The complexity of regex crosswords*

Stephen Fenner[†1], Daniel Padé[‡1], and Thomas Thierauf[§2]

[1]University of South Carolina, Columbia, SC USA
[2]Aalen University, Aalen, Germany

August 13, 2019

Abstract

In a typical regex crossword puzzle, one is given two non-empty lists $\langle R_1, \ldots, R_m \rangle$ and $\langle C_1, \ldots, C_n \rangle$ of regular expressions (regexes) over some alphabet, and the challenge is to fill in an $m \times n$ grid of characters such that the string formed by the i^{th} row is in $L(R_i)$ and the string formed by the j^{th} column is in $L(C_j)$, for all $1 \le i \le m$ and $1 \le j \le n$. We consider a restriction of this puzzle where all the R_i are equal to one another and similarly the C_j. We consider a 2-player version of this puzzle, showing it to be **PSPACE**-complete. Using a reduction from 3SAT, we also give a new, simple proof of the known result that the existence problem of a solution for the restricted (1-player) puzzle is **NP**-complete.

Keywords: Complexity; Regular expressions; Regex crossword; Picture language; NP-complete; PSPACE-complete

1 Introduction

Regular expression crossword puzzles (regex crosswords, for short) share some traits in common with traditional crossword puzzles and with sudoku. One is typically given two lists $R_1, \ldots, R_m$ and $C_1, \ldots, C_n$ of regular expressions labeling the rows and columns, respectively, of an $m \times n$ grid of blank squares. The object is to fill in each square with a letter so that each row, read left to right as a string, *matches* (i.e., is in the language denoted by) the corresponding regular expression, and similarly for each column, read top to bottom. The solution itself may have some additional property, e.g., spelling out a phrase or sentence in row major order or providing a clue to another puzzle.

Regex crosswords have enjoyed some recent popularity, having been discussed in several popular media sources [3, 5], and thanks to a website where people can solve the puzzles online [1]. Some variants of the basic puzzle have also been posed [2].

A natural complexity theoretic question to ask is: How hard is it to solve a regex crossword in general? The folklore answer—easy to show and apparently found by several people independently[1]—is that it is **NP**-hard, and the corresponding decision problem ("Does a solution exist?") is **NP**-complete.

*Journal version of the conference paper [7]

†Email: `fenner.sa@gmail.com`

‡Email: `djpade@gmail.com`

§Email: `thomas.thierauf@htw-aalen.de`

[1]Glen Takahashi posted this question to Stack Exchange in 2012 [14], but it has been asked by others independently. That post includes an anonymous proof ("FrankW").

In this paper, we consider a number of variations on the basic regex crossword puzzle: (1) a restriction of the puzzle where all the row regexes $R_1, \ldots, R_m$ are equal and all the column regexes $C_1, \ldots, C_n$ are equal; (2) a 2-player game where players take turns attempting to fill in successive rows and columns of the grid; (3) a further restriction of the puzzle where all row and column regexes are equal (to each other); (4) restriction to regexes over a binary alphabet; (5) unbounded and semi-bounded versions of the puzzle; (6) versions with string-valued entries in each cell. Variation (2) can also be restricted to having equal row regexes and equal column regexes for the two players. These variants have corresponding decision problems: Let RC be the solution existence problem for variation (1), RCG' the first-player-win problem for variation (2), and RCG the first-player-win problem for the restricted version of (2) (see Sections 2.3 and 4 for precise definitions). One of our main results is that RCG' and RCG are both **PSPACE**-complete (see Section 4, below). We give explicit polynomial reductions from TQBF to RCG' and from RCG' to RCG.

The **NP**-completeness of RC was shown in [6], but the polynomial reduction used there was indirect and needlessly general for this particular hardness result. As a warm-up to our main result on games, we give a simple, straightforward polynomial reduction from $\mathsf{3SAT}$ to RC.

We also give general techniques for transforming a general regex crossword problem into an equivalent, more restricted problem: (1) transforming a regex crossword problem over an arbitrary alphabet to one over a binary alphabet; (2) transforming an (R, C)-crossword problem into an (E, E)-crossword problem (i.e., one where the row and column regexes are all equal to each other. We use these techniques to strengthen a variety of hardness results.

As with the Post Correspondence Problem in computability, our results have the pedagogical benefit of showing the hardness of some decision problems in automata theory that are simply stated and accessible to any undergraduate theory student. The proofs given here are similarly accessible.

1.1 Connections to other work

Regex crossword techniques bear some similarity to results in cellular automata, to the Cook-Levin theorem, and to results of Berger from the 1960s showing the undecidability of tiling the plane with Wang tiles (the so-called "domino problem" [4], which was the first proof that there exist finite tile sets that tile the whole plane but only aperiodically).

The particular problems we study here are perhaps chiefly inspired by results in the theory of two-dimensional languages (picture languages) from formal language theory [9]. Given two regexes R and C for the rows and columns, respectively, the *unbounded* (R, C)-crossword problem asks whether a solution grid exists of *any size*. One can show that the recognizable picture languages coincide exactly with the letter-to-letter projections of (R, C)-crossword solutions [9, Theorem 8.6] (except that the empty picture may also be included in the language). Recognizable picture languages can be defined in terms of finite objects known as tiling systems [8] (cf. [9, Definition 7.2]), and given a tiling system $\mathcal{T}$, it is not hard to show that one can effectively find two regular expressions R and C (over some alphabet) and a projection π that defines the same picture language as $\mathcal{T}$ (see [9, Theorem 8.6]). The existence problem for recognizable picture languages ("Given a tiling system, does it define a nonempty language?") is known to be undecidable ([9, Theorem 9.1]), and so, putting these results together, we get that the solution existence problem for unbounded (R, C)-crosswords is undecidable as well. A much more direct reduction from the halting problem to unbounded (R, C)-crosswords is given in Section 5, and it is shown in Section 6 that one could even fix the column regex C once and for all, as well as restricting R and C to be over a binary alphabet.[2]

[2]These latter results first appeared in [6].

The unbounded regex crossword problem naturally assumes one regex R for all rows and one regex C for all columns, since the number of rows and columns is unspecified. This directly motivates us to impose similar restrictions on the bounded regex crossword problems we study here, where the dimensions of the grid are given as part of the input.

We give some basic concepts and definitions in Section 2. Section 3 gives our polynomial reduction from 3SAT to RC. This reduction suggests the technique we use to show our main results about 2-player crossword games in Section 4. Section 3.3 gives a simple proof of **NP**-completeness when the row and column regexes are equal to each other, i.e., when there is a single regex for all rows and columns. Section 6 describes the two techniques for transforming regex crossword problems into more restricted equivalent ones and uses these techniques to obtain stronger hardness results. Section 7 describes a number of complexity-theoretic corollaries of our main results. We give open problems in Section 8.

2 Preliminaries

Our conventions regarding regular expressions are fairly standard, conforming to the conventions in Sipser [12] or Hopcroft, Motwani, & Ullman [10] for the most part.

Given an alphabet Σ, regular expressions (*regexes* for short) over Σ are constructed in the usual way from $\emptyset$ and single symbols from Σ (the *atomic* regexes) using the operators $\cup$, $\|$, and $*$, where $\|$ or juxtaposition both indicate concatenation, and $*$ is the Kleene star operator (see, for example, Sipser [12]). For syntactic grouping, the $\cup$ operator has lowest precedence, followed by concatenation, followed by the $*$ operator (highest precedence). Parentheses are used freely to force arbitrary grouping as usual. Given a regex R and a string w over Σ, we say that R *matches* w (or, w *matches* R) to mean that w is in the language $L(R)$ denoted by R (see the next paragraph for exact rules). If there is a possibility of confusion, we sometimes distinguish symbols from Σ with their corresponding atomic regexes by showing the latter in boldface; for example, if symbol a is in Σ, then $\mathbf{a}$ is the atomic regex that matches the length-1 string "a" and nothing else. If there is no possibility of confusion, then we identify a string with the regex that matches it and nothing else.

We now briefly review the meaning of a regex over an alphabet Σ, i.e., the strings that it matches, via recursive syntactic rules. We mostly follow the conventions in [12].

- The regex $\emptyset$ matches no strings.
- For any $a \in \Sigma$, the regex $\mathbf{a}$ matches the string "a" (of length 1) and nothing else.
- Given regexes r and s, the regex $r \cup s$ matches exactly those strings that match r or s.
- Given regexes r and s, the regex rs (or $r\|s$) matches exactly those strings that are formed by concatenating a string matching r followed by a string matching s.
- Given regex r, the regex r^* matches exactly those strings that are concatenations of any finite number (zero or more) strings, each of which matches r.

Note that r^* always matches the empty string (of length 0), regardless of r. We let "ε" denote both the empty string and the regex $\emptyset^*$, which matches the empty string and nothing else. Which meaning is used will be clear from the context.

We will assume two common additional regex operators defined in terms of the primitive ones above: for regexes r and s,

- r^+ denotes rr^*,
- $r?$ denotes $r \cup \varepsilon$, and
- $r \cap s$ (intersection) denotes the regex (obtained from r and s in some standard effective way) that matches those strings that are matched by both r and s simultaneously.[3]

We assume throughout the paper that all alphabets contain 0 and 1 at least. (Results are trivial for unary alphabets.) For the **NP**-completeness result of Section 2.3, one can assume the alphabet $\{0, 1\}$. For the **PSPACE**-completeness result of Section 4, it suffices that the alphabet be $\{0, 1, 2\}$.

2.1 3SAT

An instance of 3SAT is a Boolean formula φ over k variables $x_1, \ldots, x_k$ in conjunctive normal form:

$$\varphi \coloneqq C_i \wedge \cdots \wedge C_d$$

where each clause C_i is a disjunction of three literals (each a variable or its negation):

$$C_i \coloneqq \ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}$$

The question is whether φ is true for some assignment of the variables (i.e., it is *satisfied*). This is one of the canonical complete problems for **NP**. In Section 2.3 we show that the language RC — the language of regex crosswords — is **NP**-complete by giving a polynomial reduction from 3SAT.

2.2 TQBF

An instance of TQBF is described by a closed Boolean formula φ, given in prenex normal form:

$$\varphi \coloneqq \exists x_1 \forall y_1 \cdots \exists x_k \forall y_k \ \tilde{\varphi}(x_1, y_1, \ldots, x_k, y_k) \tag{1}$$

where $\tilde{\varphi}$ is a quantifier-free Boolean formula which can be assumed to be in conjunctive normal form with c clauses and $2k$ variables, for some positive c and k. Here, the quantifiers alternate, starting with $\exists$, with variables $x_1, \ldots, x_k$ being existentially quantified and $y_1, \ldots, y_k$ universally quantified.

The sentence φ is naturally viewed as a two-player game, where the players alternate choosing truth values for the variables in order, the first player wishing to make the formula $\tilde{\varphi}$ true and second player wishing to make it false. The question to be answered is whether φ is true when the quantified variables range over the Boolean values FALSE and TRUE.[4] That is, whether the first player has a winning strategy in the corresponding game.

