Fair Sticker Languages*

Hendrik Jan Hoogeboom Nikè van Vugt

LIACS, Universiteit Leiden
Niels Bohrweg 1 2333 CA
Leiden, The Netherlands
{hoogeboo,nvvugt}@liacs.nl

July 5, 2000

Abstract

Codings of fair sticker languages are characterized as languages accepted by blind one-counter automata.

1 Introduction

Abstracting from their bio-chemical properties, single DNA strands can be seen as strings over the alphabet $\{\mathsf{a}, \mathsf{c}, \mathsf{g}, \mathsf{t}\}$ where the letters represent the nucleic acid bases adenine, cytosine, guanine, and thymine. Under favourable circumstances two strands of DNA join ('anneal') to form a double strand, provided the bases match: adenine binds with thymine, and cytosine binds with guanine.

Adleman [Adl94] used this 'complementarity principle' to propose a bio-chemical implementation of an algorithm to solve the Hamiltonian Path Problem, the question whether a given graph contains a path entering each of the nodes exactly once. In Adleman's scheme nodes are represented by short DNA strands, and edges are designed to match with the second half of their source node and the first half of their target node. We illustrate this in Figure 1 for two nodes '$\mathsf{acgggtgg}$' and '$\mathsf{atcctagt}$', connected by the edge '$\mathsf{cacctagg}$'. In this way, when the nodes and edges are put together in a solution, paths in the graph are formed by self-assembly. Then the Hamiltonian paths may be detected in the solution by a rather involved biochemical selection process.

Sticker systems are introduced as a formal language model for the self-assembly phase of Adleman's experiment [KP$^+$98]. A sticker system specifies finite sets of upper and lower 'stickers' (single-stranded molecules), and a finite set of axioms (used as a seed for the process joining upper and lower strands). The complementarity relation is modelled by a binary relation on

*To be published in Acta Informatica

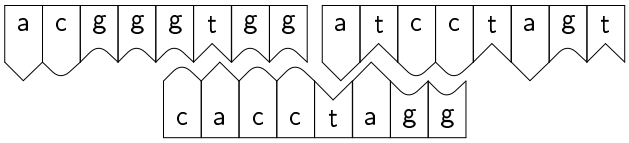

Figure 1: Connecting two nodes by an edge

the alphabet. The language generated by the system consists of all strings formed by upper stickers for which an exactly matching (i.e., complementary) sequence of lower stickers can be found.

In the sequel we use the term 'sticker system' for the simple regular sticker systems from [KP$^+$98].

Without restrictions, sticker systems generate only regular languages (but not all of them). Additional restrictions can be imposed on the matching pairs of strands to obtain more powerful languages. For example, the matching is *fair* if the number of upper stickers used is the same as the number of lower stickers used. In [KP$^+$98] it is demonstrated that the family of fair sticker languages contains non-regular languages but not all regular languages, while the family of languages generated by context-free matrix grammars with arbitrary rules is given as an upper bound. In connection with this rather large upper bound the following open problem is formulated: "*is the family [of fair sticker languages] included in the family of context-free languages (or even in the family of linear languages)?*".

We answer this question by giving a fair sticker language that is non-linear (Example 2), while demonstrating that the fair sticker languages are strictly included in another subfamily of the context-free languages, the blind one-counter languages (Theorem 6 and Lemma 7).

We conclude by showing as our main result that the connection between these two families is quite strong: blind one-counter languages can be characterized as the codings of fair sticker languages (Theorem 12).

2 Preliminaries

We adhere to standard formal language theory notations and terminology as in, e.g., [RS97].

The set of integers is denoted by $\mathbb{Z}$. The empty string is written as λ. For a language L and a string w we define the *left-quotient* of L by w as the set $\{ x \mid wx \in L \}$. We use $\mathsf{REG}, \mathsf{LIN}, \mathsf{CF}$ to denote the families of regular, linear, and context-free languages. A letter to letter morphism is called a *coding*. For a language family $\mathcal{F}$, we use $\mathsf{COD}(\mathcal{F})$ to denote the family of codings of languages in $\mathcal{F}$.

Double stranded DNA molecules can be written as a pair of 'matching' strings over the alphabet $\{\mathsf{a}, \mathsf{c}, \mathsf{g}, \mathsf{t}\}$ of bases. Alternatively, for the matching

base pairs we may use the symbols $\binom{\mathsf{a}}{\mathsf{t}}$, $\binom{\mathsf{t}}{\mathsf{a}}$, $\binom{\mathsf{c}}{\mathsf{g}}$, $\binom{\mathsf{g}}{\mathsf{c}}$. We appreciate both approaches, and will not distinguish between a (two dimensional) pair of matching strands like $(\mathsf{agac}, \mathsf{tctg})$ and a (one dimensional) string of paired bases $\binom{\mathsf{a}}{\mathsf{t}}\binom{\mathsf{g}}{\mathsf{c}}\binom{\mathsf{a}}{\mathsf{t}}\binom{\mathsf{c}}{\mathsf{g}}$.

For our purposes we will consider an alphabet Σ of 'abstract' DNA bases, and a relation $\rho \subseteq \Sigma \times \Sigma$ representing the complementarity relation. We extend ρ to a subset of $\Sigma^* \times \Sigma^*$ by demanding that two strings are complementary if they are of equal length and their letters are one by one complementary.

