Undecidability in binary tag systems and the post correspondence problem for five pairs of words

Author(s):
Neary, Turlough

Publication date:
2015

Permanent link:
https://doi.org/10.3929/ethz-b-000106301

Originally published in:
Leibniz International Proceedings in Informatics (LIPIcs) 30, https://doi.org/10.4230/LIPIcs.STACS.2015.649

Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words

Turlough Neary

Institute of Neuroinformatics, University of Zürich and ETH Zürich, Switzerland
tneary@ini.phys.ethz.ch

Abstract

Since Cocke and Minsky proved 2-tag systems universal, they have been extensively used to prove the universality of numerous computational models. Unfortunately, all known algorithms give universal 2-tag systems that have a large number of symbols. In this work, tag systems with only 2 symbols (the minimum possible) are proved universal via an intricate construction showing that they simulate cyclic tag systems. We immediately find applications of our result. We reduce the halting problem for binary tag systems to the Post correspondence problem for 5 pairs of words. This improves on 7 pairs, the previous bound for undecidability in this problem. Following our result, only the cases for 3 and 4 pairs of words remains open, as the problem is known to be decidable for 2 pairs. In a further application, we apply the reductions of Vesa Halava and others to show that the matrix mortality problem is undecidable for sets with six 3×3 matrices and for sets with two 18×18 matrices. The previous bounds for the undecidability in this problem was seven 3×3 matrices and two 21×21 matrices.

1998 ACM Subject Classification F.1.2 [Theory of Computation]: Computation by Abstract Devices—Modes of Computation

Keywords and phrases tag system, Post correspondence problem, undecidability

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.649

1 Introduction

Introduced by Post [17], tag systems have been used to prove Turing universality in numerous computational models, including some of the simplest known universal systems [6, 10, 11, 14, 19, 20, 21]. Many universality results rely either on direct simulation of tag systems or on a chain of simulations the leads back to tag systems. Such relationships between models means that improvements in one model often has applications to many others. The results in [23] are a case in point, where an exponential improvement in the time efficiency of tag systems had the domino effect of showing that many of the simplest known models of computation [6, 10, 11, 14, 19, 20, 21] are in fact polynomial time simulators of Turing machines. Despite being central to the search for simple universal systems for 50 years, tag systems have not been the subject of simplification since the sixties.

In 1961, Minsky [13] solved Post's longstanding open problem by showing that tag systems, with deletion number 6, are universal. Soon after, Cocke and Minsky [5] proved that tag systems with deletion number 2 (2-tag systems) are universal. Later, Hao Wang [22] showed that 2-tag systems with even shorter instructions were universal. The systems of both Wang, and Cocke and Minsky use large alphabets and so have a large number of rules. Here we show that tag systems with only 2 symbols, and thus only 2 rules, are universal. Surprisingly, one of our two rules is trivial. We find immediate applications of our result. Using Cook's [6] reduction of tag systems to cyclic tag systems, it is a straightforward matter

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 649–661

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


to give a binary cyclic tag system program that is universal and contains only two 1 symbols. We also use our binary tag system construction to improve the bound for the number of pairs of words for which the Post correspondence problem [18] is undecidable, and the bounds for the simplest sets of matrices for which the mortality problem [16] is undecidable.

The search for the minimum number of word pairs for which the Post correspondence problem is undecidable began in the 1980s [4]. The best result until now was found by Matiyasevich and Sénizergues, whose impressive 3-rule semi-Thue system [12], along with a reduction due to Claus [4], showed that the problem is undecidable for 7 pairs of words. Improving on this undecidability bound of 7 pairs of words seemed like a challenging problem. In fact, Blondel and Tsitsiklis [2] stated in their survey "The decidability of the intermediate cases ($3 \leqslant n \leqslant 6$) is unknown but is likely to be difficult to settle". We give the first improvement on the bound of Matiyasevich and Sénizergues in 17 years: We reduce the halting problem for our binary tag system to the Post correspondence problem for 5 pairs of words. This leaves open only the cases for 3 and 4 pairs of words, as the problem is known to be decidable for 2 pairs [7].

A number of authors [1, 3, 8, 9, 16], have used undecidability bounds for the Post correspondence problem to find simple matrix sets for which the mortality problem is undecidable. The matrix mortality problem is, given a set of $d \times d$ integer matrices, decide if the zero matrix can be expressed as a product of matrices from the set. Halava et al. [9] proved the mortality problem undecidable for sets with seven 3×3 matrices, and using a reduction due Cassaigne and Karhumäki [3] they also showed the problem undecidable for sets with two 21×21 matrices. Applying the reductions used in [3, 8] to our new bound, we find that the matrix mortality problem is undecidable for sets with six 3×3 matrices and for sets with two 18×18 matrices.

In the sequel while simulating cyclic tag systems our binary tag system produces garbage that grows exponentially during the simulation, and this results in an exponential time simulation overhead.

2 Preliminaries

We write $c_1 \vdash c_2$ if a configuration c_2 is obtained from c_1 via a single computation step. We let $c_1 \vdash^t c_2$ denote a sequence of t computation steps. The length of a word w is denoted $|w|$, and ϵ denotes the empty word. We let $\langle v \rangle$ denote the encoding of v, where v is a symbol or a word. We use the binary modulo operation $a = m \bmod n$, where $a = m - ny$, $0 \leqslant a < n$, and a, m, n, and y are integers.

2.1 Tag Systems

▶ **Definition 1.** A tag system consists of a finite alphabet of symbols Σ, a finite set of rules $R : \Sigma \to \Sigma^*$ and a deletion number $\beta \in \mathbb{N}$, $\beta \geqslant 1$.

The tag systems we consider are deterministic. The computation of a tag system acts on a word $w = w_0 w_1 \dots w_{|w|-1}$ (here $w_i \in \Sigma$) which we call the *dataword*. The entire configuration is given by w. In a computation step, the symbols $w_0 w_1 \dots w_{\beta-1}$ are deleted and we apply the rule for w_0, i.e. a rule of the form $w_0 \to w_{0,1} w_{0,2} \dots w_{0,e}$, by appending the word $w_{0,1} w_{0,2} \dots w_{0,e}$ (here $w_{0,j} \in \Sigma$). A dataword (configuration) w' is obtained from w via a single computation step as follows:

$$w_0 w_1 \dots w_\beta \dots w_{|w|-1} \vdash w_\beta \dots w_{|w|-1} w_{0,1} w_{0,2} \dots w_{0,e}$$

where $w_0 \to w_{0,1}w_{0,2}\dots w_{0,e} \in R$. A tag system halts if $|w| < \beta$. We use the term *round* to describe the $\lfloor \frac{|w|}{\beta} \rfloor$ or $\lceil \frac{|w|}{\beta} \rceil$ computation steps that traverse a word w exactly once. We say a symbol w_0 is *read* if and only if at the start of a computation step it is the leftmost symbol (i.e. the rule $w_0 \to w_{0,0}w_{0,1}\dots w_{0,e}$ is applied), and we say a word $w = w_0w_1\dots w_{|w|-1}$ is *entered with shift* $z < \beta$ if w_z is the leftmost symbol that is read in w. We let $\underset{[z]}{w}$ denote the word obtained by removing the leftmost z symbols of w (i.e. $\underset{[z]}{w} = w_z\dots w_{|w|-1}$) and let $\overline{\underset{[z]}{w}}$ denote the sequence of symbols read during a single round on $\underset{[z]}{w}$. So $\overline{\underset{[z]}{w}} = w_z w_{z+\beta} w_{z+2\beta} w_{z+3\beta}, \dots, w_{z+l\beta}$ where $z + l\beta < |w|$. If $z < \beta$ then $\overline{\underset{[z]}{w}}$ is read when w is entered with shift z and we call $\overline{\underset{[z]}{w}}$ track z of w. A word w has a *shift change* of $0 \leqslant s < \beta$ if $|w| = y\beta - s$ where $y \in \mathbb{N}$ and $y > 0$. The proof of Lemma 2 is left to the reader.