As 3SAT is for **NP**, TQBF is the canonical complete problem for **PSPACE**. In Section 4, we show that RCG — the language of regex crossword games (defined below) with a winning strategy for the first player — is **PSPACE**-complete by reduction from TQBF.

[3] The resulting regex may have size exponential in that of r and s.

[4] More precisely, the question is whether the sentence $\exists x_1 \forall y_1 \cdots \exists x_k \forall y_k [\tilde{\varphi}(x_1, y_1, \ldots, x_k, y_k) = \text{TRUE}]$ holds in the two-element Boolean algebra $(\{\text{FALSE}, \text{TRUE}\}, \wedge, \vee, \neg)$.

2.3 $(\boldsymbol{R},\boldsymbol{C})$-crosswords

Given an alphabet Σ, an (R,C)-*crossword over* Σ (or just an (R,C)-*crossword* if Σ is assumed) is a 4-tuple $\langle 0^m, 0^n, R, C\rangle$ where m and n are positive integers (the number of *rows* and *columns*, respectively) represented in unary, and R and C are regexes over Σ.

Definition 2.1. Given an alphabet Σ, a Σ-*grid* is a two-dimensional array of symbols from Σ. If G is a Σ-grid with m rows and n columns, then we say that G is $m \times n$.

A *solution* to an (R,C)-crossword $\langle 0^m, 0^n, R, C\rangle$ is an $m \times n$ Σ-grid such that, interpreting rows and columns as strings, each row, read left to right, matches R and each column, read top to bottom, matches C. We call an (R,C)-crossword *solvable* if it has a solution, and we call it *uniquely solvable* if it has exactly one solution.

3 An NP-Completeness Proof for $(\boldsymbol{R},\boldsymbol{C})$-Crossword Solvability

Definition 3.1. For alphabet Σ, the language RC_Σ is the set of all solvable (R,C)-crosswords over Σ. We drop the subscript if Σ is clear from the context.

Theorem 3.2 ([6])**.** $\mathsf{RC}_{\{0,1\}}$ *is* **NP**-*complete.*

Theorem 3.2 was shown in [6] via an indirect, complicated reduction. In this section, we give a much more straightforward polynomial reduction from $\mathsf{3SAT}$ to $\mathsf{RC}_{\{0,1\}}$.

We assume the alphabet $\Sigma := \{0,1\}$ for this section, letting RC denote $\mathsf{RC}_{\{0,1\}}$. RC is in **NP**; this easily follows from the fact that deciding whether a given string matches a given regex is decidable in polynomial time (and the fact that the dimensions of the grid are given in unary). It remains to show that RC is **NP**-hard. Theorem 3.2 is then an immediate corollary of the following technical lemma, which we prove in subsections 3.1 and 3.2. Lemma 3.3 is stronger than needed for Theorem 3.2 as it describes the number of solutions to the crossword.

Lemma 3.3. *There exists a polynomial-time computable function f such that, given any Boolean formula φ as defined in Section 2.1 with $k \geq 1$ variables and $d \geq 2$ clauses, $f(\varphi)$ is an instance $\langle 0^{d+1}, 0^{d+k}, R, C\rangle$ of RC such that, if $s \geq 0$ is the number of satisfying assignments of φ, then $f(\varphi)$ has exactly $d!s$ many solutions.*

Given φ as in the lemma, we define the function f as follows: For $1 \leq i \leq d$, we define t_i to be the regex

$$t_i = \mathbf{0}^{i-1}\mathbf{1}\mathbf{0}^{d-i} = \underbrace{\mathbf{0}\cdots\mathbf{0}}_{i-1}\mathbf{1}\underbrace{\mathbf{0}\cdots\mathbf{0}}_{d-i}\ . \tag{2}$$

Then we define

$$S := \mathbf{1}^d\mathbf{0}^* \tag{3}$$

$$R := \left(\bigcup_{i=1}^{d} t_i R_i\right) \cup S \tag{4}$$

$$C := \mathbf{1}\,(\mathbf{0}^*\mathbf{1}\mathbf{0}^*) \cup \mathbf{0}(\mathbf{0}^* \cup \mathbf{1}^*) \tag{5}$$

where S is called the 'spine,' and for $1 \leq i \leq d$, R_i is derived from the formula φ as follows:

$$R_i := (a_{i,1}\cdots a_{i,k}) \cup (b_{i,1}\cdots b_{i,k}) \cup (c_{i,1}\cdots c_{i,k}) \tag{6}$$

where, for $1 \leq j \leq k$,

$$a_{i,j} = \begin{cases} \mathbf{1} & \text{if the first literal in the } i^{\text{th}} \text{ clause is } x_j \\ \mathbf{0} & \text{if the first literal in the } i^{\text{th}} \text{ clause is } \overline{x_j} \\ (\mathbf{1} \cup \mathbf{0}) & \text{otherwise} \end{cases} \tag{7}$$

and $b_{i,j},\ c_{i,j}$ are set similarly according to the second and third literals in each clause. Finally, we define

$$f(\varphi) := \langle 0^{d+1}, 0^{d+k}, R, C \rangle \ .$$

The function f is evidently polynomial-time computable.

Let $s \geq 0$ be the number of satisfying assignments to φ.

3.1 $f(\varphi)$ has at least $d!s$ many solutions

Let $\langle z_1, \ldots, z_k \rangle$ be any satisfying assignment to φ. This sets up a $d+1$ by $d+k$ crossword solution of the following form:

$$\begin{array}{c|ccccc|ccc}
 & c_1 & c_2 & c_3 & \cdots & c_d & c_{d+1} & \cdots & c_{d+k} \\
\hline
r_0 & 1 & 1 & 1 & \cdots & 1 & 0 & \cdots & 0 \\
\hline
r_1 & 1 & 0 & 0 & \cdots & 0 & z_1 & \cdots & z_k \\
r_2 & 0 & 1 & 0 & \cdots & 0 & z_1 & \cdots & z_k \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & z_1 & \cdots & z_k \\
r_d & 0 & 0 & 0 & \cdots & 1 & z_1 & \cdots & z_k
\end{array}$$

Figure 1: *Solution*

Here, the first row is the spine (matching S); the block on the left below the spine is akin to an identity matrix; and the block on the right consists of columns where each column is either all 1's or all 0's (save the first element, which is always 0), according to each z_i. An overview representation is shown below:

<table>
<tr><td colspan="2">Spine</td></tr>
<tr><td>Calibration
Region</td><td>Clause
Verification</td></tr>
</table>

Where the spine is the string that matches S. The 'clause verification region' is determined by the satisfying assignment to φ, i.e., if z_j is true in the satsifying assignment, then column c_{d+j} will match the regex $\mathbf{01}^*$; otherwise it will match $\mathbf{00}^*$.

By construction, it is then clear that $f(\varphi)$ is solvable. In other words, there is a way to fill in the grid such that all rows match the regex R, and all columns match the regex C.

In fact, since the calibration region requires only one 1 per row and column, the solution given in table 1 is not the only valid one. It is easy to see that once any solution is given, any permutation of the (non-spine) rows gives another (distinct) valid solution. Thus we get $d!$ many distinct solutions for every satisfying assignment $\langle z_1, \ldots, z_k \rangle$. To see then that $f(\varphi)$ has at least $d!s$ many solutions, we only need to observe than any two solutions corresponding to different satisfying assignments must also differ (somewhere in their last k columns).

3.2 $f(\varphi)$ has at most $d!s$ many solutions

To complete the proof of Lemma 3.3, we show that every solution of $f(\varphi) = \langle 0^{d+1}, 0^{d+k}, R, C\rangle$ corresponds to a satisfying assignment to φ, and further, every satisfying assignment of φ corresponds to at most $d!$ many solutions to $f(\varphi)$. We do this via a series of claims.

Suppose $f(\varphi)$ has a solution with rows $r_0, \ldots, r_d$ and columns $c_1, \ldots, c_{d+k}$.

Observe that since each r_j matches R, it must either start with d many 1's or else have exactly one 1 among its first d symbols (recalling that we are assuming $d \geq 2$).

Claim 3.4. The string r_0 matches S.

Proof. Assume not. Then r_0 must match $t_i R_i$ for some $1 \leq i \leq d$. Fix such an i. The picture below shows the case where r_0 matches $t_2 R_2$, i.e., $r_0 = 010\cdots$:

$$\begin{array}{c|cccccc|cc}
 & c_1 & c_2 & c_3 & c_4 & \cdot & c_d & \cdot & \cdot \\
\hline
r_0 & 0 & 1 & 0 & 0 & \cdot & 0 & & \\
\hline
\vdots & & & & & & & &
\end{array}$$

From the definition of C, we see that c_i must match $\mathbf{1}(\mathbf{0}^*\mathbf{1}\mathbf{0}^*)$, that is, $c_i = 10^{j-1}10^{d-j}$ for some $1 \leq j \leq d$. The picture below shows the case where $i = 2$ and $j = 2$, that is, where $c_i = c_2 = 10100\cdots 0$:

$$\begin{array}{c|cccccc|cc}
 & c_1 & c_2 & c_3 & c_4 & \cdot & c_d & \cdot & \cdot \\
\hline
r_0 & 0 & 1 & 0 & 0 & \cdot & 0 & & \\
\hline
r_1 & & 0 & & & & & & \\
r_2 & & 1 & & & & & & \\
r_3 & & 0 & & & & & & \\
\vdots & & \vdots & & & & & &
\end{array}$$

For r_j, we have two cases, both leading to contradiction:

r_j **matches** S**:** This requires that all of the first d columns other than c_i match $\mathbf{01}^*$, which means $r_{j'}$ starts with $1^{i-1}01^{d-i}\cdots$ for all $j' \geq 1$ such that $j' \neq j$. These rows do not match R, and there is at least one of them, since $d \geq 2$.

r_j **matches** $t_i R_i$**, that is,** $r_j = 0^{i-1}10^{d-i}\cdots$**:** This requires that all of the first d columns other than c_i match $\mathbf{0}^*$, which means no rows other than r_j and r_0 will match R, since they all start with 0^d. Again, there is at least one such row because $d \geq 2$.