The alphabet Σ_ρ, representing matching pairs, consists of all symbols $\binom{a}{b}$ where $(a, b) \in \rho$ and $a, b \in \Sigma$. As explained above, we identify Σ_ρ^* with the subset ρ of $\Sigma^* \times \Sigma^*$: $\binom{a_1}{b_1} \ldots \binom{a_n}{b_n}$ represents the same double stranded molecule as $(a_1 \ldots a_n, b_1 \ldots b_n)$ with $(a_i, b_i) \in \rho$ and $a_i, b_i \in \Sigma$.

Our definition of sticker systems is equivalent to the definition given in [KP+98], but stated directly in terms of strings, avoiding a lengthy definition of a 'sticker operation'.

Definition 1 A *sticker system* is a 5-tuple $\gamma = (\Sigma, \rho, D_u, D_\ell, A)$ where Σ is an alphabet, $\rho \subseteq \Sigma \times \Sigma$ is the complementarity relation, $D_u, D_\ell \subseteq \Sigma^+$ are finite sets of *upper* and *lower stickers*, and $A \subseteq \Sigma^* \times \Sigma^*$ is a finite set of *axioms*. □

The *molecular (sticker) language* generated by γ is defined as $ML(\gamma) = \{(x, y) \in \rho \mid x = x_0 x', y = y_0 y' \text{ with } (x_0, y_0) \in A, x' \in D_u^* \text{ and } y' \in D_\ell^*\}$. The *fair* molecular (sticker) language generated by γ is defined as $ML_f(\gamma) = \{(x, y) \in \rho \mid x = x_0 x', y = y_0 y' \text{ with } (x_0, y_0) \in A, x' \in D_u^n \text{ and } y' \in D_\ell^n \text{ for some } n \geq 0\}$.

Furthermore, the *(sticker) language* generated by γ is the projection onto the first (upper) component of the molecular language, $L(\gamma) = \{x \in \Sigma^* \mid (x, y) \in ML(\gamma) \text{ for some } y \in \Sigma^* \}$, and similarly for $L_f(\gamma)$, the *fair* (sticker) language generated by γ.

The family of all sticker languages is denoted SL, while the family of fair sticker languages is denoted SL_f.

Example 2 Consider the following sticker system, a slight extension of the one given in the proof of Theorem 3 in [KP+98]: $\gamma = (\Sigma, \rho, D_u, D_\ell, A)$ with

$$\begin{aligned} &\Sigma = \{a, b, c\},\ \rho = \{(a, a), (b, b), (b, c)\}, \\ &D_u = \{aa, b\},\ D_\ell = \{a, bc\}, \\ &A = \{(\lambda, \lambda)\}. \end{aligned}$$

Now $ML(\gamma) = \{ \binom{a}{a}\binom{a}{a}, \binom{b}{b}\binom{b}{c} \}^*$. Consequently $L(\gamma) = \{aa, bb\}^*$, while $L_f(\gamma) = \{x \in L(\gamma) \mid \#_a(x) = \#_b(x)\}$. □

Obviously, $L(\gamma)$ can be obtained from $ML(\gamma)$ by applying a coding. However, we may not reverse this: in general $ML(\gamma)$ is not the image of $L(\gamma)$ under an inverse coding, as clear from Example 2.

A rather unexpected but useful normal form for sticker systems is proved in the following theorem: without changing the language, we can always replace the complementarity relation ρ by the identity id on the alphabet Σ. Note that, of course, the molecular language *does* change if ρ was not already equal to id.

Theorem 3 *For every sticker system $\gamma = (\Sigma, \rho, D_u, D_\ell, A)$ a sticker system $\gamma' = (\Sigma, id, D_u, D'_\ell, A')$ can be constructed with $L(\gamma') = L(\gamma)$ and $L_f(\gamma') = L_f(\gamma)$.*

Proof. Let $D'_\ell = \{w \in \Sigma^+ \mid (w, v) \in \rho \text{ for some } v \in D_\ell\}$, and let $A' = \{(x_0, z_0) \mid (x_0, y_0) \in A \text{ for some } y_0, \text{ and } (z_0, y_0) \in \rho\}$.

Now assume that $x \in L(\gamma)$, i.e., there is a y such that $(x, y) \in \rho$, $x = x_0 x_1 \dots x_n$ and $y = y_0 y_1 \dots y_m$, where $(x_0, y_0) \in A$, $x_i \in D_u$ for $1 \leq i \leq n$ and $n \geq 0$, and $y_j \in D_\ell$ for $1 \leq j \leq m$ and $m \geq 0$. Then x can also be written as $x = z_0 z_1 \dots z_m$, where $(z_k, y_k) \in \rho$ for $0 \leq k \leq m$. According to the definition of A' then $(x_0, z_0) \in A'$, and from the definition of D'_ℓ it follows that $z_j \in D'_\ell$ for $1 \leq j \leq m$. Therefore $(x_0 x_1 \dots x_n, z_0 z_1 \dots z_m) = (x, x) \in ML(\gamma')$, hence $x \in L(\gamma')$.

To prove that $L(\gamma') \subseteq L(\gamma)$, the above can be read backwards.

Obviously, the number of stickers used is not changed, hence $L_f(\gamma') = L_f(\gamma)$ as well. □

The following result (a combination of Theorem 4.1, Theorem 4.7 and Corollary 4.7 from [PRS98]) and its proof indicate that the sticker languages we consider are not very suitable to implement states (of a finite state automaton) except if one makes them explicit in the symbols. A coding makes it possible to abstract from this.