▶ **Lemma 2** (shift change). *Given a tag system T with deletion number β and the word $rv \in \Sigma^*$, where the word r has a shift change of s and $|v| \geqslant \beta$, after one round of T on r entered with shift z the word v is entered with shift $(z + s) \bmod \beta$.*

2.2 Cyclic Tag Systems

Cyclic tag systems were introduced and proved universal by Cook [6].

▶ **Definition 3.** A cyclic tag system $\mathcal{C} = \alpha_0, \dots, \alpha_{p-1}$ is a list of words $\alpha_i \in \{0,1\}^*$ called appendants.

A *configuration* of a cyclic tag system consists of (i) a *marker* that points to a single appendant α_m in $\mathcal{C}$, and (ii) a word $w = w_1\dots w_{|w|} \in \{0,1\}^*$. We call w the *dataword*. Intuitively the list $\mathcal{C}$ is a program with the marker pointing to instruction α_m. In the initial configuration the marker points to appendant α_0 and w is the binary input word.

▶ **Definition 4.** A computation step is deterministic and acts on a configuration in one of two ways:

- If $w_1 = 0$ then w_1 is deleted and the marker moves to appendant $\alpha_{(m+1 \bmod p)}$.
- If $w_1 = 1$ then w_1 is deleted, the word α_m is appended onto the right end of w, and the marker moves to appendant $\alpha_{(m+1 \bmod p)}$.

A cyclic tag system completes its computation if (i) the dataword is the empty word or (ii) it enters a repeating sequence of configurations. As an example we give first 5 steps of the cyclic tag system $\mathcal{C} = 001, 01, 11$ on the input word 101. In each configuration $\mathcal{C}$ is given on the left with the marked appendant highlighted in bold font.

$$\begin{aligned} &\mathbf{001}, 01, 11 \quad 101 \quad \vdash \quad 001, \mathbf{01}, 11 \quad 01001 \quad \vdash \quad 001, 01, \mathbf{11} \quad 1001 \\ \vdash\ &\mathbf{001}, 01, 11 \quad 00111 \quad \vdash \quad 001, \mathbf{01}, 11 \quad 0111 \quad \vdash \quad 001, 01, \mathbf{11} \quad 111 \quad \vdash \quad \cdots \end{aligned}$$

3 The Halting Problem for Binary Tag Systems

▶ **Definition 5** (Halting problem for binary tag systems). Given a 2-symbol tag system $\mathcal{T}$ with deletion number β and a dataword w, does $\mathcal{T}$ produce a sequence of computation steps of the form $w \vdash^* w'$ where $|w'| < \beta$?

▶ **Theorem 6.** *The halting problem for binary tag systems is undecidable.*

The proof of Theorem 6 proceeds in two stages. First we construct a non-halting binary tag system $\mathcal{T}_\mathcal{C}$ that simulates an arbitrary cyclic tag system $\mathcal{C}$ (Sections 3.1 and 3.2). Following this in Lemma 9 we show how to modify our construction so that when simulating the cyclic tag system in [15] our system halts if an only if it is simulating a halting Turing machine.

Table 1 Table defining u. In the middle column is the sequence of symbols (track) read in u when the object in the left column is entered with shift $(\beta - 4m) \bmod \beta$, where $\beta = 4p$ is the deletion number, α_m is a cyclic tag system appendant, and $\langle\sigma_i\rangle'$ is a binary word where $\langle\sigma_i\rangle' = bbcbb$ if $\sigma_i = 0$, and $\langle\sigma_i\rangle' = bbbcb$ if $\sigma_i = 1$. Also $\underset{[1]}{\langle\sigma_1\rangle'}$ is the word $\langle\sigma_1\rangle'$ with its leftmost symbol removed.

Object	Track read in u	Values for m and α_m
u	$\underset{[(\beta-4m) \bmod \beta]}{\overline{u}} = c^s$	$0 \leqslant m < p$
$\langle\epsilon\rangle = bubbb$	$\underset{[\beta-1]}{\overline{u}} = cbbb(bcbbb)^{p-1}c^{s-5p+1}$	$m = 0$
	$\underset{[\beta-4m-1]}{\overline{u}} = c^s$	$0 < m < p$
$\langle 0\rangle = bbubb$	$\underset{[\beta-2]}{\overline{u}} = cbbb(bcbbb)^{p-1}c^{s-5p+1}$	$m = 0$
	$\underset{[\beta-4m-2]}{\overline{u}} = c^s$	$0 < m < p$
$\langle 1\rangle = bbbub$	$\underset{[\beta-3]}{\overline{u}} = cbbb(bcbbb)^{p-1}c^{s-5p+1}$	$\alpha_0 = \epsilon, \quad m = 0$
	$\underset{[\beta-3]}{\overline{u}} = \underset{[1]}{\langle\sigma_1\rangle'}\langle\sigma_2\rangle' \dots \langle\sigma_v\rangle' c^{s-5v+1}$	$\alpha_0 = \sigma_1\sigma_2 \dots \sigma_v$ for $v > 0, \quad m = 0$
	$\underset{[\beta-4m-3]}{\overline{u}} = \underset{[1]}{\langle\sigma_1\rangle'}\langle\sigma_2\rangle' \dots \langle\sigma_v\rangle' c^{s-5v}$	$\alpha_m = \sigma_1\sigma_2 \dots \sigma_v$ for $0 < m < p$

3.1 Binary Tag System $\mathcal{T}_\mathcal{C}$ and Its Encoding

Here we give a binary tag system $\mathcal{T}_\mathcal{C}$ that simulates the computation of an arbitrary $\mathcal{C} = \alpha_0, \dots, \alpha_{p-1}$. The deletion number of $\mathcal{T}_\mathcal{C}$ is $\beta = 4p$, its alphabet is $\{b, c\}$, and its rules are of the form $b \to b$ and $c \to u$, where $u \in \{b, c\}^*$. The binary word u encodes the entire program of $\mathcal{C}$ and is defined by Table 1, where $|u| = \beta s$, $s \geqslant 5(max(p, r))$ and r is the length of the longest appendant in $\mathcal{C}$. See Section 3.3.1 for an example of how Table 1 is used to give u.

The cyclic tag system symbols 0 and 1 are encoded as the binary words $\langle 0\rangle = bbubb$ and $\langle 1\rangle = bbbub$ respectively. We refer to $\langle 0\rangle$ and $\langle 1\rangle$ as objects.

▶ **Definition 7** (Input to $\mathcal{T}_\mathcal{C}$)**.** An arbitrary input dataword $w_1w_2 \dots w_n \in \{0, 1\}^*$ to a cyclic tag system is encoded as the $\mathcal{T}_\mathcal{C}$ input dataword $\langle w_1\rangle\langle w_2\rangle \dots \langle w_n\rangle$.

During the simulation we make use of an extra *garbage* object: the binary word $\langle\epsilon\rangle = bubbb$. The cyclic tag system configuration $(\alpha_0, \alpha_1 \dots \alpha_{m-1}\boldsymbol{\alpha_m}\alpha_{m+1} \dots \alpha_{p-1} \quad w_1w_2 \dots w_l)$ is encoded as

$$\underset{[(\beta-4m) \bmod \beta]}{\langle w_1\rangle} \{\langle\epsilon\rangle^p, u\}^*\langle w_2\rangle\{\langle\epsilon\rangle^p, u\}^*\langle w_3\rangle \dots \{\langle\epsilon\rangle^p, u\}^*\langle w_l\rangle\{\langle\epsilon\rangle^p, u\}^* \tag{1}$$

where $\underset{[(\beta-4m) \bmod \beta]}{\langle w_1\rangle}$ denotes the word given by an object $\langle w_1\rangle \in \{\langle 0\rangle, \langle 1\rangle\}$ with its leftmost $[\beta - 4m) \bmod \beta]$ symbols deleted. This implies that $\langle w_1\rangle$ is entered with the shift $[(\beta - 4m) \bmod \beta]$ and this shift value records that the currently marked cyclic tag system appendant is α_m. If a u subword in the dataword of $\mathcal{T}_\mathcal{C}$ does not form part of one of the three objects $\langle 0\rangle$, $\langle 1\rangle$ and $\langle\epsilon\rangle$ we will refer to this u subword as a garbage object. So words of the form $\{\langle\epsilon\rangle, u\}^*$ in Equation (1) consist only of garbage objects.