This proves the claim. $\square$

By Claim 3.4, the first d columns must match $\mathbf{1}(\mathbf{0}^*\mathbf{1}\mathbf{0}^*)$; that is, ignoring the spine, there is exactly one 1 in each of the first d columns. We call such columns *calibration columns.*

Claim 3.5. *No row other than r_0 matches S.*

Proof. Again assume this not the case. By the previous claim, r_0 must match S. Suppose r_j also matches S for some $j \geq 1$. Then C forces $r_{j'}$ to start with d many 0's for all $1 \leq j' \neq j$, because the calibration columns are only allowed a single 1 below the spine. Thus none of these $r_{j'}$ matches R, and there is at least one of them, since $d \geq 2$. □

Claim 3.6. *For every i, $1 \leq i \leq d$, there exists a row that matches $t_i R_i$.*

Proof. By Claims 3.4 & 3.5, we have that r_0 is the only row to match the spine S. Since $R = \left(\bigcup_{i=1}^{d} t_i R_i\right) \cup S$, it follows that each of the other rows matches $t_i R_i$ for some i. For the purposes of contradiction, assume that there is some $t_i R_i$ not matched by any row. Then by the pigeonhole principle, there must be two distinct rows r_n and r_m both matching $t_\ell R_\ell$ for the same ℓ. By the definition of t_ℓ, the column c_ℓ will thus have at least two 1's below the spine:

$$
\begin{array}{c|ccccccc|cc}
 & c_1 & \cdot & c_{\ell-1} & c_\ell & c_{\ell+1} & \cdot & c_d & c_{d+1} & \cdot \\
\hline
r_0 & 1 & \cdot & 1 & 1 & 1 & \cdot & 1 & \cdot & \\
\hline
\vdots & & & & \vdots & & & & & \\
r_n & 0 & \cdot & 0 & 1 & 0 & \cdot & 0 & \cdot & \\
\vdots & & & & \vdots & & & & & \\
r_m & 0 & \cdot & 0 & 1 & 0 & \cdot & 0 & \cdot & \\
\vdots & & & & & & & & &
\end{array}
$$

But then column c_ℓ does not match C. This completes the proof. □

Claim 3.7. *φ is satisfiable.*

Proof. Because of the spine in the first row, note that for $1 \leq j \leq k$, c_{d+j} matches either $\mathbf{01}^*$ or $\mathbf{00}^*$. Set

$$z_j := \begin{cases} 1 & \text{if } c_{d+j} \text{ matches } \mathbf{01}^*, \\ 0 & \text{if } c_{d+j} \text{ matches } \mathbf{00}^*. \end{cases}$$

We show that $\langle z_1, \ldots, z_k \rangle$ is a satisfying truth assignment for φ. Consider the i^{th} clause C_i of φ. By Claim 3.6, some non-spine row matches $t_i R_i$. Let r be the suffix of that row obtained by removing its first d symbols. Then r matches either $a_{i,1} \cdots a_{i,k}$, $b_{i,1} \cdots b_{i,k}$, or $c_{i,1} \cdots c_{i,k}$. Suppose r matches $a_{i,1} \cdots a_{i,k}$ (the other two cases are handled similarly). Let x_j be the variable mentioned by the first literal $\ell_{i,1}$ of C_i. If $\ell_{i,1} = x_j$, then $a_{i,j} = \mathbf{1}$, whence r has a 1 as its j^{th} symbol, whence c_{d+j} matches $\mathbf{01}^*$, whence $z_j = 1$, which makes $\ell_{i,1}$ true, satisfying C_i. Similarly, if $\ell_{i,1} = \overline{x_j}$, then $z_j = 0$, also satisfying C_i.

Since i was arbitrary, we have that φ is satisfied by $\langle z_1, \ldots, z_k \rangle$. □

To finish the proof of Lemma 3.3, we make one more claim. Let h be the map that takes a solution to $f(\varphi)$ and outputs the corresponding satisfying assignment to φ as defined in the proof of Claim 3.7.

Claim 3.8. *Each satisfying assignment of φ has at most $d!$ many pre-images under the map h.*

Proof. Given $a := \langle z_1, \ldots, z_k \rangle$ satisfying φ, any pre-image of a under h has its last k columns equal to $0z_1^d, \ldots, 0z_k^d$, respectively, and the $d \times d$ subgrid consisting of its first d columns below the spine must be a permutation matrix. There are only $d!$ many such matrices, and each determines the entire solution corresponding to a via h. □

This concludes the proof of Lemma 3.3, from which Theorem 3.2 follows immediately.

Remark. Notice that the column regex $C := \mathbf{1}\,(\mathbf{0}^*\mathbf{1}\mathbf{0}^*) \cup \mathbf{0}(\mathbf{0}^* \cup \mathbf{1}^*)$ that we defined above does not depend on the input formula φ at all. Thus we get the following modest improvement over Theorem 3.2:

Definition 3.9. Given regex C over $\{0,1\}$, define the language

$$\mathsf{RC}(C) := \{\langle 0^m, 0^n, R\rangle \mid \langle 0^m, 0^n, R, C\rangle \in \mathsf{RC}\} .$$

Proposition 3.10. $\mathsf{RC}\left(\mathbf{1}(\mathbf{0}^*\mathbf{1}\mathbf{0}^*) \cup \mathbf{0}(\mathbf{0}^* \cup \mathbf{1}^*)\right)$ *is* **NP**-*complete.*

□

3.3 (E, E)-crosswords

In this section, we give a simple proof of a companion result to Proposition 3.10: The binary regex crossword problem remains **NP**-complete even if we insist that the grid is square and that the row and column regexes equal each other. As with Theorem 3.2, Theorem 3.13 below was shown in [6] via an indirect, complicated series of results. Here we give a much more direct argument.

Definition 3.11. For regexes R and S and $n \geq 0$, we write $R \equiv_n S$ to mean that R and S match exactly the same strings of length n.

Definition 3.12. Define the language

$$\mathsf{R_{=}C} := \{\langle 0^\ell, E\rangle \mid \langle 0^\ell, 0^\ell, E, E\rangle \in \mathsf{RC}\} .$$

A *solution* to an instance $\langle 0^\ell, E\rangle$ is the same as a solution to $\langle 0^\ell, 0^\ell, E, E\rangle$.

Note that the underlying alphabet is that of RC, which is assumed to be $\{0,1\}$.

Theorem 3.13. $\mathsf{R_{=}C}$ *is* **NP**-*complete.*

Proof. Membership in **NP** is immediate. For **NP**-hardness, we reduce from $\mathsf{3SAT}$.

Given a 3-cnf Boolean formula φ with $k \geq 1$ variables and d clauses as defined in Section 2.1 above (where we can assume $d \geq 2$ without loss of generality), letting $q := k + d$, we construct a $3q \times 3q$ (E, E)-crossword that is solvable if and only if φ is satisfiable. We define E as follows: first we define regexes R and C, etc. exactly as in (2–7) of the proof of Theorem 3.2, except we modify C slightly for technical convenience:

$$C := \mathbf{1}^k\,(\mathbf{0}^*\mathbf{1}\mathbf{0}^*) \cup \mathbf{0}^k\,(\mathbf{0}^* \cup \mathbf{1}^*) \equiv_q \mathbf{1}^k \left(\bigcup_{i=1}^{d} t_i\right) \cup \mathbf{0}^k \left(\mathbf{0}^d \cup \mathbf{1}^d\right) . \tag{8}$$

Remark. Since we will only consider strings of length q where C is concerned, it does not matter which of these two regexes we take for C. The latter regex *only* matches strings of length q. One advantage of the latter expression is that we get **NP**-hardness even when restricted to regexes avoiding the $*$-operator completely. (R can also be modified in a similar way to avoid the $*$-operator, and thus so can E. See (9–10), below.) □

The proof of Theorem 3.2 can be modified easily to show that φ is satisfiable if and only if $\langle 0^q, 0^q, R, C\rangle$ is solvable; in any solution, the spine is simply be repeated in the first k rows and appears nowhere else. We now define

$$E := \mathbf{0}^{2q}R \,\cup\, C\mathbf{1}^{2q}\ .$$

Constructing $P := \langle 0^{3q}, E\rangle$ from φ clearly takes polynomial time. It remains to show that φ is satisfiable if and only if $P \in \mathsf{R_{=}C}$, i.e., iff P is solvable.

For the forward implication, suppose φ has a satisfying assignment $\langle z_1, \dots, z_k\rangle$, where each z_i is in $\{0,1\}$. Then, similarly to the proof of Theorem 3.2, there exists a $q \times q$ solution X to the corresponding (R,C)-crossword, shown in Figure 2.

$$X \;:=\; \begin{array}{c|ccccc|ccc} & c_1 & c_2 & c_3 & \cdots & c_d & c_{d+1} & \cdots & c_{d+k} \\ \hline r_1 & 1 & 1 & 1 & \cdots & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_k & 1 & 1 & 1 & \cdots & 1 & 0 & \cdots & 0 \\ \hline r_{k+1} & 1 & 0 & 0 & \cdots & 0 & z_1 & \cdots & z_k \\ r_{k+2} & 0 & 1 & 0 & \cdots & 0 & z_1 & \cdots & z_k \\ r_{k+3} & 0 & 0 & 1 & \cdots & 0 & z_1 & \cdots & z_k \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \cdots & \vdots \\ r_{k+d} & 0 & 0 & 0 & \cdots & 1 & z_1 & \cdots & z_k \end{array}$$

Figure 2: *Solution (modified)*

Here, the rows $r_1, \dots, r_k$ of X match the spine S, columns $c_1, \dots, c_d$ match $\mathbf{1}^k t_1, \dots, \mathbf{1}^k t_d$, respectively, and column c_{d+i} equals $0^k(z_i)^d$ for each $1 \le i \le k$, enabling row r_{k+j} to match $t_j R_j$ for $1 \le j \le d$.

One can readily verify that the following $3q \times 3q$ grid is then a solution to P:

0	0	X
0	1	1
X^{T}	1	1

This grid is chopped into nine $q \times q$ subgrids. Those marked as 0 or 1 contain all 0's or all 1's, respectively. The solution X of Figure 2 appears as the upper right subgrid, and its transpose X^{T} appears as the lower left. Note that 0^q matches C, and hence 0^q1^{2q} matches E, and so the middle rows and columns are also legal. This shows the forward implication.

For the reverse implication, consider any solution G of P with rows $r_1, \dots, r_{3q}$ and columns

$c_1, \ldots, c_{3q}$, chopped into nine $q \times q$ subgrids:

$$G = \begin{array}{|c|c|c|} \hline G_{11} & G_{12} & G_{13} \\ \hline G_{21} & G_{22} & G_{23} \\ \hline G_{31} & G_{32} & G_{33} \\ \hline \end{array}$$

We show that G_{13} must be a solution to $\langle 0^q, 0^q, R, C\rangle$. Then, as in the proof of Theorem 3.2, φ must be satisfiable.

First note that every column of G_{11}, G_{12}, and G_{13} must match C. This follows directly from the structure of E: each column of G matches either: (1) $\mathbf{0}^{2q}R$, whence its prefix of length q equals 0^q (which matches C); or (2) $C\mathbf{1}^{2q}$, whence its prefix of length q matches C. Thus it suffices to show that every row of G_{13} matches R.

Recall that

$$E = \mathbf{0}^{2q}R \,\cup\, C\mathbf{1}^{2q} = \mathbf{0}^{2q}\left(\left(\textstyle\bigcup_{i=1}^{d} t_iR_i\right) \cup \mathbf{1}^d\mathbf{0}^*\right) \cup \left(\mathbf{1}^k\,(\mathbf{0}^*\mathbf{1}\mathbf{0}^*) \cup \mathbf{0}^k\,(\mathbf{0}^* \cup \mathbf{1}^*)\right)\mathbf{1}^{2q} \tag{9}$$

$$\equiv_{3q} \mathbf{0}^{2q}\left(\left(\textstyle\bigcup_{i=1}^{d} t_iR_i\right) \cup \mathbf{1}^d\mathbf{0}^k\right) \cup \left(\mathbf{1}^k\left(\textstyle\bigcup_{i=1}^{d} t_i\right) \cup \mathbf{0}^k\left(\mathbf{0}^d \cup \mathbf{1}^d\right)\right)\mathbf{1}^{2q}\,, \tag{10}$$

where t_i and R_i are defined by (2) and (6), respectively. From E we observe that G_{22} must be either all 0's or all 1's, because the middle third of any string matching E is either 0^q or 1^q. We consider each case in turn.

