. In Claudio Ferretti, Giancarlo Mauri, and Claudio Zandron, editors, *DNA 11*, volume 3384 of *LNCS*, pages 344–354. 2005.
22. E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, Caltech, 1998.