3.2 The Simulation Algorithm

The sequence of symbols that is read in an object is determined by the shift value with which it is entered (see Section 2.1). So in the simulation the shift value is used for algorithm

(i) $\underset{[\beta-4m]}{\langle 1\rangle}\ \langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \vdash^s \underset{[(\beta-4(m+1)) \bmod \beta]}{\langle a_2\rangle}\ \langle a_3\rangle\dots\langle a_l\rangle\langle\sigma_1\rangle\langle\sigma_2\rangle\dots\langle\sigma_v\rangle u^{s-5v}$

(ii) $\underset{[\beta-4m]}{\langle 0\rangle}\ \langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \vdash^s \underset{[(\beta-4(m+1)) \bmod \beta]}{\langle a_2\rangle}\ \langle a_3\rangle\dots\langle a_l\rangle u^s$

(iii) $\underset{[\beta-4m]}{\langle \epsilon\rangle}\ \langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \vdash^s \underset{[(\beta-4(m+1)) \bmod \beta]}{\langle a_2\rangle}\ \langle a_3\rangle\dots\langle a_l\rangle u^s$

(iv) $\underset{[(\beta-4m) \bmod \beta]}{u}\ \langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \vdash^s \underset{[(\beta-4m) \bmod \beta]}{\langle a_2\rangle}\ \langle a_3\rangle\dots\langle a_l\rangle u^s$

Figure 1 Objects $\langle 1\rangle$, $\langle 0\rangle$, $\langle\epsilon\rangle$ and u being read when entered with shift $[\beta-4m]$. Above $\vdash^s$ denotes the s computation steps that read each object, $a_i \in \{u, \langle 0\rangle, \langle 1\rangle, \langle\epsilon\rangle\}$, $\beta = 4p$ is the deletion number, and p is the length of $\mathcal{C}$'s program. In (i), (ii) and (iii) $0 < m < p$ and in (iv) $0 \leqslant m < p$. On the left is the dataword before the object is read and on the right is the dataword after the object has been read. After $\langle 1\rangle$, $\langle 0\rangle$ or $\langle\epsilon\rangle$ is read the adjacent object $\langle a_2\rangle$ is entered with shift $[\beta-4(m+1)]$, and in (iv) after u is read $\langle a_2\rangle$ is entered with shift $[(\beta-4m) \bmod \beta]$. Reading the objects $\langle 0\rangle$, $\langle\epsilon\rangle$ or u appends u^s, and reading the object $\langle 1\rangle$ appends the encoding of cyclic tag system appendant $\alpha_m = \sigma_1\sigma_2\dots\sigma_v$.

control flow. Figures 1 and 2 give a high level view of reading each of the four objects that appear in Equation (1) and this view includes the shift change that occurs from reading each object. Objects $\langle 1\rangle$, $\langle 0\rangle$ and $\langle\epsilon\rangle$ have length $|\langle 1\rangle| = |\langle 0\rangle| = |\langle\epsilon\rangle| = \beta s + 4$ and so have a shift change of $\beta - 4$ (see end of Section 2.1). So when these objects are entered with shift $[(\beta-4m) \bmod \beta]$ the adjacent object $\langle a_2\rangle$ is entered with shift $[(\beta-4(m+1)) \bmod \beta]$ (as shown in Figures 1 and 2). When reading $\langle 1\rangle$ or $\langle 0\rangle$ this shift change of $\beta-4$ simulates the appendant marker moving from appendant α_m to $\alpha_{((m+1) \bmod p)}$. Recall that $\beta = 4p$ and so reading p of the $\langle 0\rangle$ and $\langle 1\rangle$ objects has a shift change of 0 (since $0 = p(\beta-4) \bmod \beta$). This shift change of 0 means that the encoding of the marked appendant returns to its original value after reading p objects, correctly simulating that the appendant marker beginning at α_m has moved through the entire length p circular program of $\mathcal{C}'$ and returned to it original position of marking α_m. The garbage objects $\langle\epsilon\rangle$ and u in Equation (1) appear only in garbage words of the form $\{\langle\epsilon\rangle^p, u\}^*$ which have a shift change of 0 (since $|\langle\epsilon\rangle^p| = (1+ps)\beta$ and $|u| = \beta s$) and so have no effect on the shift value when entering subsequent objects (see Lemma 2). In Figures 1 and 2 we see that reading an $\langle 1\rangle$ simulates reading a 1 by appending the encoding of the marked appendant $\alpha_m = \sigma_1\sigma_2\dots\sigma_v$ and reading an $\langle 0\rangle$ simulates reading a 0 by appending a garbage word from $\{\langle\epsilon\rangle^p, u\}^*$ to simulate that nothing is appended. In Figures 1 and 2 when the garbage objects $\langle\epsilon\rangle$ and u are read they append garbage words from $\{\langle\epsilon\rangle^p, u\}^*$ that have no effect on the simulation.

To help the reader, in Section 3.3 we define u for a specific example and use this value to give an example of our algorithm simulating a cyclic tag system computation step. Here we explain how Table 1 defines u so that when each object is read it appends the appendants shown in Figures 1 and 2. Recall that a track $\underset{[z]}{\overline{w}}$ is the sequence of symbols read in a word w when it is entered with shift z (see Section 2.1). Table 1 defines u by giving (in the middle column) each possible track in u (for detailed example see Section 3.3.1). Each u track is read when the object in the left column of Table 1 is entered with shift $[(\beta-4m) \bmod \beta]$. To see this note that the number of b symbols that proceed the u word in each object causes a shift change which in turn causes the u track given in middle column of Table 1 to be read. For example the word bbb at the left end of $\langle 1\rangle = bbbub$ has a shift change value of $\beta-3$ (see Section 2.1) and thus when $\langle 1\rangle$ is entered with shift $[(\beta-4m) \bmod \beta]$ we enter u with shift $[\beta-4m-3]$. So when $m > 0$ and we read an $\langle 1\rangle$, we see from row 8 of Table 1 that track $\underset{[\beta-4m-3]}{\overline{u}} = \langle\sigma_1\rangle'\langle\sigma_2\rangle'\dots\langle\sigma_v\rangle' c^{s-5v}$ is read. Applying the rules

$$\text{(i)} \quad \underset{[0]}{\langle 1\rangle}\langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \quad \vdash^{s+1} \quad \underset{[\beta-4]}{\langle a_2\rangle}\langle a_3\rangle\dots\langle a_l\rangle\langle\sigma_1\rangle\langle\sigma_2\rangle\dots\langle\sigma_v\rangle u^{s-5v+1}$$

$$\text{(ii)} \quad \underset{[0]}{\langle 0\rangle}\langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \quad \vdash^{s+1} \quad \underset{[\beta-4]}{\langle a_2\rangle}\langle a_3\rangle\dots\langle a_l\rangle\langle\epsilon\rangle^p u^{s-5p+1}$$

$$\text{(iii)} \quad \underset{[0]}{\langle \epsilon\rangle}\langle a_2\rangle\langle a_3\rangle\dots\langle a_l\rangle \quad \vdash^{s+1} \quad \underset{[\beta-4]}{\langle a_2\rangle}\langle a_3\rangle\dots\langle a_l\rangle\langle\epsilon\rangle^p u^{s-5p+1}$$

Figure 2 Objects $\langle 1\rangle$, $\langle 0\rangle$ and $\langle\epsilon\rangle$ being read when entered with shift 0. Above $\vdash^{s+1}$ denotes the $s+1$ computation steps that read each object when entered with shift 0. Here $a_i \in \{u, \langle 0\rangle, \langle 1\rangle, \langle\epsilon\rangle\}$, $\beta = 4p$ is the deletion number, and p is the length of $\mathcal{C}$'s program. On the left is the dataword before the object is read and on the right is the dataword after the object has been read. After $\langle 1\rangle$, $\langle 0\rangle$ or $\langle\epsilon\rangle$ is read the adjacent object a_2 is entered with shift $[\beta-4]$. Reading the objects $\langle 0\rangle$ or $\langle\epsilon\rangle$ appends $\langle\epsilon\rangle^p u^{s-5p+1}$, and reading the object $\langle 1\rangle$ appends the encoding of cyclic tag system appendant $\alpha_0 = \sigma_1\sigma_2\dots\sigma_v$.