G_{22} is all 0's. In this case, columns $c_{q+1}, \ldots, c_{2q}$ all must match $\mathbf{0}^{2q}R$. It follows that G_{12} is all 0's. This implies that rows $r_1, \ldots, r_q$ must all match $\mathbf{0}^{2q}R$ (because they cannot match $C\mathbf{1}^{2q}$), making all the rows of G_{13} match R.

G_{22} is all 1's. As in the previous case, it suffices in this case to show that G_{12} is all 0's. Suppose otherwise. Then some column p of G_{12} contains a 1. Noting that p matches C, there are just two possibilities for p:

Case 1: $p = 1^k0^{m-1}10^{d-m}$ for some $1 \le m \le d$. Then $r_1, \ldots, r_k$ and r_{k+m} must all match $C\mathbf{1}^{2q}$ (because they do not start with 0^{2q}), and this implies that the first k rows and the $(k+m)^{\text{th}}$ row of G_{13} are all 1^q. Since each column of G_{13} matches C, the only way this can happen is when each column of G_{13} equals p as well. Now consider another row r of G_{13} besides the first k rows and the $(k+m)^{\text{th}}$ row. (Such a row exists because $d \ge 2$.) We then have $r = 0^q$, but this is impossible, as no row or column of G can end with 0^q. Contradiction.

Case 2: $p = 0^k1^d$. Similarly to Case 1, all rows $r_{k+1}, \ldots, r_q$ of G must match $C\mathbf{1}^{2q}$. Hence, rows $(k+1)$ through q of G_{13} all equal 1^q. Since there are at least two such rows (recall that $d \ge 2$), each column of G_{13} must equal 0^k1^d (since it matches C). This makes the first k rows of G_{13} all equal 0^q, which is again impossible. Contradiction.

This completes the proof. $\square$

Remark. We can give the number of solutions of P in terms of the number of satisfying assignments to φ, as we did in Lemma 3.3 above. Here, it is slightly more complicated, however. Let G be any solution to P, chopped up into $q \times q$ subgrids G_{ij} as above. If G_{22} is all 1's, then this uniquely determines the rest of the grid except for G_{13} and G_{31}: the former being a solution to some (modified) solution to the corresponding RC instance; the latter being the transpose of some solution. Thus if φ has s many satisfying assignments, there are $d!s$ many ways of choosing G_{13} and the same number of ways of (independently) choosing G_{31}, making $(d!s)^2$ many choices in all. Again, these all have G_{22} all 1's.

There can be, however, some anomalous solutions where G_{22} is all 0's. These only occur if φ is satisfied by $\langle 0, 0, \ldots, 0\rangle$ (all variables false), and in this case, G_{13} and G_{31} must correspond to this assignment, and the rest of the grid is determined by the choice of G_{13} and G_{31}. This gives $(d!)^2$ many additional solutions when φ is satisfied by $\langle 0, 0, \ldots, 0\rangle$. Thus we get that the number t of solutions to P is

$$t = \begin{cases} (d!)^2 s^2 & \text{if } \langle 0, 0, \ldots, 0\rangle \text{ does not satisfy } \varphi, \\ (d!)^2(s^2+1) & \text{otherwise.} \end{cases}$$

□

4 (R, C)-games

For two given regexes R and C over an alphabet Σ, an *(R, C)-game* is a two-player combinatorial game that can be thought of as follows: We start with a two-dimensional grid X with m rows and n columns (m and n are positive integers). X is initially empty. Player 1, who we call *Rose*, fills in the first row of X with symbols from Σ to form a string matching R. Player 2, who we call *Colin*, responds by filling the remainder of the first column of X with symbols from Σ so that the entire column matches C. Rose then fills the remainder of the second row so that it matches R, then Colin the remainder of the second column to match C, etc. The first player unable to fill a row (respectively, column) in this way loses, and the other player wins.[5]

We represent an (R, C)-game as a 4-tuple $\langle 0^m, 0^n, R, C\rangle$, where m and n are positive integers (the number of rows and columns of the grid, respectively), and R and C are the corresponding regexes over Σ. Note that the numbers m and n are given in *unary*.

Definition 4.1. Given an alphabet Σ, the language RCG_Σ is the set of all (R, C)-games where R and C are regexes over Σ and where Rose has a winning strategy. We drop the subscript if Σ is clear from the context.

4.1 Upper-bounding the complexity of RCG

It is straightforward to observe that RCG is in **PSPACE**.

Proposition 4.2. $\mathsf{RCG}_\Sigma \in$ **PSPACE** *for any alphabet* Σ.

Proof sketch. This follows straightforwardly from the properties of (R, C)-games: Given an instance of RCG of size N,

- all game positions are representable by strings of polynomial length (in N),

[5]For the last move of the game, Rose or Colin may encounter a row or column, respectively, that is already completely filled in. In this case, she or he wins if and only if the respective row or column matches its corresponding regex.

- any play of the game lasts for at most polynomially many turns, and
- given any game position, whether a given next move is legal can be determined in polynomial space (polynomial time, in fact).

For this it is crucial that the dimensions of the board be given in unary. If the dimensions were given in binary, then we conjecture that the corresponding language would be complete for **EXPSPACE**. Also note that the regex matching problem ("Given a regex E and string w, does w match E?") is in **P**. $\square$

4.2 Hardness of RCG

Here is the main result of this section:

Theorem 4.3. $\mathsf{TQBF} \leq_p \mathsf{RCG}_{\{0,1,2\}}$.

To prove Theorem 4.3, we first consider a variant of RCG, where each row and each column may correspond to a different regex, that is, the input is a pair $\langle\langle R_1, \ldots, R_m\rangle, \langle C_1, \ldots, C_n\rangle\rangle$ of lists of regexes over a given alphabet Σ. Rose and Colin alternate turns as before, but on her i^{th} turn, Rose must fill the remainder of the i^{th} row so that it matches R_i, and similarly, on his j^{th} turn, Colin must fill the remainder of the j^{th} column so that it matches C_j. Call this variant RCG'_Σ.

We show our main result in two steps: in Lemma 4.4 we show how to polynomially reduce TQBF to $\mathsf{RCG}'_{\{0,1\}}$; then we give a polynomial reduction from $\mathsf{RCG}'_{\{0,1\}}$ to $\mathsf{RCG}_{\{0,1,2\}}$ (Lemma 4.5 below). Lemmas 4.4 and 4.5 immediately imply Theorem 4.3. In using RCG', the goal is to first consider this "simpler" game to verify that there is a correspondence between the formulæ in TQBF and the possible games in RCG.

Lemma 4.4. $\mathsf{TQBF} \leq_p \mathsf{RCG}'_{\{0,1\}}$.

Proof. Throughout this proof, we drop the alphabet subscript, letting RCG' denote $\mathsf{RCG}'_{\{0,1\}}$.

Given an instance φ of TQBF as in Equation (1) of Section 2.2:

$$\varphi \coloneqq \exists x_1 \forall y_1 \cdots \exists x_k \forall y_k \; \tilde{\varphi}(x_1, y_1, \ldots, x_k, y_k)$$

with $c \geq 1$ clauses (numbered 1 through c from left to right) and $2k$ variables (with $k \geq 1$), we construct an equivalent instance of RCG' with $m \coloneqq k + c$ rows and $n \coloneqq k + c - 1$ columns. The intersection of the first k rows and first k columns we will call the *variable region*. The players choose truth values for the variables in this region. There are c rows below this region, one for each clause of $\tilde{\varphi}$, which collectively we call the *clause region*. This is the region where Rose, if she can, verifies that all the clauses of $\tilde{\varphi}$ are satisfied by the values for the variables chosen previously in the variable region. The regexes for each player in RCG' are defined as follows (with an explanation afterward): for $1 \leq i \leq m$, we let

$$R_i \coloneqq \begin{cases} (\mathbf{0} \cup \mathbf{1})^* & \text{if } 1 \leq i \leq k, \\ (\mathbf{0} \cup \mathbf{1})^* \mathbf{1} (\mathbf{0} \cup \mathbf{1})^* \mathbf{0}^{c-1} & \text{if } k+1 \leq i \leq m, \end{cases}$$

and for all $1 \leq i \leq n$, we let

$$C_i \coloneqq \begin{cases} \bigcup_{a \in \{\mathbf{0},\mathbf{1}\}} (\mathbf{0} \cup \mathbf{1})^{i-1} a (\mathbf{0} \cup \mathbf{1})^{k-i} \| (S_{a,\mathbf{0},i} \cup S_{a,\mathbf{1},i}) & \text{if } 1 \leq i \leq k, \\ (\mathbf{0} \cup \mathbf{1})^* & \text{if } k+1 \leq i \leq n, \end{cases}$$

where the regexes $S_{a,b,i}$ for $b \in \{\mathbf{0},\mathbf{1}\}$ are defined as follows: First, for $1 \le j \le c$, let

$$u_{i,j} \coloneqq \begin{cases} \mathbf{0} & \text{if } x_i \text{ occurs negatively in clause } j, \\ \mathbf{1} & \text{if } x_i \text{ occurs positively in clause } j, \\ \perp & \text{if } x_i \text{ does not occur in clause } j. \end{cases}$$

for $1 \le i \le k$, and similarly let

$$v_{i,j} \coloneqq \begin{cases} \mathbf{0} & \text{if } y_i \text{ occurs negatively in clause } j, \\ \mathbf{1} & \text{if } y_i \text{ occurs positively in clause } j, \\ \perp & \text{if } y_i \text{ does not occur in clause } j. \end{cases}$$

for $1 \le i \le k$. Now for $1 \le j \le c$ and $a, b \in \{\mathbf{0},\mathbf{1}\}$ define

$$d_{a,b,i,j} \coloneqq \begin{cases} \mathbf{1} & \text{if } u_{i,j} = a \text{ or } v_{i,j} = b, \\ \mathbf{0} & \text{otherwise.} \end{cases} \qquad (1 \le i \le k).$$

Finally, we let $S_{a,b,i} := d_{a,b,i,1} \| \cdots \| d_{a,b,i,c}$ for $1 \le i \le k$. Note that each $S_{a,b,i}$ then matches a unique string of length c.