Proposition 4 $\mathsf{SL} \subset \mathsf{REG} = \mathsf{COD}(\mathsf{SL})$.

3 Blind one-counter automata

A *blind one-counter automaton* (BCA) is a finite state device equipped with an external memory (the 'counter') that contains an integer value which may be incremented and decremented by the automaton.

Definition 5 A BCA is a 5-tuple $\mathcal{B} = (Q, \Sigma, \delta, q_{in}, F)$, where Q is a finite set of states, Σ is the input alphabet, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ is the set of final states , and $\delta \subseteq Q \times \Sigma \times \{-1, 0, 1\} \times Q$ is a finite set of instructions. □

An instantaneous description of $\mathcal{B}$ is an element of $Q \times \Sigma^* \times \mathbb{Z}$. For two instantaneous descriptions (p, ax, i) and (q, x, j), we write $(p, ax, i) \vdash (q, x, j)$ if $(p, a, \varepsilon, q) \in \delta$ and $j = i + \varepsilon$. By $\vdash^*$ we denote the reflexive and transitive closure of $\vdash$.

The *language accepted by* $\mathcal{B}$ consists of all strings for which the automaton in a computation on this string ends in a final state *and* at the same time has counter value zero. It is defined as $L(\mathcal{B}) = \{x \in \Sigma^* \mid (q_{in}, x, 0) \vdash^* (f, \lambda, 0) \text{ for some } f \in F\}$. The family of all languages accepted by blind one-counter automata (BCA-languages) is called 1BCA.

The datatype above is called *blind* because the automaton cannot test its counter value during the computation, i.e., it may not check whether its counter value is zero and act according to the outcome of this test.

The blind one-counter automaton can be 'implemented' on a more commonly known device: the stack of a pushdown automaton may act as a counter. Consequently 1BCA $\subseteq$ CF. Since the context-free language $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w) \text{ and } \#_a(x) \geq \#_b(x) \text{ for every prefix } x \text{ of } w\}$ is not in 1BCA (see [Gre78, Theorem 3]), we even have 1BCA $\subset$ CF.

In contrast with the definition of BCA given in [Gre78], we do not allow λ-instructions, i.e., instructions of the form $(p, \lambda, \varepsilon, q)$. However, these two definitions are equivalent, which can be explained as follows. The possible successful instruction sequences of the blind one-counter datatype are naturally modelled by the 'two-sided Dyck language' $D_1^* = \{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$, where a and b represent addition of $+1$ and -1, respectively. Now, using standard AFA theory [GG69], the BCA can be seen as a finite state device mapping input strings to strings over $\{a, b\}$ according to the instructions taken during the computation. The input is accepted precisely when the output belongs to D_1^*. Hence it can be shown that the family 1BCA is equal to the smallest language family that contains D_1^* and is closed under λ-free homomorphism, inverse homomorphism and intersection with regular languages. In other words, 1BCA equals $\mathcal{C}^f(D_1^*)$, the faithful rational cone generated by D_1^*. Similarly, the family of languages generated by BCA's that can have λ-instructions is equal to $\mathcal{C}(D_1^*)$, the rational cone generated by D_1^* (the λ-free homomorphism above is replaced by an arbitrary homomorphism). In [Lat79, Proposition II.11] it is proved (as a special case of a more general result) that $\mathcal{C}^f(D_1^*) = \mathcal{C}(D_1^*)$. Hence the BCA's with λ-instructions are equivalent to the BCA's without λ-instructions.

From the discussion above we conclude that 1BCA is a principal rational cone, and in particular that 1BCA is closed under codings, left-quotient with strings, and union.

Apart from this connection with AFA/AFL theory, blind one-counter automata were also studied as 'integer weighted finite automata' in [HH99] and as 'additive regular valence grammars (over $\mathbb{Z}$)' in [Pău80] (see also [FS97]). In these devices the instructions (productions) are assigned an integer value, and one considers only computations (derivations) for which

these values add to 0.

4 Fair sticker languages are BCA-languages

We answer the question left open in [KP+98, p. 419]: is the family of fair sticker languages included in the family of context-free languages (or even in the family of linear languages)? To start, observe that the language $L_f(\gamma) = \{x \in \{aa, bb\}^* \mid \#_a(x) = \#_b(x)\}$ from Example 2 is context-free, but not linear.

The non-linearity of $L_f(\gamma)$ can be proved using the pumping lemma for linear languages [HU79, Exercise 6.11], which says that if K is linear, then there is a constant n such that every $z \in K$ with $|z| > n$ can be written as $z = uvwxy$ with $|uvxy| \leq n$, $|vx| \geq 1$ and $uv^iwx^iy \in K$ for all $i \geq 0$. In the case of $L_f(\gamma)$, it is clear that there are no such u, v, w, x, y for $z = a^{2n}b^{4n}a^{2n} \in L_f(\gamma)$.

We will now give a first answer to the question posed in [KP+98], by proving that every fair sticker language is a BCA-language, hence context-free.

Theorem 6 $\mathsf{SL}_f \subseteq \mathsf{1BCA}$.