$b \to b$ and $c \to u$ when reading this track appends the appendant $\langle\sigma_1\rangle\langle\sigma_2\rangle\dots\langle\sigma_v\rangle c^{s-5v}$ as shown in Figure 1 (i). For $m > 0$ the entire sequence of symbol read in $\langle 1\rangle$, $\langle 0\rangle$, and $\langle\epsilon\rangle$ are respectively given by the u tracks in rows 3, 5 and 8 of Table 1. One can see that applying the rules $b \to b$ and $c \to u$ to these tracks appends the correct appendant for each object read in Figure 1. For $m = 0$ we have a special case where to get the entire sequence of symbols read in each object we prepend an extra b to the u tracks given in rows 2, 4, 6 and 7 of Table 1 and now applying the rules $b \to b$ and $c \to u$ to this sequence appends the appendants shown in Figure 2. For example, to give the sequence read in $\langle 1\rangle$ when $m = 0$, we prepend b to track $\underset{[\beta-3]}{\overline{u}} = \underset{[1]}{\langle\sigma_1\rangle'\langle\sigma_2\rangle'\dots\langle\sigma_v\rangle' c^{s-5v+1}}$ (row 7 of Table 1) to give the sequence $b\underset{[1]}{\langle\sigma_1\rangle'\langle\sigma_2\rangle'\dots\langle\sigma_v\rangle' c^{s-5v+1}} = \langle\sigma_1\rangle'\langle\sigma_2\rangle'\dots\langle\sigma_v\rangle' c^{s-5v+1}$ which appends the word $\langle\sigma_1\rangle\langle\sigma_2\rangle\dots\langle\sigma_v\rangle c^{s-5v+1}$ as shown in Figure 2 (i). Finally, in row 1 of Table 1 are the u tracks that give the sequence of symbols read in u garbage objects for all values of m.

3.2.1 Tag system $\mathcal{T}_\mathcal{C}$ Simulating an Arbitrary Computation Step of $\mathcal{C}$

Equation (2) gives an arbitrary computation step of cyclic tag system $\mathcal{C}$, where $w' = \sigma_1\dots\sigma_v$ if $w_1 = 1$ and $\alpha_m = \sigma_1\sigma_2\dots\sigma_v$, and $w' = \epsilon$ if $w_1 = 0$.

$$\alpha_0, \dots \boldsymbol{\alpha_m} \alpha_{m+1} \dots \alpha_{p-1} \quad w_1 w_2 \dots w_l \quad \vdash \quad \alpha_0, \dots \alpha_m \boldsymbol{\alpha_{m+1}} \dots \alpha_{p-1} \quad w_2 \dots w_l w' \tag{2}$$

Lemma 8 essentially states that $\mathcal{T}_\mathcal{C}$ simulates the arbitrary computation step given in Equation (2). In Lemma 8, Equations (3) and (4) respectively encode the left and right configurations in Equation (2) (see Equation (1)).

▶ **Lemma 8** ($\mathcal{T}_\mathcal{C}$ simulates an arbitrary computation step of $\mathcal{C}$). *Given a dataword of the form*

$$\underset{[(\beta-4m) \bmod \beta]}{\langle w_1\rangle} \{\langle\epsilon\rangle^p, u\}^*\langle w_2\rangle\{\langle\epsilon\rangle^p, u\}^*\langle w_3\rangle\dots\{\langle\epsilon\rangle^p, u\}^*\langle w_l\rangle\{\langle\epsilon\rangle^p, u\}^* \tag{3}$$

where $w_i \in \{0, 1\}$, $\mathcal{T}_\mathcal{C}$ reads the word $\underset{[(\beta-4m) \bmod \beta]}{\langle w_1\rangle} \{\langle\epsilon\rangle^p, u\}^*$ to give a dataword of the form

$$\underset{[(\beta-4(m+1)) \bmod \beta]}{\langle w_2\rangle} \{\langle\epsilon\rangle^p, u\}^*\langle w_3\rangle\dots\{\langle\epsilon\rangle^p, u\}^*\langle w_l\rangle\{\langle\epsilon\rangle^p, u\}^*\langle w'\rangle\{\langle\epsilon\rangle^p, u\}^* \tag{4}$$

where $\langle w'\rangle = \langle\sigma_1\rangle\langle\sigma_2\rangle\dots\langle\sigma_v\rangle$ if $w_1 = 1$ and $\alpha_m = \sigma_1\sigma_2\dots\sigma_v$, and $\langle w'\rangle = \epsilon$ if $w_1 = 0$.

Proof. We use Figures 1 and 2 to verify that given Configuration (3), $\mathcal{T}_{\mathcal{C}}$ produces Configuration (4). Following this we discuss the correctness of Figures 1 and 2.

From (i) and (ii) of Figures 1 and 2 there are 4 possible cases for reading $\langle w_1 \rangle$, where each case is determined by the value of $\langle w_1 \rangle \in \{\langle 1 \rangle, \langle 0 \rangle\}$ and the shift ($[0]$ or $[\beta - 4m]$ for $m > 0$) with which it is entered. The technique for verifying that $\mathcal{T}_{\mathcal{C}}$ produces Configuration (4) from Configuration (3) is similar for all 4 cases and so we will only go through one case. We choose the case where $\langle w_1 \rangle = \langle 1 \rangle$ and is entered with shift $[\beta - 4m]$ for $m > 0$. From Figure 1 (i) when we read $\langle w_1 \rangle = \langle 1 \rangle$ with shift $[\beta - 4m]$ in Configuration (3) we get

$$\underset{[(\beta-4(m+1) \bmod \beta]}{\{\langle \epsilon \rangle^p, u\}^*} \langle w_2 \rangle \{\langle \epsilon \rangle^p, u\}^* \langle w_3 \rangle \dots \{\langle \epsilon \rangle^p, u\}^* \langle w_l \rangle \{\langle \epsilon \rangle^p, u\}^* \langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle u^{s-5v} \tag{5}$$

In Configuration (5) the word $\{\langle \epsilon \rangle^p, u\}^*$ is entered with shift $[(\beta - 4(m+1) \bmod \beta]$. Each $\langle \epsilon \rangle$ has a shift change of $\beta - 4$ and each u has a shift change of 0, and so as we read the word $\{\langle \epsilon \rangle^p, u\}^*$ the $\langle \epsilon \rangle$ and u objects are entered with shifts of the form $[(\beta - 4m') \bmod \beta]$ where $0 \leqslant m' < p$. So the cases for reading the objects in the word $\{\langle \epsilon \rangle^p, u\}^*$ are given by (iii) in Figures 1 and 2 and (iv) in Figure 1. Thus when we read the word $\{\langle \epsilon \rangle^p, u\}^*$ at the left end of Configuration (5) we append a word of the form $\{\langle \epsilon \rangle^p, u\}^*$ as shown in Configuration (6). Recall that words of the form $\{\langle \epsilon \rangle^p, u\}^*$ have a shift change of $[0]$ and so when we enter $\{\langle \epsilon \rangle^p, u\}^*$ with shift $[\beta - 4(m+1) \bmod \beta]$ as shown in Configuration (5) we also enter the adjacent object $\langle w_2 \rangle$ with shift $[\beta - 4(m+1) \bmod \beta]$ as shown in Configuration (6).