Example. Suppose $\varphi = \exists x_1 \forall y_1 \exists x_2 \forall y_2 [(\overline{x_1} \vee y_2) \wedge (x_1 \vee \overline{y_1} \vee \overline{x_2}) \wedge (x_2 \vee y_1 \vee \overline{y_2})]$. Then $k = 2$, $c = 3$, and

$$[u_{i,j}] = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \perp \\ \perp & \mathbf{0} & \mathbf{1} \end{bmatrix}, \qquad [v_{i,j}] = \begin{bmatrix} \perp & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \perp & \mathbf{0} \end{bmatrix},$$

$$[d_{\mathbf{0},\mathbf{0},i,j}] = \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \end{bmatrix}, \quad [d_{\mathbf{0},\mathbf{1},i,j}] = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \end{bmatrix}, \quad [d_{\mathbf{1},\mathbf{0},i,j}] = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}, \quad [d_{\mathbf{1},\mathbf{1},i,j}] = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} \end{bmatrix},$$

$$[S_{\mathbf{0},\mathbf{0},i}] = \begin{bmatrix} \mathbf{110} \\ \mathbf{011} \end{bmatrix}, \qquad [S_{\mathbf{0},\mathbf{1},i}] = \begin{bmatrix} \mathbf{101} \\ \mathbf{110} \end{bmatrix}, \qquad [S_{\mathbf{1},\mathbf{0},i}] = \begin{bmatrix} \mathbf{010} \\ \mathbf{001} \end{bmatrix}, \qquad [S_{\mathbf{1},\mathbf{1},i}] = \begin{bmatrix} \mathbf{011} \\ \mathbf{101} \end{bmatrix},$$

and thus

$$\begin{aligned} C_1 &= [\mathbf{0}(\mathbf{0} \cup \mathbf{1}) \| (\mathbf{110} \cup \mathbf{101})] \cup [\mathbf{1}(\mathbf{0} \cup \mathbf{1}) \| (\mathbf{010} \cup \mathbf{011})] \,, \\ C_2 &= [(\mathbf{0} \cup \mathbf{1})\mathbf{0} \| (\mathbf{011} \cup \mathbf{110})] \cup ((\mathbf{0} \cup \mathbf{1})\mathbf{1} \| (\mathbf{001} \cup \mathbf{101})] \,. \end{aligned}$$

□

The entries along the main diagonal of the variable region each correspond to Rose's choice of a the truth value (0 or 1) for x_1 through x_k in the original formula, as depicted in Figure 3. The remainder of the rows (c of them) correspond to the clauses of φ.

x_0	?	?	$\cdot$	?
?	x_1	?	$\ddots$	?
?	?	x_2	$\ddots$	?
$\vdots$	$\ddots$	$\ddots$	$\ddots$	$\vdots$
?	?	?	?	x_k

Figure 3: *The layout of the variable region. The question marks represent either* 0 *or* 1.

Here is how this RCG' game reflects the original instance of TQBF viewed as a game. Fix i such that $1 \le i \le k$. When Rose plays the i^{th} row, she is able to choose a truth value $a \in \{0,1\}$ for x_i by placing a in the corresponding square in Figure 3. (Rose can play any binary string in the remainder of her row, because $R_i = (\mathbf{0} \cup \mathbf{1})^*$.) Then when Colin plays the remainder of the i^{th} column according to C_i, he can effectively choose a truth value for y_i—either 0 or 1—by playing a string whose last c symbols match either $S_{a,\mathbf{0},i}$ or $S_{a,\mathbf{1},i}$, respectively (and these are the only two choices for Colin, because he is constrained by Rose's choice of a). His play lets Rose know for her next turn the truth value he chose for y_i.[6] Because of the $S_{a,b,i}$ component of C_i, Colin is forced to place a 1 in each of the last c positions corresponding to a clause that is satisfied by the truth settings just chosen for x_i and y_i.

Also note that in order for Rose to complete the board, there must be a 1 in at least one of the first k positions in every row of the clause region. That is, Rose can win just when the chosen truth values of the variables satisfy all clauses of $\tilde{\varphi}$. Thus the two games are equivalent. Our construction is clearly polynomial time, which finishes the proof. □

4.2.1 Constraining the regexes to be row- and column-independent

Lemma 4.5. *For any alphabet Σ such that $\{0,1\} \subseteq \Sigma$ and $2 \notin \Sigma$, $\mathsf{RCG}'_\Sigma \le_p \mathsf{RCG}_{\Sigma\cup\{2\}}$.*

The rest of this section is a proof of Lemma 4.5. From now on, we let RCG' and RCG denote RCG'_Σ and $\mathsf{RCG}_{\Sigma\cup\{2\}}$, respectively. To reduce from RCG' to RCG we need to provide a method to consolidate the families of regexes into one regex per player. Here, we present a generic construction that can be applied to any RCG' game — forcing each player to play their families of regexes in index order.

Given an arbitrary instance $G := \langle\langle R_1, \dots, R_m\rangle, \langle C_1, \dots, C_n\rangle\rangle$ of RCG', we construct an equivalent instance of RCG. Our construction requires the RCG alphabet to contain a third symbol "2" that is not part of any string matching any of the R_i or C_i. We currently do not know how to remove this requirement. We can assume that the grid for G is square, i.e., $m = n$: Suppose this is not the case; for example, suppose $m < n$. Then we can pad the grid with $n - m$ bottom rows by

- concatenating each C_i with $\mathbf{0}^{n-m}$ on the right, and
- defining $R_i := \mathbf{1}^*$ for $m < i \le n$,

yielding an evidently equivalent $n \times n$ game. If $m > n$, we do the same thing but swap the roles of the rows and columns. The instance of RCG we construct from G will then be a $(2n+1) \times (2n+1)$ game $H := \langle 0^{2n+1}, 0^{2n+1}, R, C\rangle$. We may also assume without loss of generality that $n \ge 3$.

The regexes R and C we construct for the respective players are given below, again with explanations afterwards:

[6] Except in the case where $S_{a,\mathbf{0},i}$ or $S_{a,\mathbf{1},i}$ match the same string. But in this case, the value Colin chooses for y_i does not matter, since both values satisfy the exact same clauses.

$$R \coloneqq \mathbf{210}^* \quad \cup \tag{11}$$

$$\underbrace{\bigcup_{i=1}^{n-1} \mathbf{0}^{i-1}\mathbf{1}^3\mathbf{0}^{n-i-1}}_{\text{I}} \,\|\, \underbrace{\mathbf{0}^{i-1}\mathbf{1}\mathbf{0}^{n-i}}_{\text{II}} \quad \cup \tag{12}$$

$$\underbrace{\mathbf{0}\mathbf{0}^{n-2}\mathbf{1}\mathbf{1}}_{\text{Ir}} \,\|\, \underbrace{\mathbf{0}^{n-1}\mathbf{1}}_{\text{II}} \quad \cup \tag{13}$$

$$\bigcup_{i=1}^{n} \underbrace{\mathbf{0}^{i}\mathbf{1}\mathbf{0}^{n-i}}_{\text{III}} \,\|\, R_i \tag{14}$$

(a) Rose's regex. Regex (11) is the 'spine regex', while regexes (12–13) define the 'calibration' region (I, II). Regex (14) continues calibration in region III while also including the row regexes from G (played in region IV).

$$C \coloneqq \mathbf{210}^* \quad \cup \tag{15}$$

$$\underbrace{\bigcup_{i=1}^{n-1} \mathbf{0}^{i-1}\mathbf{1}^3\mathbf{0}^{n-i-1}}_{\text{I}} \,\|\, \underbrace{\mathbf{0}^{i-1}\mathbf{1}\mathbf{0}^{n-i}}_{\text{III}} \quad \cup \tag{16}$$

$$\underbrace{\mathbf{0}\mathbf{0}^{n-2}\mathbf{1}\mathbf{1}}_{\text{Ic}} \,\|\, \underbrace{\mathbf{0}^{n-1}\mathbf{1}}_{\text{III}} \quad \cup \tag{17}$$

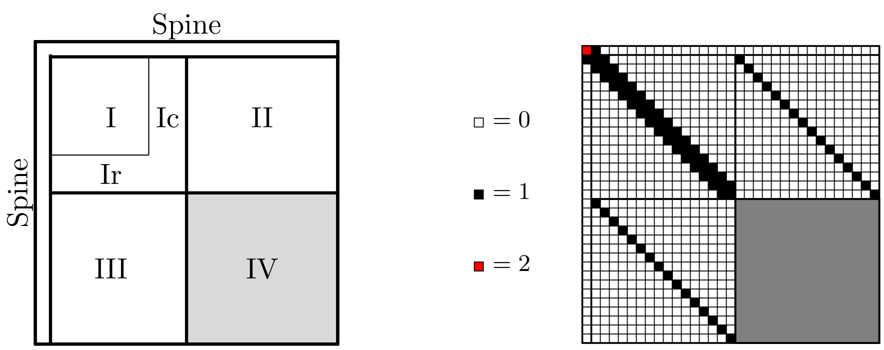
$$\underbrace{\bigcup_{i=1}^{n} \mathbf{0}^{i}\mathbf{1}\mathbf{0}^{n-i}}_{\text{II}} \,\|\, C_i \quad \cup \tag{18}$$

$$(\mathbf{0} \cup \mathbf{1} \cup \mathbf{100} \cup \mathbf{00}^*\mathbf{10})\mathbf{2}^* \tag{19}$$

(b) Colin's regex. Regex (15) is the 'spine regex', regexes (16–17) are the calibration region (I and III), regex (18) continues calibration in region II while also including the column regexes from G (played in region IV), and regex (19) is a 'bomb' used to punish Rose for cheating.

Figure 4: *The regexes wrapping games in* RCG*. Regexes are bracketed with the regions they describe, illustrated in Figure 5a.*

Figure 5a illustrates how H 'wraps' around the game G: players first fill in the spine, which consists of the first row and first column, then regions I, II, and III before simulating the game G in the lower right square (region IV).



(a) Regions of the board **(b)** An example of normal play

Figure 5: *Regions I–IV to constrain the players. Each region is an $n \times n$ square.*

4.2.2 Normal Play

By a *round*, we mean a pair of consecutive turns, starting with Rose. We index the rounds starting with round 0. Normal play, i.e., play where neither player cheats (see below), is in three stages:

Spine: In round 0, both players play the 'spine string,' i.e., 210^{2n-1}, the unique string of length $2n + 1$ matching $\mathbf{210}^*$.

Calibration: In round i, where $1 \le i \le n$, Rose and Colin each play a 'calibration string,' i.e., either the string matching $\mathbf{0}^{i-1}\mathbf{1}^3\mathbf{0}^{n-i-1}\|\mathbf{0}^{i-1}\mathbf{1}\mathbf{0}^{n-i}$ (if $i < n$) or the one matching $\mathbf{0}\mathbf{0}^{n-2}\mathbf{11}\|\mathbf{0}^{n-1}\mathbf{1}$ (if $i = n$).

Simulation: Rose and Colin now simulate the given RCG' game: In round $(n+i)$, for $1 \le i \le n$, Rose plays a string matching $\mathbf{0}^i\mathbf{1}\mathbf{0}^{n-i}\|R_i$ (if she can), and Colin plays a string matching $\mathbf{0}^i\mathbf{1}\mathbf{0}^{n-i}\|C_i$ (if he can).