Proof. Let $\gamma = (\Sigma, \rho, D_u, D_\ell, A)$. Because of Theorem 3 we may assume that $\rho = id$. For each $(x_0, y_0) \in A$, construct two BCA's: $\mathcal{B}_{x_0}$ and $\mathcal{B}_{y_0}$, as follows. We describe the construction of $\mathcal{B}_{x_0} = (Q, \Sigma, \delta, q_{in}, \{f\})$ in detail, $\mathcal{B}_{y_0}$ can be made in an analogous way.

If $x_0 = \lambda$, then $q_{in} = f$. If $x_0 \neq \lambda$, then $\mathcal{B}_{x_0}$ has a path labelled by x_0 from its initial to its final state. In both cases the counter is not changed, since the axioms do not have to be counted. Moreover, for each $w \in D_u$, let $\mathcal{B}_{x_0}$ have a (new) path labelled with w from its final to its final state and add 1 to the counter at one moment somewhere along this path. Note that $L(\mathcal{B}_{x_0}) = \{x_0\}$, which does not seem very useful yet!

Now we construct from each pair of BCA's $\mathcal{B}_{x_0}$ and $\mathcal{B}_{y_0}$, where $(x_0, y_0) \in A$, a BCA $\mathcal{B}_{x_0,y_0}$ for which $L(\mathcal{B}_{x_0,y_0}) = L_f(\gamma_{x_0,y_0})$, where $\gamma_{x_0,y_0} = (\Sigma, id, D_u, D_\ell, \{(x_0, y_0)\})$, as follows: for each pair of instructions (p, a, ε, q) in $\mathcal{B}_{x_0}$ and (r, a, ε', s) in $\mathcal{B}_{y_0}$, the BCA $\mathcal{B}_{x_0,y_0}$ contains the instruction $(\langle p, r\rangle, a, \varepsilon - \varepsilon', \langle q, s\rangle)$.

Finally, it is clear that $L_f(\gamma) = \bigcup_{(x_0,y_0)\in A} L(\mathcal{B}_{x_0,y_0})$ is in 1BCA, since 1BCA is closed under union and A is finite. □

Omitting the counter from the previous proof, one constructs a finite state automaton for $L(\gamma) = \bigcup_{(x_0,y_0)\in A} x_0 \cdot D_u^* \cap y_0 \cdot D_\ell^*$. This elementary observation shows that $\mathsf{SL} \subseteq \mathsf{REG}$.

The inclusion $\mathsf{SL}_f \subseteq \mathsf{1BCA}$ is strict because ba^*b is not a fair sticker language, although $ba^*b \in \mathsf{REG} \subseteq \mathsf{1BCA}$.

Lemma 7 $ba^*b \notin \mathsf{SL}_f$.

Proof. We reconsider the proof of $ba^+b \notin \mathsf{SL}$, cf. [PR98, Theorem 10]. Assume ba^*b is the fair language of a sticker system $\gamma = (\{a,b\}, \rho, D_u, D_\ell, A)$. According to Theorem 3 we may assume that $\rho = id$. Let $D_u \cap a^+ = \{x_1, \ldots, x_m\}$ and $D_\ell \cap a^+ = \{y_1, \ldots, y_n\}$ be the sets of stickers consisting of a's only. Every string ba^ib that is longer than the axioms can be decomposed as $\alpha_u x_1^{j_1} \ldots x_m^{j_m} \beta_u = \alpha_\ell y_1^{k_1} \ldots y_n^{k_n} \beta_\ell$, with α_u the upper part of an axiom (or a string from D_u starting with b), and $\beta_u \in D_u$ ending in b, and similarly for α_ℓ, β_ℓ. The vector $\nu_i = (j_1, \ldots, j_m, k_1, \ldots, k_n)$ assigns to ba^ib the number of stickers containing only a's occurring in a possible decomposition of the upper and the lower strand.

Because we have only a finite number of choices, an infinite number of ba^ib have the same strings $\alpha_u, \alpha_\ell, \beta_u, \beta_\ell$ in their decompositions. According to Dickson's Lemma [Dic13, Lemma B] we can find ba^ib and $ba^{i'}b$ $(i' > i)$ in this infinite sequence such that $\nu_{i'} \geq \nu_i$ (componentwise). Now the vector $\nu_{i'} - \nu_i$ defines a 'fair decomposition' of $a^{i'-i}$, which shows that $ba^iba^{i'-i} \in L_f(\gamma)$, contradicting $L_f(\gamma) = ba^*b$. □

In the next section we make our answer more precise, in the sense that we show that 1BCA is a rather close upper bound for SL_f: every BCA-language is a coding of a fair sticker language.

5 BCA-languages are codings of fair sticker languages

In the case of arbitrary, i.e., not necessarily fair, sticker languages the simulation of sticker systems by finite automata can be reversed provided that one can use a coding (Proposition 4). In this section we demonstrate that Proposition 4 can be extended to fair sticker languages and BCA-languages: every language in 1BCA is the coding of a fair sticker language (Theorem 12). First we illustrate this in Example 8. Then we show that, for a particular kind of BCA called *sticky*, this example can be generalized (Lemma 10). Finally, we explain how these sticky BCA's can be used to construct a coding of a fair sticker language for every BCA-language.