$$\underset{[(\beta-4(m+1) \bmod \beta]}{\langle w_2 \rangle} \{\langle \epsilon \rangle^p, u\}^* \langle w_3 \rangle \dots \{\langle \epsilon \rangle^p, u\}^* \langle w_l \rangle \{\langle \epsilon \rangle^p, u\}^* \langle \sigma_1 \rangle \dots \langle \sigma_v \rangle u^{s-5v} \{\langle \epsilon \rangle^p, u\}^* \tag{6}$$

Configuration (6) is of the form of Configuration (4) for the case where $\langle w_1 \rangle = \langle 1 \rangle$ and is entered with shift $[(\beta - 4m) \bmod \beta]$ for $m > 0$. So for this case we have shown using Figures 1 and 2 that given Configuration (3) $\mathcal{T}_{\mathcal{C}}$ produces Configuration (4). The correctness of Figures 1 and 2 can be verified by showing that on each line of these figures, reading the leftmost object in the configuration on the left produces the configuration on the right. This is achieved by showing that when we enter the adjacent object $\langle a_2 \rangle$ with the correct shift and also that the correct word gets appended at the end of the dataword. The shift value with which $\langle a_2 \rangle$ is entered follows immediately from Lemma 2 and the length of the object being read. We direct the reader to paragraph 2 of Section 3.2 where we explain how Table 1 defines u so that when each object is read it appends the appendants shown in Figures 1 and 2. ◀

Our first main Theorem (Theorem 6) follows from Lemma 9. For tag system $\mathcal{T}'_{\mathcal{C}}$ in Lemma 9 we restrict the type of input allowed as this lets us simulate $\mathcal{T}'_{\mathcal{C}}$ in Theorem 11 when proving the Post correspondence problem undecidable for 5 pairs of words.

▶ **Lemma 9.** *Let $\mathcal{T}'_{\mathcal{C}}$ be an arbitrary binary tag system with deletion number β, alphabet $\{b, c\}$ and rules of the form $b \to b$ and $c \to u_0 u_1 \dots u_l b$ ($u_i \in \{b, c\}$). The halting problem is undecidable for tag systems of the form of $\mathcal{T}'_{\mathcal{C}}$ when given $u_{\beta-1} u_\beta \dots u_l b$ as input.*

Proof. In [15] the cyclic tag system $\mathcal{C}$ simulates the computation of an arbitrary Turing machine and appends an appendant (which we will call α_h) encoding the Turing machine halt state if and only if the simulated Turing machine halts. So given the cyclic tag system $\mathcal{C}$, its input $w_1 w_2 \dots w_n$, and an appendant α_h, we construct a binary tag system $\mathcal{T}'_{\mathcal{C}}$ that takes $u_{\beta-1} u_\beta \dots u_l b$ as input and simulates $\mathcal{C}$ on input w halting if an only if $\mathcal{C}$ appends α_h. It then follows that halting problem is undecidable for tag systems of the form of $\mathcal{T}'_{\mathcal{C}}$ when given $u_{\beta-1} u_\beta \dots u_l b$ as input.

Table 2 Table defining u for tag system in Lemma 9. In the middle column is the sequence of symbols (or track) read in u when the object in the left column is entered with shift $[(\beta - 10m + 1) \bmod \beta]$, where $\beta = 10p$ is the deletion number, α_m is a cyclic tag system appendant, $\langle\epsilon\rangle' = b^4cb^6$, and $\langle\sigma_i\rangle' = b^6cb^4$ if $\sigma_i = 0$, and $\langle\sigma_i\rangle' = b^8cb^2$ if $\sigma_i = 1$. Also $\underset{[1]}{\langle\epsilon\rangle'}$ and $\underset{[1]}{\langle\sigma_1\rangle'}$ are the words $\langle\epsilon\rangle'$ and $\langle\sigma_1\rangle'$ with their leftmost symbol removed.

Object	Track read in u	Values for m and α_m
input track	$\underset{[\beta-1]}{\overline{u}} = b^{\beta-2}\langle w_1'\rangle \dots \langle w_n'\rangle u^{s-11(n+p)-\beta+2}(\langle\epsilon\rangle')^p$	
u	$\underset{[(\beta-10m+1) \bmod \beta]}{\overline{u}} = c^s$	$0 \leqslant m < p$
$\langle\epsilon\rangle = b^4ub^6$	$\underset{[\beta-3]}{\overline{u}} = \underset{[1]}{\langle\epsilon\rangle'}(\langle\epsilon\rangle')^{p-1}c^{s-11p+1}$	$m = 0$
	$\underset{[\beta-10m-3]}{\overline{u}} = (\langle\epsilon\rangle')^p c^{s-11p}$	$0 < m < p$
$\langle 0\rangle = b^6ub^4$	$\underset{[\beta-5]}{\overline{u}} = \underset{[1]}{\langle\epsilon\rangle'}(\langle\epsilon\rangle')^{p-1}c^{s-11p+1}$	$m = 0$
	$\underset{[\beta-10m-5]}{\overline{u}} = (\langle\epsilon\rangle')^p c^{s-11p}$	$0 < m < p$
$\langle 1\rangle = b^8ub^2$	$\underset{[\beta-7]}{\overline{u}} = \underset{[1]}{\langle\sigma_1\rangle'}\langle\sigma_2\rangle' \dots \langle\sigma_v\rangle' c^{s-11v+1}$	$\alpha_0 = \sigma_1 \dots \sigma_v, \quad m = 0$
	$\underset{[\beta-10m-7]}{\overline{u}} = \langle\sigma_1\rangle'\langle\sigma_2\rangle' \dots \langle\sigma_v\rangle' c^{s-11v}$	$\alpha_m = \sigma_1 \dots \sigma_v$, $0 < m < p$, $m \neq h$
	$\underset{[\beta-10h-7]}{\overline{u}} = bu^{s-1}$	
halting tracks	$\underset{[2i]}{\overline{u}} = b^s$	$i = \{0, 1, 2, 3, \dots \frac{\beta-2}{2}\}$