Figure 5b illustrates the state of the grid after round n of normal play (here, $n = 16$). If either player deviates from normal play, we say that the first player to do so is *cheating*. The next lemmas show that Colin cannot cheat, and if Rose cheats, then Colin can force her to lose in a constant number of rounds by *dropping a bomb*, i.e., playing a string matching $(\mathbf{0} \cup \mathbf{1} \cup \mathbf{100} \cup \mathbf{00}^*\mathbf{10})\mathbf{2}^*$ (cf. (19) above), once or twice.

Note that, except for the spine string and bombs, the length-$(n+1)$ prefix of any string played by either player must match $(\mathbf{0} \cup \mathbf{1})^*$, and such a prefix has at least four characters.

Claim 4.6. *In round* 0*, if Rose does not play the spine string, then Colin can win; otherwise, Colin must also play the spine string.*

Proof. If Rose does not play $\mathbf{210}^*$, she has two choices for her first character b: either 0 or 1. Whichever b she chooses, Colin can drop a bomb matching $(\mathbf{0} \cup \mathbf{1})\mathbf{2}^*$ (see Figure 6), which forces Rose to play the spine string in any subsequent round.

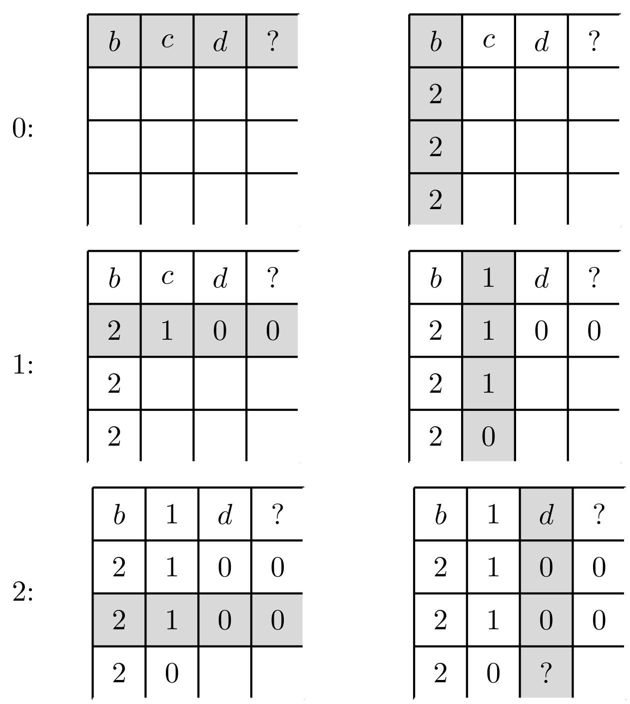


Figure 6: *The first three rounds when Rose cheats with $bcd\cdots$ round* 0*, for some $b, c, d \in \{0, 1\}$, and Colin then drops a bomb. Note that Rose has no regex to match the prefix* 20*. We set $c := 1$ in round* 1 *to show the worst case, where Colin must survive through round* 2 *(not required if $c = 0$).*

We now have two cases for Colin's move in round 1 (see Figure 6, middle left), depending on Rose's second character $c \in \{0, 1\}$ played in round 0:

If $c = 1$: Colin plays $1110\cdots$ as shown in Figure 6 (cf. regex (16) where $i = 1$). After Rose plays the spine string in round 2, Colin survives this round by playing any string starting with prefix $d00$ where $d \in \{0, 1\}$, e.g., either the bomb $1002\cdots$ or the bomb $000102\cdots$. Rose then cannot play in round 3, as she has no legal option with prefix 20.

If $c = 0$: Colin drops the bomb $0102\cdots$, preventing Rose from playing in round 2, as she has no legal option with prefix 20.

Thus in either case, Rose quickly loses.

If Rose does play the spine string $2100\cdots$ in round 0, then Colin must play a string starting with 2, his only option matching the spine regex $\mathbf{210}^*$. This proves the claim. $\square$

Claim 4.7. *If Rose cheats in any round 1 through n, then Colin can win. That is, after normal play through round $(i-1)$ for $1 \leq i \leq n$, Rose prefers regex (12) to regex (14) in round i if $i < n$, and she prefers regex (13) to regex (14) in round n.*

Proof. In round 1, because of the spine, Rose must play a string with prefix 1, and so she must play a string matching regex (12). Now suppose $2 \leq i \leq n$, and consider the following portion of the board at the start of round i when both players have been playing normally (we show the case where $i < n$; the portion of the board at the start of round n looks similar):

?	1	0	0
1	1	1	0
0	1		
0	0		

Rose must play a string with prefix $0^{i-1}1$. If $i < n$, then Rose's choice is between regexes (12) and (14), as these are the ones that can match a string with that prefix. (If $i = n$, then Rose's choice is between regexes (13) and (14).) Say Rose cheats by choosing regex (14), thus playing a string matching $\mathbf{0}^{i-1}\mathbf{1}\mathbf{0}^{n-i+1} \| R_{i-1}$. Colin can then respond by dropping the bomb $0^{i-1}102\cdots$:

1	1	0	0
1	1	1	0
0	1	0	0
0	0		

(a) Rose cheats; plays regex (14)

1	1	0	0
1	1	1	0
0	1	0	0
0	0	2	

(b) Colin plays regex (19); Rose loses

Rose cannot then play any string with prefix 0^i2, so she loses in round $(i+1)$. $\square$

Claim 4.8. *Colin cannot cheat in any round 0 through n.*

Proof. Colin cannot cheat in round 0 by Claim 4.6, so we consider Colin's move in round i for $1 \leq i \leq n$. Assuming normal play beforehand, Colin is faced with the prefix $0^{i-1}11$ in round i, and thus must play a string matching regex (16) (if $i < n$) or (17) if $i = n$), i.e., play normally. $\square$

The preceding lemmas show that normal play is optimal for both players (even required for Colin) through round n. Thus we can assume normal play through round n, filling regions II and III of the grid with 1's along their diagonals and 0's elsewhere (as with the identity matrix).

Claim 4.9. *Assume normal play through round n. For $1 \le i \le n$, in round $(n+i)$, Rose must play a string matching $\mathbf{0}^i\mathbf{1}\mathbf{0}^{n-i}\|R_i$ and Colin must play a string matching $\mathbf{0}^i\mathbf{1}\mathbf{0}^{n-i}\|C_i$. That is, Rose and Colin must play normally in rounds $(n+1)$ through $2n$.*

Proof. In round $(n+i)$, Rose and Colin are both faced with prefix 0^i10^{n-i}, and, except for the bomb regex, the only regexes that could possibly match a string with this prefix are the respective regexes given above for Rose and Colin. It remains to argue that Colin cannot drop a bomb in round $(n+i)$ for $1 \le i \le n$: The only two bombs with prefixes of the form 0^i10^{n-i} are $0^{n-1}102^n$ and 0^n102^{n-1}. Colin cannot drop either of these unless $i \ge n-1$, but by that time, Rose has already filled in the first two rows of region IV with strings matching R_1 and R_2, respectively, and neither of these strings can contain a 2. □

In rounds $(n+1)$ through $2n$, the players are essentially playing the game G in region IV, so the winner of H is the winner of G. This completes the proof of Lemma 4.5. Combining Lemmas 4.4 and 4.5 with $\Sigma := \{0,1\}$ proves Theorem 4.3.

5 (R, C)-crosswords and Turing Machines

In this section we show how an (R,C)-crossword solution can closely reflect an arbitrary Turing machine computation. This was previously done to a large extent by Giammarresi & Restivo, who essentially showed that the unbounded version of the regex crossword problem ("Given regexes R and C, does there exist an (R,C)-crossword solution of any size?") is undecidable [9].

Definition 5.1. For alphabet Σ, define the language

$$\mathsf{URC}_\Sigma := \{\langle R,C\rangle \mid R \text{ and } C \text{ are regexes over } \Sigma \text{ and } \langle 0^m,0^n,R,C\rangle \text{ is solvable for some } m,n>0\}\ .$$

The "U" in URC_Σ stands for "unbounded."

Theorem 5.2 (Giammarresi, Restivo [9])**.** *There exists an alphabet Σ such that URC_Σ is undecidable (m-equivalent to the Halting Problem).*

Their result was given in the context of characterizing 2-dimensional languages by way of tiling systems, which they used to constrain the computational trace of an arbitrary Turing machine.

In this section, we give a more direct connection between crosswords and computations and prove a slightly stronger version of this result, namely, there is a *fixed* regex C such that the problem, "given a regex R, does an (R,C)-crossword solution exist?" is undecidable. Using techniques from Section 6, we obtain that it is undecidable whether an (E,E)-crossword solution exists. This strengthens Theorem 5.2, which holds that given *both* regexes R and C, determining whether an (R,C)-crossword solution exists is undecidable.

Definition 5.3. Given alphabet Σ and regex C over Σ, define the language

$$\mathsf{URC}_\Sigma(C) := \{R \mid R \text{ is a regex over } \Sigma \text{ and } \langle R,C\rangle \in \mathsf{URC}_\Sigma\}\ .$$

Theorem 5.4. *There exists an alphabet Σ and regex C over Σ such that $\mathsf{URC}_\Sigma(C)$ is undecidable (m-equivalent to the Halting Problem).*

The next definition is for purely technical reasons. It is used mainly in Section 6. Removing these restrictions does not affect our complexity results.

Definition 5.5. We say that a regex is *positive* iff it is not matched by the empty string. A pair (R, C) of regexes is *plural* iff both R and C are positive and every (R, C)-crossword solution has at least two rows and at least two columns.

The rest of this section is a proof of Theorem 5.4 but with the main technical argument relegated to an appendix.

We reduce from the Halting Problem. Our computational model—a slight modification of that found in many textbooks, e.g., [12]—is that of a deterministic Turing machine with a unique halting state (distinct from the start state) and a single one-way infinite tape whose initial contents starts with blank symbols in the two left-most cells, followed by an input string w of nonblank symbols, followed on the right with blank tape. In each step, the tape head must move either left or right by one cell. The grid to be filled in encodes the tableau of a halting computation: each row encodes the configuration of the machine at a single time step, and each column encodes the history of a single tape cell throughout the computation. Thus each symbol in the crossword solution represents the contents of a tape cell at a certain time in the computation, possibly with some extra information about the state of the machine and the position of the head. The expression R ensures that the whole configuration of the Turing machine is legitimate at each time step, and C ensures that the contents of each tape cell is correct over time. We view the tableau with the initial configuration on the top row and time moving downward.

Remark. One might think that, in order to handle transitions correctly, a grid symbol should represent a "window" in the tableau, spanning perhaps two or three adjacent tape cells at two adjacent time steps, and that these windows should overlap consistently. It is possible to do this, but it turns out to be unnecessary; we use a trick whereby the machine's transition information is passed in two directions—first horizontally (checked by R), then vertically (checked by C). (This idea is somewhat analogous to the characterization of recognizable picture languages via domino systems and hv-local languages [11].) □

Both results of this section use the following lemma, which we prove in Appendix A using the formal Turing machine model in detail. It says essentially that halting Turing machine computations correspond one-to-one with crossword solutions whose dimensions are roughly the time and space requirements of the computation (up to an additive constant). Lemma 5.6 is stronger than what is needed for Theorem 5.4. The extra strength will be used in Section 7.