Example 8 Consider the BCA $\mathcal{A}$ with states $Q = \{a_1, a_3, b_0, b_2, c_0, c_2\}$ with b_0 as initial state, final state set $\{b_0, c_0\}$ and instructions

$$\begin{array}{llll} (b_0, a, 0, a_1), & (a_1, b, -1, b_2), & (b_2, a, 0, a_3), & (a_3, b, 0, b_0), \\ (c_0, a, 0, a_1), & (a_1, c, 0, c_2), & (c_2, a, 0, a_3), & (a_3, c, +1, c_0). \end{array}$$

The automaton accepts the language $\{\ w \in \{ab, ac\}^* \mid \#_b(w) = \#_c(w)\ \}$. This can be verified by considering the four-letter segments *abab*, *abac*, *acab*, and *acac*. While the automaton makes a cycle on these segments (starting and ending in $\{b_0, c_0\}$), it changes its counter by -1, 0, 0, and $+1$, respectively.

First, we forget about the counter, and we have a look at the finite state behaviour of $\mathcal{A}$, $(a(b+c)a(b+c))^*$. A computation can be simulated by a sticker system with overlapping stickers, cf. [KP$^+$98, Lemma 5], illustrated as follows, with brackets to delimit the stickers and the axiom:

$$\begin{array}{l} [\rangle\langle a_1\, b_2\ \ a_3\, b_0\rangle\langle a_1\, b_2\ \ a_3\, c_0\rangle\langle a_1\, c_2\ \ a_3\, b_0\rangle\langle a_1\, c_2\ \ a_3\, c_0\rangle \\ [\ \ a_1\, b_2\rangle\langle a_3\, b_0\ \ a_1\, b_2\rangle\langle a_3\, c_0\ \ a_1\, c_2\rangle\langle a_3\, b_0\ \ a_1\, c_2\rangle\langle a_3 c_0\rangle \end{array}$$

Second, we can include the contents of the counter by representing it as the difference between the number of upper and lower stickers in the computation of the sticker system. For each increment instruction we detach the last component of an upper sticker, and similarly for decrement instructions and lower stickers.

$$\begin{array}{l} \qquad\qquad\qquad\qquad\qquad\qquad\quad +1 \qquad\qquad\qquad\qquad\qquad\quad +1 \\ [\rangle\langle a_1\ \ b_2\ \ a_3\, b_0\rangle\langle a_1\ \ b_2\ \ a_3\rangle\langle c_0\rangle\langle a_1\, c_2\ \ a_3\, b_0\rangle\langle a_1\, c_2\, a_3\rangle\langle c_0\rangle \\ [\ \ a_1\rangle\langle b_2\rangle\langle a_3\, b_0\, a_1\rangle\langle b_2\rangle\langle a_3\ \ c_0\ \ a_1\, c_2\rangle\langle a_3\, b_0\ \ a_1\, c_2\rangle\langle a_3\ \ c_0\rangle \\ \qquad\quad -1 \qquad\qquad\quad -1 \end{array}$$

Let $\gamma = (Q, id, D_u, D_\ell, A)$ be the sticker system specified by

$$\begin{array}{rcl} A & = & \{\ ([\lambda\rangle, [\lambda\rangle),\ ([\lambda\rangle, [a_1\rangle),\ ([\lambda\rangle, [a_1c_2\rangle)\ \} \\ D_u & = & \{\langle a_1b_2a_3\rangle, \langle a_1b_2a_3b_0\rangle, \langle a_1c_2a_3\rangle, \langle a_1c_2a_3b_0\rangle, \langle c_0\rangle\} \\ D_\ell & = & \{\langle a_3b_0a_1\rangle, \langle a_3b_0a_1c_2\rangle, \langle a_3c_0a_1\rangle, \langle a_3c_0a_1c_2\rangle, \langle a_3b_0\rangle, \langle a_3c_0\rangle, \langle b_2\rangle\}. \end{array}$$

Then $L(\mathcal{A})$ is obtained by applying to $L_f(\gamma)$ the coding $h : Q \to \{a, b, c\}$ that maps a_1, a_3 to a, b_0, b_2 to b, and c_0, c_2 to c. □

A crucial property of the BCA from the above example is formalized in the following notion.

Definition 9 Let $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, F)$ be a BCA. It is called *sticky* if there is a partition of its state set $Q = \bigcup_{i=0}^{3} Q_i$ such that δ is a subset of

$$\begin{array}{lll} (Q_0 \times \Sigma \times \{0\} \times Q_1) & \cup & (Q_1 \times \Sigma \times \{-1, 0\} \times Q_2) \quad \cup \\ (Q_2 \times \Sigma \times \{0\} \times Q_3) & \cup & (Q_3 \times \Sigma \times \{0, +1\} \times Q_0) \end{array}$$

and such that $q_{in} \in Q_0$ and $F \subseteq Q_0$. □

The BCA $\mathcal{A}$ from Example 8 is sticky.

A sticky BCA changes its counter in a very restrictive way: in each segment of four instructions the automaton may increment and decrement its counter only once, and only at specific positions. Note that the language accepted by a sticky BCA always consists of strings with lengths that are multiples of four.

We generalize the construction from Example 8.

Lemma 10 *Let $\mathcal{A}$ be a sticky* BCA. *Then there exist a sticker system γ and a coding h such that $L(\mathcal{A}) = h(L_f(\gamma))$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, F)$ be a sticky BCA. We write the state set as a disjoint union $Q = \bigcup_{i=0}^{3} Q_i$ as in the definition for sticky BCA.