We obtain $\mathcal{T}_{\mathcal{C}}'$ by modifying tag system $\mathcal{T}_{\mathcal{C}}$ from Section 3.1. In $\mathcal{T}_{\mathcal{C}}'$, u is defined by Table 2 and we have $\langle 0\rangle = b^6ub^4$, $\langle\epsilon\rangle = b^4ub^6$, $\langle 1\rangle = b^8ub^2$. The deletion number is now $\beta = 10p$, and $|u| = \beta s$ with $s = 11(max(p+n+\beta-2, r))$. On input $u_{\beta-1}u_\beta \dots u_l b$ tag system $\mathcal{T}_{\mathcal{C}}'$ reads track $\underset{[\beta-1]}{\overline{u}}$ and from row 1 of Table 2 this appends $b^{\beta-2}\langle w_1\rangle\langle w_2\rangle \dots \langle w_n\rangle u^{s-11(n+p)-\beta+2}(\langle\epsilon\rangle)^p$. So $\mathcal{T}_{\mathcal{C}}'$ begins its computation by appending the encoding of the input to $\mathcal{C}$. Reading $u_{\beta-1}u_\beta \dots u_l b$, which has a shift change of $\beta - 1$, causes us to enter the encoded dataword with a shift of $\beta - 1$ and this means that we can enter u garbage objects with shift $\beta - 1$ and read the track that appends the encoded input. To avoid this we append the word $b^{\beta-2}$ to the left of $\langle w_1\rangle$ and since $b^{\beta-2}$ has a shift change of 2 the encoded input is entered with shift [1] (instead of $[\beta - 1]$). After reading this $b^{\beta-2}$ $\mathcal{T}_{\mathcal{C}}'$ begins the simulation of $\mathcal{C}$ on input w, making use of the same algorithm as $\mathcal{T}_{\mathcal{C}}$. Since we enter the encoding of the input word w with shift [1] and objects $\langle 0\rangle = b^6ub^4$ and $\langle 1\rangle = b^8ub^2$ now have a shift change of $\beta - 10$, $\mathcal{T}_{\mathcal{C}}'$ enters objects with shifts of the form $[(\beta - 10m + 1)) \bmod \beta]$. This means that when reading garbage object u and objects $\langle\epsilon\rangle$, $\langle 0\rangle$, and $\langle 1\rangle$, we enter u with shifts of $[(\beta - 10m + 1)) \bmod \beta]$, $[(\beta - 10m - 3)) \bmod \beta]$, $[(\beta - 10m - 5)) \bmod \beta]$ and $[(\beta - 10m - 7)) \bmod \beta]$ respectively. This gives the shift values for the tracks in Table 2. By comparing the tracks in Tables 1 and 2 we can see that the when objects in $\mathcal{T}_{\mathcal{C}}'$ are read they append similar sequences of objects to those in $\mathcal{T}_{\mathcal{C}}$. So the computation of $\mathcal{T}_{\mathcal{C}}'$ proceeds in the same manner as the computation of $\mathcal{T}_{\mathcal{C}}$. However, if $\mathcal{C}$ appends α_h then we enter $\langle 1\rangle$ with shift $[\beta - 6h]$ and track $\underset{[\beta-6h-4]}{\overline{u}} = bc^{s-1}$ is read appending the word bu^{s-1}. When bu^{s-1} is read during the next traversal of the dataword the single b in this word causes a shift change of $\beta - 1$ which means that all u subwords in the dataword of $\mathcal{T}_{\mathcal{C}}'$ will now be entered with an even valued shift. From row 10 of Table 1 all even valued tracks in u append only b symbols and so after one further traversal the dataword consists entirely of b symbols. After this the rule $b \to b$, which appends one b and deletes β symbols, is repeated until the number of symbols is $< \beta$ and the computation halts. ◀

Table 3 Table defining u for the cyclic tag system $\mathcal{C} = 10, 0$. In the middle column is the sequence of symbols (track) read in u when the object in the left column is entered with shift $(8 - 4m)$ mod 8, where α_m is a cyclic tag system appendant, $\langle 1\rangle' = bbbcb$, $\langle 0\rangle' = bbcbb$ and $\underset{[1]}{\langle 0\rangle'} = bcbb$.

Object	Track read in u	Values for m and α_m
u	$\underset{[(8-4m) \bmod 8]}{\overline{u}} = c^{10}$	$0 \leqslant m < 2$
$\langle\epsilon\rangle = bubbb$	$\underset{[7]}{\overline{u}} = cbbb(bcbbb)c$	$m = 0$
	$\underset{[3]}{\overline{u}} = c^{10}$	$m = 1$
$\langle 0\rangle = bbubb$	$\underset{[6]}{\overline{u}} = cbbb(bcbbb)c$	$m = 0$
	$\underset{[2]}{\overline{u}} = c^{10}$	$m = 1$
$\langle 1\rangle = bbbub$	$\underset{[5]}{\overline{u}} = \underset{[1]}{\langle 0\rangle'} c^6$	$\alpha_0 = 0$
	$\underset{[1]}{\overline{u}} = \langle 0\rangle'\langle 1\rangle'$	$\alpha_1 = 01$

3.3 Example Simulation for $\mathcal{T}_{\mathcal{C}}$

3.3.1 Using Table 1 to define u

To help explain how Table 1 is used to define u, we take the example of defining u for the cyclic tag system program $\mathcal{C} = 0, 01$. The value for u is given in Equation (7), where to improve readability, we have split u into two equal length subwords u' and u'' with a space between every fourth symbol.

$$\begin{aligned} u &= u'u'' \\ u' &= cbcc\ cbcc\ cbcc\ ccbb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ ccbb \\ u'' &= cbcc\ cccc\ cbcc\ ccbb\ cbcc\ ccbb\ cccc\ ccbb\ cbcc\ cccc \end{aligned} \tag{7}$$

From Section 3.1 when $\mathcal{C} = 0, 01$ we have $p = 2$, $s \geqslant 10$, $\beta = 8$, $|u| = 80$, $\alpha_0 = 0$ and $\alpha_1 = 01$. By substituting these values into Table 1 we get Table 3 which defines u for our tag system that simulates $\mathcal{C} = 0, 01$. Table 3 defines the word u as a series of tracks. Recall from Section 2.1 that a track $\underset{[z]}{\overline{w}} = w_z w_{z+\beta} w_{z+2\beta}, \ldots, w_{z+l\beta}$ in a word w is the sequence of symbol read in that word when it is entered with shift z. Here $\beta = 8$ and so for $m = 0$, row 1 of Table 3 defines track $\underset{[0]}{\overline{u}} = u_0 u_8 u_{16} \ldots u_{72} = c^{10}$, which is shown in bold below.

$$\begin{aligned} u' &= \mathbf{c}bcc\ cbcc\ \mathbf{c}bcc\ ccbb\ \mathbf{c}ccc\ cbbb\ \mathbf{c}bcc\ cbbb\ \mathbf{c}bcc\ ccbb \\ u'' &= \mathbf{c}bcc\ cccc\ \mathbf{c}bcc\ ccbb\ \mathbf{c}bcc\ ccbb\ \mathbf{c}ccc\ ccbb\ \mathbf{c}bcc\ cccc \end{aligned}$$

Taking row 2 of Table 3 gives $\underset{[7]}{\overline{u}} = u_7 u_{15} u_{23} \ldots u_{79} = cbbb(bcbbb)c$ which again is given in bold immediately below.

$$\begin{aligned} u' &= cbcc\ cbc\mathbf{c}\ cbcc\ ccb\mathbf{b}\ cccc\ cbb\mathbf{b}\ cbcc\ cbb\mathbf{b}\ cbcc\ ccb\mathbf{b} \\ u'' &= cbcc\ ccc\mathbf{c}\ cbcc\ ccb\mathbf{b}\ cbcc\ ccb\mathbf{b}\ cccc\ ccb\mathbf{b}\ cbcc\ ccc\mathbf{c} \end{aligned}$$

Rows 3 to 7 of Table 3 give the tracks that complete the definition of u for our tag system that simulates the cyclic tag system $\mathcal{C} = 10, 0$.

3.3.2 Simulating a Computation Step with $\mathcal{T}_{\mathcal{C}}$.

In this section we give the low level details of our simulation algorithm for $\mathcal{T}_{\mathcal{C}}$ by simulating the first computation step ($\mathbf{0}, 01 \quad 11 \quad \vdash \quad 0, \mathbf{01} \quad 10$) of cyclic tag system $\mathcal{C} = 0, 01$ on the input dataword 11. The input dataword 11 to $\mathcal{C}$ is encoded via Definition 7 as the tag system dataword $\langle 1\rangle\langle 1\rangle$ and using $\langle 1\rangle = bbbub$ and Equation (7) this can be rewritten as

$$\overbrace{bbb\underbrace{cbcc\ cbcc\ cbcc\ ccbb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ ccbb}_{u'}u''b}^{\langle 1\rangle}\ \langle 1\rangle \tag{8}$$

Because the dataword is quite long we only give the left end of the dataword as b and c symbols and use higher level objects on the right. In configuration (8) the leftmost symbol is b and so we apply the rule $b \to b$ by appending b and deleting the leftmost 8 symbols from the dataword to give

$$\vdash \qquad \underbrace{bcc\ cbcc\ ccbb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ ccbb}_{\substack{u'\\ [5]}}\ u''b\ \langle 1\rangle\ b \tag{9}$$