Lemma 5.6. *Let M be a Turing machine (as described above). There exists an alphabet Σ and a regex $C := C(M)$ over Σ (Σ and C both depending on M), and for any input string w there exists a regex $R := R(M, w)$ over Σ (depending on M and w) such that (R, C) is plural, and M halts on input w if and only if an (R, C)-crossword solution exists, and if this is the case, then*

- *the (R, C)-crossword solution is unique, and*
- *there is a constant c, independent of M and w, such that the unique solution is a grid with between $t + 2|w|$ and $t + 2|w| + c$ rows and between $\max(s, |w|)$ and $\max(s, |w|) + c$ columns, where t (respectively s) is the number of steps M takes (respectively, the number of cells M ever scans) on input w.*

Furthermore, R is computable from M and w in polynomial time, and C is computable from M.

Proof. See Appendix A. $\square$

Lemma 5.6 yields the following result, which immediately implies Theorem 5.4 as a corollary:

Theorem 5.7. *Given alphabet Σ and positive regex C over Σ, let $\mathsf{W}_\Sigma(C)$ be the following decision problem:*

> $\mathsf{W}_\Sigma(C) :=$ *"Given a regex R over Σ such that (R, C) is plural, does an (R, C)-crossword solution exist?"*

There exists an alphabet Σ and a positive regex C over Σ such that $\mathsf{W}_\Sigma(C)$ is m-equivalent to the Halting Problem (and is thus undecidable).

Proof. We apply Lemma 5.6 letting M be a universal Turing machine (or any Turing machine recognizing the Halting Problem). Let Σ and C be as constructed in the proof. We get a computable function g such that, for any string w, $g(w)$ is a regex R over Σ such that (R, C) is plural, and for all w, M halts on w if and only if an (R, C)-crossword solution exists. Thus g m-reduces the Halting problem to $\mathsf{W}_\Sigma(C)$. Conversely, $\mathsf{W}_\Sigma(C)$ is clearly c.e. (for all C uniformly, in fact), and thus m-reduces to the Halting Problem. $\square$

Corollary 5.8 (Giammarresi, Restivo [9])**.** *Given regexes R and C, it is undecidable (m-equivalent to the Halting Problem) whether an (R, C)-crossword solution exists.*

Proof. Just note that decision problem W of Theorem 5.7 is c.e. uniformly in C. $\square$

6 Techniques for Restricting Regex Crossword Problems

In this section, we show, given two regexes R and C, how to find a regex E such that an (R, C)-crossword solution exists of a certain size if and only if an (E, E)-crossword solution exists of a roughly similar size. We gave a special case of such a reduction in the proof of Theorem 3.13 in Section 3.3. In Section 6.1 we give a different, general construction that works for any regexes over any alphabet. We also show how to convert a regex crossword over an arbitrary alphabet into an equivalent regex crossword over the binary alphabet $\{0, 1\}$ (see Section 6.3). Finally, in Section 6.4 we show that insisting on a square solution (with the same number of rows as columns) does not alter our hardness results.

6.1 Making the row and column expressions equal

(R, C)-crossword problems retain their hardness even if we insist that $R = C$. This was the case with Theorem 3.13, for example. Here, we give a generic construction that can be applied more generally. We get this from the following lemma:

Lemma 6.1. *There exists a polynomial-time computable function b such that, for any alphabet Σ and any regexes R and C over Σ such that (R, C) is plural, $E := b(\Sigma, R, C)$ is a positive regex (over a slightly bigger alphabet Σ', which can be computed from Σ alone) such that an (E, E)-crossword solution exists if and only if an (R, C)-crossword solution exists. Furthermore, there is a one-to-one map ρ mapping Σ-grids of size $m \times n$ (where $m, n \geq 2$) to Σ'-grids of size $(m+1) \times (n+1)$ that takes (R, C)-crossword solutions to (E, E)-crossword solutions, and for every (E, E)-crossword solution Y, there exists an (R, C)-crossword solution X such that $\rho(X)$ is either Y or the matrix transpose of Y.*

Proof. Let Σ, R, and C be given as in the lemma. We want to effectively find an E so that a unique (E, E)-crossword solution corresponds to any given (R, C)-crossword solution and vice versa. A first attempt at constructing E would be to set $E := R \cup C$. This may not work, because an (R, C)-crossword solution may not exist, but there is an (E, E)-crossword solution where each row and column might match R, but the columns do not match C, say. There are perhaps several ways to correct this problem, and here is a fairly simple fix:

1. Introduce three new symbols not in Σ: $\spadesuit$ (the "bottom edge marker"); $\heartsuit$ (the "left edge marker"); and $\diamondsuit$ (the "corner marker").

2. Then modify R and C slightly to R' and C', respectively, so that any $(R' \cup C', R' \cup C')$-crossword solution or its matrix transpose has its first column matching $\heartsuit^*\diamondsuit$, its last row matching $\diamondsuit\spadesuit^*$, and the rest of the array being an (R, C)-crossword solution:

$$
\begin{array}{|c|ccc|}
\hline
\heartsuit & & & \\
\vdots & & (R,C)\text{-crossword} & \\
\vdots & & \text{solution} & \\
\heartsuit & & & \\
\hline
\diamondsuit & \spadesuit & \cdots \quad \cdots & \spadesuit \\
\hline
\end{array}
$$

Informally, the $\heartsuit$ and $\spadesuit$ markers prevent rows from being confused with columns, and the $\diamondsuit$ marker prevents $\heartsuit$ and $\spadesuit$ from being confused with each other. Here are the formal definitions:

$$
\begin{aligned}
\Sigma' &:= \Sigma \cup \{\spadesuit, \heartsuit, \diamondsuit\} , \\
R' &:= \heartsuit R \cup \diamondsuit\spadesuit\spadesuit\spadesuit^* , \\
C' &:= C\spadesuit \cup \heartsuit\heartsuit\heartsuit^*\diamondsuit , \\
E &:= R' \cup C' .
\end{aligned}
$$

Clearly, $E = b(\Sigma, R, C)$ is positive and computable in polynomial time. To see that this construction works, first observe that an $m \times n$ (R, C)-crossword solution X (with $m, n \geq 2$ because (R, C) is plural) becomes an $(m + 1) \times (n + 1)$ (E, E)-crossword solution $\rho(X)$ by prepending the column $\heartsuit^m$ then appending the row $\diamondsuit\spadesuit^n$. This defines the map ρ, which is clearly one-to-one and maps (R, C)-crossword solutions to (E, E)-crossword solutions with one more row and column. It follows that an (E, E)-crossword solution exists if an (R, C)-crossword solution exists.

Conversely, let Y be any (E, E)-crossword solution—say, $m \times n$—with rows $r_1, \ldots, r_m$ and columns $c_1, \ldots, c_n$, all matching E. We show first that $m, n \geq 3$. Suppose not. We must have $m, n \geq 2$, because both R and C are positive. We may assume that $m = 2$; otherwise, we apply the same argument to the transpose of Y, which is still an (E, E)-crossword solution. Then each column of Y has length 2 and thus must match either $\heartsuit R$ or $C\spadesuit$. We claim that c_2 cannot start with $\heartsuit$: Suppose otherwise. Then since r_1 has $\heartsuit$ as its second symbol, it must match $\heartsuit\heartsuit\heartsuit^*\diamondsuit$, whence c_n starts with $\diamondsuit$; but then $|c_n| \geq 3$, contradicting our assumption that $m = 2$ and establishing the claim. Therefore, it must be that c_2 matches $C\spadesuit$ ($c_2 = a\spadesuit$ for some $a \in \Sigma$ matching C). Then r_2 has $\spadesuit$ as its second symbol, and so it either matches $\diamondsuit\spadesuit\spadesuit\spadesuit^*$ or equals $b\spadesuit$ for some $b \in \Sigma$ matching C. The former case would make c_1 have $\diamondsuit$ as its second symbol, which is impossible. In the latter case, we must have $c_1 = \heartsuit b$, which matches $\heartsuit R$. The resulting grid would then look like this:

$$
\begin{array}{|c|c|c}
\hline
\heartsuit & a & \cdots \\
\hline
b & \spadesuit & \cdots \\
\hline
\end{array}
$$

But then b matches both R and C, making a 1×1 (R,C)-crossword solution, which contradicts the fact that (R,C) is plural.

Having established that $m, n \geq 3$, we next show that removing the first column and last row from either Y or its transpose (depending on where the $\diamondsuit$ is) results in an (R,C)-crossword solution X, from which it will be clear that $\rho(X)$ is either Y or its transpose, respectively.

Consider r_2, which has length ≥ 3 and matches either R' or C'.

Case 1: r_2 matches R'. Then r_2 must begin with $\heartsuit$: otherwise, it begins with $\diamondsuit$, but then c_1 has $\diamondsuit$ as its second symbol, which is impossible. Then we have $r_2 = \heartsuit r$ for some string r matching R, and since c_1 has $\heartsuit$ as its second symbol, we have $c_1 = \heartsuit^{m-1}\diamondsuit$, whence it follows that $r_m = \diamondsuit\spadesuit^{n-1}$. Now consider the columns $c_2, \ldots, c_n$. These all end with $\spadesuit$, and so they must all match $C\spadesuit$ (they cannot match $\diamondsuit\spadesuit\spadesuit\spadesuit^*$ because they all contain symbols in Σ from r_2). So now we know that all symbols in Y other than the first column and last row are in Σ, that is, for each $1 \leq i \leq m-1$, all symbols in r_i are in Σ except the first, which is $\heartsuit$ (because of c_1). The only way this can happen is if each r_i matches $\heartsuit R$. This establishes that Y minus the first column and last row is an (R,C)-crossword solution (whose image under ρ is Y).

Case 2: r_2 matches C'. By transposing Y, we can assume instead that c_2 matches C', which will be less confusing conceptually. The argument here is similar to Case 1. The string c_2 cannot end with $\diamondsuit$, as that would also be the second symbol of r_m, which is impossible. So we have that $c_2 = c\spadesuit$ for some string c matching C, making $\spadesuit$ the second symbol of r_m, which is not its last symbol, because $|r_m| \geq 3$. It follows that $r_m = \diamondsuit\spadesuit^{n-1}$, whence $c_1 = \heartsuit^{m-1}\diamondsuit$. Now then, $r_1, \ldots, r_{m-1}$ all start with $\heartsuit$ and contain at least one symbol from Σ (because of c_2), and so they all match $\heartsuit R$. So again, all symbols in Y except the first column and last row are from Σ, and since $c_2, \ldots, c_n$ all end in $\spadesuit$, they much all match $C\spadesuit$. So again we have that deleting the first column and last row results in an (R,C)-crossword solution.

We have shown that removing the first column and last row from either Y (in Case 1) or its transpose (in Case 2) results in an (R,C)-crossword solution X such that $\rho(X)$ is either Y or its transpose, respectively. In particular, if an (E,E)-crossword solution exists, then an (R,C)-crossword solution exists. $\square$

Theorem 6.2. *For any alphabet Σ, let $\mathsf{UR}_=\mathsf{C}_\Sigma$ be the following decision problem:*

> $\mathsf{UR}_=\mathsf{C}_\Sigma :=$ *"Given a positive regex E over Σ, does an (E,E)-crossword solution (of any size) exist?"*

There exists an alphabet Σ such that $\mathsf{UR}_=\mathsf{C}_\Sigma$ is undecidable (in fact, m-equivalent to the Halting Problem).