Let $h : Q \to \Sigma$ be a coding such that each instruction is of the form $(p, h(q), \varepsilon, q)$, i.e., all instructions ending in a given state read the same letter. This can easily be achieved by splitting states into several copies – one for each letter from the alphabet, each of which has the same outgoing instructions – and re-routing the instructions into the appropriate copy. In the same vein we assume that there exists a partition $Q_2 = Q_2^0 \cup Q_2^-$, such that each instruction (p, a, ε, q) entering Q_2^0 (Q_2^-) has $\varepsilon = 0$ ($\varepsilon = -1$, respectively). Similarly we assume $Q_0 = Q_0^0 \cup Q_0^+$.

Construction. A sticker system $\gamma = (Q, id, D_u, D_\ell, A)$ is constructed as follows. We keep the intuitive bracket notation from Example 8.

upper stickers. For every pair of consecutive instructions $(p_1, a_2, \varepsilon_2, p_2)$, $(p_2, a_3, 0, p_3)$ with $p_1 \in Q_1$, D_u contains the stickers $\langle p_1 p_2 p_3 \rangle$ and, for every $p_0 \in Q_0^0$, $\langle p_1 p_2 p_3 p_0 \rangle$. For each $p_+ \in Q_0^+$, D_u contains the sticker $\langle p_+ \rangle$.

lower stickers. For every pair of consecutive instructions $(p_3, a_0, \varepsilon_0, p_0)$, $(p_0, a_1, 0, p_1)$ with $p_3 \in Q_3$, D_ℓ contains the stickers $\langle p_3 p_0 p_1 \rangle$ and, for every $p_2 \in Q_2^0$, $\langle p_3 p_0 p_1 p_2 \rangle$. For each $p_- \in Q_2^-$, D_ℓ contains the sticker $\langle p_- \rangle$. For every instruction $(p_3, a_0, \varepsilon_0, p_0)$ with $p_3 \in Q_3$, $p_0 \in F$, D_ℓ contains the sticker $\langle p_3 p_0 \rangle$.

axioms. For every instruction $(q_{in}, a_1, 0, p_1)$ with $p_1 \in Q_1$, A contains the pairs $([\lambda\rangle, [p_1\rangle)$ and, for every $p_2 \in Q_2^0$, $([\lambda\rangle, [p_1 p_2\rangle)$. If $q_{in} \in F$, i.e., $\lambda \in L(\mathcal{A})$, then $([\lambda\rangle, [\lambda\rangle)$ is added to A.

Correctness. Observe that $\lambda \in L(\mathcal{A})$ iff $\lambda \in L_f(\gamma)$ iff $\lambda \in h(L_f(\gamma))$.
Now, let $\pi = p_1 p_2 p_3 \dots p_n \in Q^+$ be an element of $L_f(\gamma)$, for some $n \geq 1$.

First, we reconstruct a computation of $\mathcal{A}$ by following the computation of π in γ.

Since there is no non-empty decomposition starting with $(\lambda, \lambda) \in A$ – all stickers in D_u start with symbols from $Q_1 \cup Q_0^+$, whereas all stickers from D_ℓ start with symbols from $Q_3 \cup Q_2^-$ – we know that the computation of π in γ started either with $(\lambda, p_1) \in A$ or with $(\lambda, p_1 p_2) \in A$, where $p_1 \in Q_1$. According to the construction of A, δ contains an instruction $(q_{in}, a_1, 0, p_1)$.

We continue by observing that each upper sticker of length 3 or 4 starts at position $4i + 1$, and that each lower sticker of length 2, 3 or 4 starts at position $4i + 3$, for some $i \geq 1$. It is easy to see that this follows from the

only possible computation of π, here illustrated for $n = 8$:

$$\begin{array}{l} [\rangle\langle\ p_1 \to p_2 \to p_3 \cdots p_4\ \rangle\langle\ p_5 \to p_6 \to p_7 \cdots p_8\rangle \\ [\quad p_1 \cdots p_2\ \rangle\langle\ p_3 \to p_4 \to p_5 \cdots p_6\ \rangle\langle\ p_7 \to p_8\rangle \end{array}$$

Here the arrows indicate parts of a sticker that represent instructions from δ, while the dotted lines do not necessarily correspond to an instruction from δ and, at the same time, indicate that the next symbol may be detached to form a sticker of length 1. Moreover, observing D_u we find instructions $(p_{4i+1}, a_{4i+2}, \varepsilon_{4i+2}, p_{4i+2})$ and $(p_{4i+2}, a_{4i+3}, 0, p_{4i+3})$, while D_ℓ gives rise to instructions $(p_{4i+3}, a_{4i+4}, \varepsilon_{4i+4}, p_{4i+4})$ and $(p_{4i+4}, a_{4i+5}, 0, p_{4i+5})$.

Since p_n is the last symbol of stickers from both D_u and D_ℓ, we know that $p_n \in F \subseteq Q_0$, and there exists an instruction $(p_{n-1}, a_n, \varepsilon_n, p_n)$ in δ. Note that n is a multiple of four, and we write $n = 4k$.

Second, we address the matter of fairness. To compute the contents of the counter we study the even positions of π. Observe that $\varepsilon_{4i+4} = +1$ iff $p_{4i+4} \in Q_0^+$, which implies that the sticker $\langle p_{4i+4}\rangle$ is used in the upper part of the solution. Otherwise, if $\varepsilon_{4i+4} = 0$, then p_{4i+4} is the fourth element of the sticker $\langle p_{4i+1}p_{4i+2}p_{4i+3}p_{4i+4}\rangle$. Thus, the number of upper stickers equals $k+\sum_{i=0}^{k-1} \varepsilon_{4i+4}$. Similarly, the number of lower stickers equals $k-\sum_{i=0}^{k-1} \varepsilon_{4i+2}$. Consequently, fairness of the sticker solution is equivalent to counter value zero and acceptance by the BCA.