Above u is entered with shift 5 and so we begin reading track $\overline{u}_{[5]}$ from Table 3. We apply the rules $b \to b$ and $c \to u$ of $\mathcal{T}_{\mathcal{C}}$ to give the next four computation steps

$$\begin{aligned}
\vdash &\qquad cbb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ ccbb\, u''b\ \langle 1\rangle\ bb\\
\vdash &\qquad bbb\ cbcc\ cbbb\ cbcc\ ccbb\, u''b\ \langle 1\rangle\ bbu\\
\vdash &\qquad bbb\ cbcc\ ccbb\, u''b\ \langle 1\rangle\ bbub\\
\vdash &\qquad cbb\ u''b\ \langle 1\rangle\ \underbrace{bbubb}_{\langle 0\rangle}
\end{aligned}$$

During the first 5 computation steps the word $\langle 0\rangle = bbubb$ was appended to the dataword. Below we give a rewritten form of the configuration immediately above where $bbubb$ is replaced with $\langle 0\rangle$ and u'' is replaced with its value from Equation (7). We also give the next 5 computation steps

$$\begin{aligned}
&cbb\ \underbrace{cbcc\ cccc\ cbcc\ ccbb\ cbcc\ ccbb\ cccc\ ccbb\ cbcc\ cccc}_{u''}b\ \langle 1\rangle\langle 0\rangle\\
\vdash^5 &\qquad ccc\, b\ \langle 1\rangle\langle 0\rangle\ u^5
\end{aligned}$$

Below we have rewritten the configuration given immediately above and given the last configuration in this simulated computation step.

$$ccc\, b\ \overbrace{bbb\underbrace{cbcc\ cbcc\ cbcc\ ccbb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ ccbb}_{u'}u''b}^{\langle 1\rangle}\ \langle 0\rangle\ u^5 \tag{10}$$

$$\vdash \qquad \overbrace{\underbrace{bcc\ cbcc\ cbcc\ ccbb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ ccbb}_{\substack{u'\\ [1]}}u''b}^{\substack{\langle 1\rangle\\ [4]}}\ \langle 0\rangle\ u^6 \tag{11}$$

In Equation (11) above we have finished reading the leftmost $\langle 1\rangle$ in the dataword and completed our simulation of the computation step $(\mathbf{0}, 01 \quad 11 \vdash 0, \mathbf{01} \quad 10)$. The word $\langle 0\rangle\, u^6$ was appended simulating appendant $\alpha_0 = 0$ from the program $\mathcal{C} = 0, 01$ was appended. From Section 3.1 the u subwords in u^6 are considered garbage objects and these u subwords have no effect on the computation (see first paragraph of Section 3.2). Also, in Equation (11) we see that the next $\langle 1\rangle$ is entered with shift 4 which encodes that second appendant $\alpha_1 = 01$ in the program $\mathcal{C} = 0, 01$ is now marked. To see this recall that entering an object with a shift of $(\beta - 4m) \bmod \beta$ encodes that appendant α_m is marked and since $\beta = 8$ we get a shift of 4 when $m = 1$ meaning α_1 is marked. The dataword in Equation (11) is of the form given in Equation (1) and is ready to begin simulation of the next computations step.

4 The Post Correspondence Problem for 5 Pairs of Words

In Theorem 11 we reduce the halting problem for the binary tag system given in Lemma 9 to the Post correspondence problem for 5 pairs of words.

▶ **Definition 10** (Post correspondence problem)**.** Given a set of pairs of words $\{(r_i, v_i) | r_i, v_i \in \Sigma^*, i \in \{0, 1, 2 \ldots, n\}\}$ where Σ is a finite alphabet, determine whether or not there exists a non-empty sequence $i_1, i_2, \ldots i_l, \in \{0, 1, 2 \ldots, n\}$ such that $r_{i_1} r_{i_2} \ldots r_{i_l} = v_{i_1} v_{i_2} \ldots v_{i_l}$.

▶ **Theorem 11.** *The Post correspondence problem is undecidable for 5 pairs of words.*

Proof. We reduce the halting problem for the binary tag systems $\mathcal{T}'_{\mathcal{C}}$ in Lemma 9 to the Post correspondence problem instance given by the 5 pairs of words

$$\mathcal{P} = \{(1,\ 1\langle u_0\rangle\langle u_1\rangle \ldots \langle u_l\rangle 10),\ (10^\beta 1,\ 110),\ (10^\beta 1, 0),\ (1, 0),\ (10^\beta 1111, 1111)\}$$

where ϵ is the empty word and $u_i \in \{b, c\}$. The symbols b and c in $\mathcal{T}'_{\mathcal{C}}$ are encoded as $\langle b\rangle = 10^\beta 1$ and $\langle c\rangle = 1$ respectively, where β is the deletion number of $\mathcal{T}'_{\mathcal{C}}$. Let $r = r_{i_1} r_{i_2} \ldots r_{i_l}$ and $v = v_{i_1} v_{i_2} \ldots v_{i_l}$, where each $(r_i, v_i) \in \mathcal{P}$ and r is a prefix of v. We will call the pair (r, v) a configuration of $\mathcal{P}$. An arbitrary dataword $x_0 x_1 \ldots x_q b \in \{b, c\}^* b$ is encoded by a $\mathcal{P}$ configuration of the form

$$(r, v) = (r,\ r\langle x_0\rangle\langle x_1\rangle \ldots \langle x_q\rangle 10^\beta) \tag{12}$$

In each configuration (r, v), the unmatched part of v (i.e. $\langle x_0\rangle\langle x_1\rangle \ldots \langle x_q\rangle 10^\beta$) encodes the current dataword of $\mathcal{T}'_{\mathcal{C}}$.

Starting from the pair $(1,\ 1\langle u_0\rangle\langle u_1\rangle \ldots \langle u_l\rangle 10)$, if $u_0 = c$ we add the pair $(1, 0)$ and this matches $\langle c\rangle = 1$ simulating the deletion of u_0. If, on the other hand, $u_0 = b$ we add the pair $(10^\beta 1, 0)$ and this matches $\langle b\rangle = 10^\beta 1$ simulating the deletion of u_0. So after matching $\langle u_0\rangle$ we have $(1\langle u_0\rangle,\ 1\langle u_0\rangle\langle u_1\rangle \ldots \langle u_l\rangle 100)$. We match $\beta - 1$ encoded $\mathcal{T}'_{\mathcal{C}}$ symbols in this way to give $(r, v) = (1\langle u_0\rangle \ldots \langle u_{\beta-2}\rangle,\ 1\langle u_1\rangle\langle u_2\rangle \ldots \langle u_l\rangle 10^\beta)$. The configuration is now of the form given in Equation (12) and the unmatched sequence $\langle u_{\beta-1}\rangle \ldots \langle u_l\rangle 10^\beta$ in v encodes the input dataword to $\mathcal{T}'_{\mathcal{C}}$ in Lemma 9.