The above shows that $h(L_f(\gamma)) \subseteq L(\mathcal{A})$. For the converse inclusion $L(\mathcal{A}) \subseteq h(L_f(\gamma))$ a similar reasoning can be given. □

Sticky BCA's form a normal form for BCA's accepting languages consisting of strings with lengths that are multiples of four. The idea behind this is the following.

In every four steps, $\mathcal{A}$ changes the contents of its counter by at most ± 4. The new BCA $\mathcal{B}$ however, may change its counter by at most ± 1 in the corresponding four steps. To make up for this, we change the interpretation of the counter value of $\mathcal{B}$: each unit on the counter of $\mathcal{B}$ represents 4 units on the counter of $\mathcal{A}$, a construction known at least since [FMR68]. Now, $\mathcal{B}$ simulates the computation of $\mathcal{A}$. Each change made to the counter of $\mathcal{A}$ is recorded in the finite state memory of $\mathcal{B}$. Only when allowed (at the specific points in the four step cycle), $\mathcal{B}$ moves any excess of ± 4 units of $\mathcal{A}$'s counter as one unit to (or from) its own counter.

Lemma 11 *For each* BCA *that accepts only strings with lengths a multiple of four, there exists an equivalent sticky* BCA.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_{in}, F)$ be a BCA. We construct a sticky BCA $\mathcal{B}$ such that $L(\mathcal{A}) = L(\mathcal{B})$.

Let $I = \{0, 1, 2, 3\}$. The state set Q' of $\mathcal{B}$ equals

$$Q \times I \times \{-4, -3, \ldots, 2, 3\},$$

the elements of which are denoted as $p.i.m$, rather than as (p, i, m). Here $p \in Q$ represents the state of $\mathcal{A}$, $i \in I$ keeps track of the four step cycle, and $-4 \leq m \leq 3$ is the remainder value of $\mathcal{A}$'s counter not yet stored in the counter of $\mathcal{B}$. (Hence, if c is the value of $\mathcal{A}$'s counter and c' the value of $\mathcal{B}$'s counter, then the equality $c = 4c' + m$ should hold for each pair of corresponding instantaneous descriptions of $\mathcal{A}$ and $\mathcal{B}$.) The initial state of $\mathcal{B}$ equals $q_{in}.0.0$, its final state set equals $F \times \{0\} \times \{0\}$.

Let (p, a, ε, q) be an instruction of $\mathcal{A}$. Then $\mathcal{B}$ has the instructions

$$\begin{array}{lll}
(p.0.m,\ a,\ 0,\ q.1.m{+}\varepsilon\ ) & & \\
(p.1.m,\ a,\ -1,\ q.2.m{+}\varepsilon{+}4\ ) & \text{if} & m+\varepsilon < -1 \\
(p.1.m,\ a,\ 0,\ q.2.m{+}\varepsilon\ ) & \text{if} & m+\varepsilon \geq -1 \\
(p.2.m,\ a,\ 0,\ q.3.m{+}\varepsilon\ ) & & \\
(p.3.m,\ a,\ +1,\ q.0.m{+}\varepsilon{-}4\ ) & \text{if} & m+\varepsilon \geq 1 \\
(p.3.m,\ a,\ 0,\ q.0.m{+}\varepsilon\ ) & \text{if} & m+\varepsilon < 1
\end{array}$$

We chose to check the relation between $m + \varepsilon$ and ± 1 rather than between $m + \varepsilon$ and ± 4, although the latter seems more logical. The reason for this is that we need to prevent the occurrence of the situation where $i = 0$ and $c = 4c' + m = 0$ while $c' \neq 0$ and $m \neq 0$ (which can occur only when m is a multiple of 4), i.e., $\mathcal{B}$ does not accept while it should. Because of this choice, the reachable configurations of $\mathcal{B}$ satisfy the following restrictions, for $p.i.m \in Q'$:

$$\begin{array}{lll}
\text{if} \quad i = 0 & \text{then } m \in & \{-3, -2, -1, 0\} \\
\quad\ \ 1 & & \{-4, -3, -2, -1, 0, 1\} \\
\quad\ \ 2 & & \{-1, 0, 1, 2\} \\
\quad\ \ 3 & & \{-2, -1, 0, 1, 2, 3\}
\end{array}$$

It is easy to see that $\mathcal{B}$ is sticky, as it adheres to the four step cycle from Definition 9.

Moreover, note that our construction introduces for each instruction (p, a, ε, q) of $\mathcal{A}$ exactly one instruction $(p.i.m, a, \varepsilon', q.i'.m')$ for each pair i, m. This makes it straightforward to show that a computation $(q_{in}, xy, 0) \vdash^j (q, y, c)$ of $\mathcal{A}$ corresponds with a computation $(q_{in}.0.0, xy, 0) \vdash^j (q.i.m, y, c')$ of $\mathcal{B}$ satisfying $c = 4c' + m$, and $i = j \bmod 4$.

To show that $L(\mathcal{B}) \subseteq L(\mathcal{A})$, observe that if $\mathcal{B}$ reaches a final state $q.0.0$ with counter value zero, then $\mathcal{A}$ (using the corresponding computation) reaches final state q, also with counter value zero.

Conversely, assume that $\mathcal{A}$ reaches a final state $q \in F$ with counter value zero. Now the corresponding computation of $\mathcal{B}$ reaches some state $q.i.m$ and counter value c' satisfying the invariant $m{+}4c' = 0$. By the length restriction of strings accepted by $\mathcal{A}$ we know that $i = 0$. Hence, taking into account the reachable states of $\mathcal{B}$, we have $m \in \{-3, \ldots, 0\}$. Thus $m + 4c' = 0$ implies that $m = 0$ and thus $c' = 0$, corresponding to acceptance with counter value zero in final state $q.0.0$.

A more formal inductive proof that $L(\mathcal{A}) = L(\mathcal{B})$ is left to the reader. □

Finally we arrive at our main result, the equivalence of blind one-counter languages and codings of fair sticker languages. Note the similarity with the situation for (arbitrary) sticker languages (Proposition 4).

Theorem 12 $\mathsf{SL}_f \subset \mathsf{1BCA} = \mathsf{COD}(\mathsf{SL}_f)$.

Proof. By Theorem 6, $\mathsf{SL}_f \subseteq \mathsf{1BCA}$. The inclusion is strict by Lemma 7. As 1BCA is closed under codings, the inclusion $\mathsf{COD}(\mathsf{SL}_f) \subseteq \mathsf{1BCA}$ follows. We proceed by proving the converse inclusion.

Let $L \in \mathsf{1BCA}$. For every string w, we define $L_w = \{\ x \mid wx \in L, |x| = 0 \bmod 4\ \}$. By the closure properties we have established for 1BCA, L_w is also in 1BCA, and, by Lemma 11, it is accepted by a sticky BCA. Consequently, it is the coding of a fair sticker language (Lemma 10).

Note that $L = \bigcup_{|w|\leq 3} w \cdot L_w$. A sticker system for the language $w \cdot L_w$ is obtained from the one for L_w by replacing each axiom (x, y) by (wx, wy) and extending the used coding with the identity on the alphabet of L. Assuming the sticker systems representing the $w \cdot L_w$ have disjoint alphabets (by renaming), we build a sticker system for L by taking their (finite) union. □

Our characterization shows that $\mathsf{COD}(\mathsf{SL}_f)$ is a more 'robust' family than SL_f itself, comparable to the situation for $\mathsf{COD}(\mathsf{SL})$ and SL. In particular, we can conclude that $\mathsf{COD}(\mathsf{SL}_f)$ enjoys the many closure properties of a principal rational cone (arbitrary morphisms, inverse morphisms, intersection with regular languages, and union). Some of these properties seem to require rather involved proofs, should we want to show them by direct construction.

Acknowledgements. We are indebted to Michel Latteux, Holger Petersen and Matthias Jantzen for their helpful suggestions concerning the equivalence of blind one-counter automata with and without λ-instructions. We thank Tero Harju and a referee for their suggestions.

References

[Adl94] L.M. Adleman. Molecular computation of solutions to combinatorial problems, *Science* 226:1021–1024, November 1994.

[Dic13] L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors, *American Journal of Mathematics* 35:413–422, 1913.

[FS97] H. Fernau, R. Stiebe. Regulation by valences, *Mathematical Foundations of Computer Science 1997*, Lecture Notes in Computer Science, volume 1295, Igor Prívara, Peter Ruzicka (eds.), 239–248, Springer-Verlag, 1997.

[FMR68] P.C. Fischer, A.R. Meyer, A.L. Rosenberg. Counter machines and counter languages, *Mathematical Systems Theory* 2:265–283, 1968.

[GG69] S. Ginsburg, S.A. Greibach. Abstract families of languages, *Memoirs of the American Mathematical Society* 87:1–32, 1969.

[Gre78] S.A. Greibach. Remarks on blind and partially blind one-way multicounter machines, *Theoretical Computer Science* 7:311–324, 1978.

[HH99] V. Halava, T. Harju. Languages accepted by integer weighted finite automata, *Jewels are forever*, J. Karhumki, H. Maurer, Gh. Păun, G. Rozenberg (eds.), 123–134, Springer-Verlag, 1999.

[HU79] J.E. Hopcroft, J.D. Ullman. *Introduction to automata theory, languages, and computation*, Addison-Wesley, 1979.

[KP+98] L. Kari, Gh. Păun, G. Rozenberg, A. Salomaa, S. Yu. DNA computing, sticker systems, and universality, *Acta Informatica* 35:401–420, 1998.

[Lat79] M. Latteux. Cônes rationnels commutatifs, *Journal of Computer and System Sciences* 18:307–333, 1979.

[Pău80] Gh. Păun. A new generative device: valence grammars, *Revue Roumaine de Mathématiques Pures et Appliquées* 25(6):911–924, 1980.

[PR98] Gh. Păun, G. Rozenberg. Sticker systems, *Theoretical Computer Science* 204:183–203, 1998.

[PRS98] Gh. Păun, G. Rozenberg, A. Salomaa. *DNA computing. New computing paradigms*, Springer-Verlag, 1998.

[RS97] G. Rozenberg, A. Salomaa (eds.). *Handbook of formal languages*, Springer-Verlag, 1997.