A step of $\mathcal{T}'_{\mathcal{C}}$ on an arbitrary dataword $x_0 x_1 \ldots x_q b$ is of one the two forms:

$$cx_1 \ldots x_q b \quad \vdash \quad x_{\beta-1} \ldots x_q b u_1 \ldots u_l b \tag{13}$$

$$bx_1 \ldots x_q b \quad \vdash \quad x_{\beta-1} \ldots x_q b b \tag{14}$$

These computation steps are simulated as follows: In Equation (12), if $x_0 = c$ then $\langle x_0\rangle = 1$ and we add the pair $(1,\ 1\langle u_0\rangle\langle u_1\rangle \ldots \langle u_l\rangle 10)$ to simulate the $\mathcal{T}'_{\mathcal{C}}$ rule $c \to u_0 \ldots u_l b$, and

this gives $(r1,\ r1\langle x_1\rangle\dots\langle x_q\rangle 10^\beta 1\langle u_0\rangle\langle u_1\rangle\dots\langle u_l\rangle 10)$. In Equation (12), if $x_0 = b$ then $\langle x_0\rangle = 10^\beta 1$ and we add the pair $(10^\beta 1, 110)$ to simulate the $\mathcal{T}'_\mathcal{C}$ rule $b \to b$, giving $(r10^\beta 1,\ r10^\beta 1\langle x_1\rangle\dots\langle x_q\rangle 10^\beta 110)$. In both cases we simulate the deletion of a further $\beta - 1$ tag system symbols as we did in the previous paragraph to complete the simulated computation step. So if $x_0 = c$ this gives $(r1\langle x_1\rangle\dots\langle x_{\beta-1}\rangle,\ r1\langle x_1\rangle\dots\langle x_q\rangle 10^\beta 1\langle u_0\rangle\langle u_1\rangle\dots\langle u_l\rangle 10^\beta)$, with the unmatched part in this pair encoding the new dataword on the right of Equation (13). Or, if $x_0 = b$ we get $(r10^\beta 1\langle x_1\rangle\dots\langle x_{\beta-1}\rangle,\ r10^\beta 1\langle x_1\rangle\dots\langle x_q\rangle 10^\beta 110^\beta)$, with the unmatched part in this pair encoding the new dataword on the right of Equation (14). The simulated computation step for both cases is now complete.

Now we show that $\mathcal{P}$ simulates $\mathcal{T}'_\mathcal{C}$ halting with a matching pair of words. Recall that $|u| = s\beta$ where we can choose s to be any natural number greater than some constant (see Section 3.1). We choose s to be of the form $s = x(\beta - 1) + 1$ and so the input dataword $u_{\beta-1}\dots u_l b$ to $\mathcal{T}'_\mathcal{C}$ has length $x\beta(\beta-1)+1$ and the rules either increase its length by $x\beta(\beta-1)$ (rule $c \to u_0\dots u_l b$) or decrease it by $\beta - 1$ (rule $b \to b$), which means all datawords of $\mathcal{T}'_\mathcal{C}$ have length $y(\beta - 1) + 1$, where $y \in \mathbb{N}$. From Lemma 9, $\mathcal{T}'_\mathcal{C}$ halts when the length of its dataword (which has the form b^*) is less than the deletion number β. So, when $\mathcal{T}'_\mathcal{C}$ halts we have $y(\beta - 1) + 1 < \beta$ which means the dataword is a single b. From Equation (12), this is encoded as the configuration $(r, v) = (r, r10^\beta)$. By appending the pair $(10^\beta 1111, 1111)$ to (r, v), we get the matching pair $(r10^\beta 1111, r10^\beta 1111)$ when $\mathcal{T}'_\mathcal{C}$ halts.

To complete our proof we show that choices that do not follow the simulation as described above leads to a mismatch. We must have $(1,\ 1\langle u_0\rangle\langle u_1\rangle\dots\langle u_l\rangle 10)$ as the leftmost pair as putting any other pair from $\mathcal{P}$ on the left will not give a match. Now recall that the pair $(1,\ 1\langle u_0\rangle\langle u_1\rangle\dots\langle u_l\rangle 10)$ encodes the initial configuration (or input) for $\mathcal{T}'_\mathcal{C}$. So now we show that given the encoding of an arbitrary configuration any choice that does not follow the simulation leads to a mismatch. From Equation (12) an arbitrary configuration has the form $(r,\ r\langle x_0\rangle\langle x_1\rangle\dots\langle x_q\rangle 10^\beta)$. If $x_0 = b$ then $\langle x_0\rangle = 10^\beta 1$ and we cannot choose either of the pairs $(1,\ 1\langle u_0\rangle\langle u_1\rangle\dots\langle u_l\rangle 10)$ or $(1, 0)$ as they will allow no further matches, the pair $(10^\beta 1111, 1111)$ cannot be chosen as four consecutive 1s do not appear in the encoding (this is because in u we cannot have 2 encoded c symbols ($\langle c\rangle = 1$) next to each other). The pair $(10^\beta 1, 0)$ appends a 0 onto the right sequence to give $10^{\beta+1}$ which cannot be matched as it is only possible to match 0 sequences of the from $10^\beta 1$. Similar arguments are used for the case of $x_0 = c$ and so we do not repeat them. After we have matched $\langle x_0\rangle$ our simulation algorithm requires that we simulate the deletion of a further $\beta - 1$ symbols. If we deviate from the simulation either we find almost immediately that no more matches are possible or we end up with a sequence of zeros that does not have the form $10^\beta 1$ and so cannot be matched. After simulating the deletion of $\beta - 1$ symbols we have completed the simulation of an arbitrary computation step and arrived at an encoded configuration of the form given by Equation (12). So it is not possible to find a match in $\mathcal{P}$ if we do not follow the simulation described above. Therefore, $\mathcal{P}$ has a matching sequence if and only if $\mathcal{T}'_\mathcal{C}$ halts. ◀

By applying the reductions in [8] and [3] to $\mathcal{P}$ in Theorem 11 we get Corollary 12.

▶ **Corollary 12.** *The matrix mortality problem is undecidable for sets with six* 3×3 *matrices and for sets with two* 18×18 *matrices.*

Acknowledgements: This work is supported by Swiss National Science Foundation grant number 200021-141029. I would like to thank Matthew Cook, Damien Woods, Vesa Halava, and Mika Hirvensalo for their comments and discussions.

References

1 Vincent D. Blondel and John N. Tsitsiklis. When is a pair of matrices mortal? *Information Processing Letters*, 63(5):283–286, 1997.

2 Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

3 Julien Cassaigne and Juhani Karhumäki. Examples of undecidable problems for 2-generator matrix semigroup. *TCS*, 204(1-2):29–34, 1998.

4 Volker Claus. Some remarks on PCP(k) and related problems. *Bull. EATCS*, 12:54–61, 1980.

5 John Cocke and Marvin Minsky. Universality of tag systems with $P = 2$. *Journal of the ACM*, 11(1):15–20, 1964.

6 Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

7 Andrzej Ehrenfeucht, Juhani Karhumäki, and Grzegorz Rozenberg. The (generalized) Post correspondence problem with lists consisting of two words is decidable. *TCS*, 21(2):119–144, 1982.

8 Vesa Halava and Tero Harju. Mortality in matrix semigroups. *American Mathematical Monthly*, 108(7):649–653, 2001.

9 Vesa Halava, Tero Harju, and Mika Hirvensalo. Undecidability bounds for integer matrices using Claus instances. *IJFCS*, 18(5):931–948, 2007.

10 Tero Harju and Maurice Margenstern. Splicing systems for universal Turing machines. In *DNA 10*, volume 3384 of *LNCS*, pages 149–158. Springer, 2005.

11 Kristian Lindgren and Mats G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.

12 Yuri Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. *TCS*, 330(1):145–169. (An earlier version appeared in "11th Annual IEEE Symposium on Logic in Computer Science, LICS 1996".), 2005.

13 Marvin Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

14 Marvin Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, volume 5, pages 229–238, Provelence, 1962. AMS.

15 Turlough Neary and Damien Woods. P-completeness of cellular automaton Rule 110. In *ICALP 2006, Part I*, volume 4051 of *LNCS*, pages 132–143. Springer, 2006.

16 Michael S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.

17 Emil L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943.

18 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of The American Mathematical Society*, 52:264–268, 1946.

19 Raphael M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.

20 Yurii Rogozhin. Small universal Turing machines. *TCS*, 168(2):215–240, 1996.

21 Paul Rothemund. A DNA and restriction enzyme implementation of Turing machines. In *DNA Based Computers*, volume 27 of *DIMACS*, pages 75–119. AMS, 1996.

22 Hao Wang. Tag systems and lag systems. *Mathematical Annals*, 152:65–74, 1963.

23 Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *47th Annual IEEE Symposium on Foundations of Computer Science*, pages 439–448, 2006.