Proof. $\mathsf{UR}_=\mathsf{C}_\Sigma$ is clearly c.e. for any Σ, and hence m-reduces to the Halting Problem. Conversely, let Σ and C be as in Theorem 5.7, let b be the function of Lemma 6.1. Define the function h by

$$h(R) := b(\Sigma, R, C)$$

for every regex R over Σ such that (R,C) is plural. Then h is computable in polynomial time, and, for all R such that (R,C) is plural, $E := h(R)$ is a positive regex, and an (E,E)-crossword solution exists if and only if an (R,C)-crossword exists. Thus h m-reduces $\mathsf{W}_\Sigma(C)$ of Theorem 5.7 to $\mathsf{UR}_=\mathsf{C}_{\Sigma'}$, where Σ' is the alphabet computed from Σ in Lemma 6.1. Since $\mathsf{W}_\Sigma(C)$ is m-equivalent to the Halting Problem by Theorem 5.7, we are done. $\square$

6.2 A decidable crossword solution existence problem

In contrast with the previous results, we have the following theorem, which shows that the crossword solution existence problem becomes decidable if we bound one of the grid dimensions but not the other. In the definition below, SB stands for "semi-bounded."

Definition 6.3. For any alphabet Σ, define SBRC_Σ to be the language of all tuples $\langle R_1, \ldots, R_m, C\rangle$ where $R_1, \ldots, R_m, C$ are regexes over Σ and there exists an $n \geq 1$ and an $m \times n$ Σ-grid all of whose columns match C and whose i^{th} row matches R_i for all $1 \leq i \leq m$. (Note that m is specified implicitly by the input.)

Theorem 6.4. *SBRC_Σ is decidable for every alphabet Σ. In fact, $\mathsf{SBRC}_\Sigma \in$ **PSPACE**.*

Proof Sketch. First, we convert each R_i into an equivalent ϵ-NFA N_i (see [10]). These automata have sizes polynomial in the sizes of the regexes. Then we nondeterministically guess a crossword one column at a time, starting with the first, and for each guessed column, we simulate one step of each of the N_i on its corresponding symbol (this can be done in polynomial time by keeping track of a subset of the state set of each N_i). We accept if ever all the N_i accept simultaneously. We can also stop after 2^n guesses, where n is the total number of states of all the N_i combined. This nondeterministic algorithm uses polynomial space, and hence can be converted into a deterministic polynomial-space algorithm by Savitch's theorem. $\square$

6.3 Regexes over the binary alphabet

The alphabets used in Theorems 5.7 and 6.2 are fixed, but they are likely quite large, having to encode all the states of a (modified) universal Turing machine $\widehat{M}$. In this section, we show how to map (in polynomial time) regexes over an arbitrary alphabet to regexes over the binary alphabet in a way that preserves crossword solutions. Thus the size-unbounded solution existence problem remains undecidable even when restricted to a binary alphabet.

The next theorem strengthens Theorem 5.7.

Theorem 6.5. *There exists a regex G over $\{0,1\}$ such that $\mathsf{W}_{\{0,1\}}(G)$ is m-equivalent to the Halting Problem.*

Theorem 6.5 is a quick corollary of the following technical lemma:

Lemma 6.6. *There is a function f such that, for any $k \geq 2$ and positive regex R over alphabet $\Sigma := \{0, \ldots, k-1\}$, $f(k, R)$ is a positive regex over the alphabet $\{0,1\}$ such that the following holds: There exists a one-to-one map ψ_k between Σ-grids and $\{0,1\}$-grids (that maps $m \times n$ grids to $(3k(m+1)+1) \times (3k(n+1)+1)$ grids) such that, for any positive regexes T and U over Σ,*

1. *for any (T, U)-crossword solution X, $\psi_k(X)$ is an $(f(k,T), f(k,U))$-crossword solution, and*
2. *for every $(f(k,T), f(k,U))$-crossword solution Y, there is a (T, U)-crossword solution X such that $\psi_k(X) = Y$.*

Furthermore, f is computable in time polynomial in $k + |R|$.

Proof. Fix k and a positive regex R over $\Sigma := \{0, \ldots, k-1\}$. The regex $F := f(k, R)$ over $\{0,1\}$, defined below, will be formed from several components. Let $\ell := 3k$, noting that $\ell \geq 6$. Any string w matching F will satisfy $|w| \equiv 1 \pmod{\ell}$. For $0 \leq i < \ell - 1$ and any string x of length ℓ, define $\mathrm{RotL}_i(x)$ to be the cyclic shift of x by i places to the left. That is, if $x = x_0 \cdots x_{\ell-1}$, then

$$\mathrm{RotL}_i(x) := x_i \cdots x_{\ell-1} x_0 \cdots x_{i-1} \,.$$

Now define $s_0 := 0^{\ell-2}11$, and for $0 < i < \ell$ define $s_i := \mathrm{RotL}_i(s_0)$. We will use the s_i to encode symbols from Σ.

Let $h : \Sigma^* \to \{0,1\}^*$ be the string homomorphism determined by

$$h(j) := s_{3j} ,$$

for all $0 \le j < k$. We extend h to apply to regexes over Σ in the usual way (see [10] for example).

Given a positive regex R over Σ, the subexpressions making up $F := f(k, R)$ come in four types—alignment, calibration, encoding, and duplication—defined as follows:

Alignment: Define

$$A := \mathbf{1}^\ell(\mathbf{0}^\ell)^+ .$$

Calibration: Define

$$C_0 := \mathbf{0001}^{\ell-3}(s_0)^+ ,$$
$$C_1 := \mathbf{01}^{\ell-1}(s_1)^+ ,$$
$$C_2 := \mathbf{01}^{\ell-1}(s_2)^+ ,$$

and for $3 \le i < \ell - 1$, define

$$C_i := \mathbf{1}^\ell(s_i)^+ .$$

Now define

$$C := \bigcup_{i=0}^{\ell-1} C_i .$$

Encoding: Define

$$E^{(R)} := s_0(h(R)) ,$$

that is, s_0 concatenated with the regex $h(R)$. Note that we make the dependence on R explicit. We use E as shorthand for $E^{(\Sigma^+)}$ and note that $L(E^{(R)}) \subseteq L(E)$, because R is positive.

Duplication: Define

$$D_0 := \bigcup_{1 \le c < k} s_{3c} ,$$

and for $j \in \{1, 2\}$, define

$$D_j := \bigcup_{0 \le c < k} s_{3c+j} .$$

Define

$$D := D_0(D_0)^+ \cup D_1(D_1)^+ \cup D_2(D_2)^+ .$$

Finally, define

$$F := \mathbf{1}(A \cup C) \cup \mathbf{0}(D \cup E^{(R)}) .$$

This completes the description of $F = f(k, R)$. It is evident that f is computable in the specified time bounds. Notice that all subexpressions of F except $E^{(R)}$ depend only on k and not on R.

Next we show how to convert any Σ-crossword solution X into a unique $\{0,1\}$-crossword solution $Y = \psi_k(X)$ such that, for any positive regexes T and U over Σ, X is a (T, U)-crossword solution if and only if Y is an (F, G)-crossword solution, where $F := f(k, T)$ and $G := f(k, U)$. It will

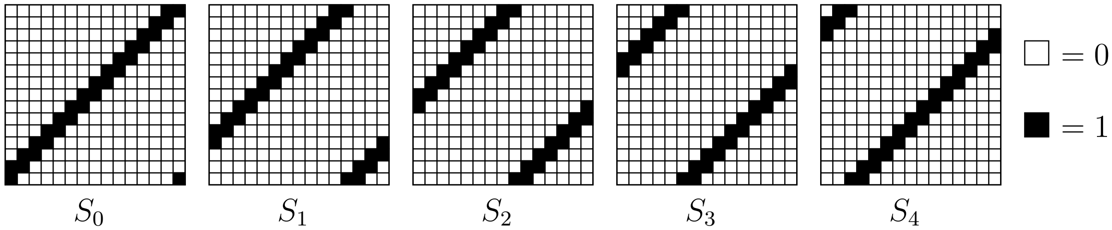


Figure 8: *The* 15×15 *squares* $S_0, \ldots, S_4$ *used to encode the individual letters* $0, \ldots, 4$*, respectively. A white cell denotes* 0*, and a black cell denotes* 1*. The columns of each successive square are cyclically shifted three spaces to the left from the previous square; likewise from* S_4 *to* S_0*.*

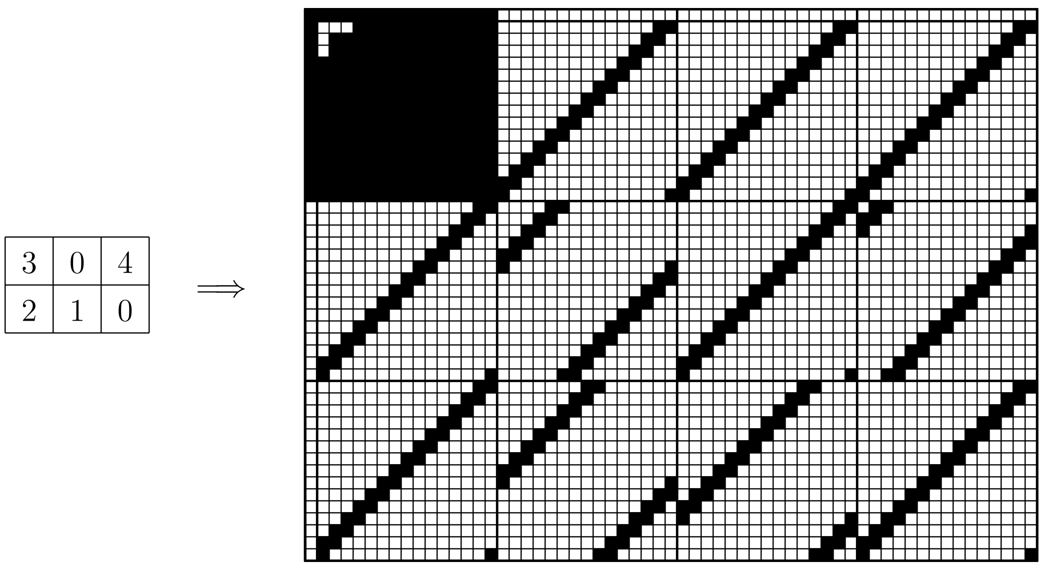


Figure 9: *The encoding* $\psi_5(X)$ *of a sample* 2×3 Σ*-grid* X*, where* $\Sigma = \{0, 1, 2, 3, 4\}$*. The slightly thicker lines give the boundaries between the* 15×15 *squares. The homomorphic image of the grid itself is in the six squares in the lower right.*

help first to see an example of how this is done. Suppose $\Sigma = \{0, 1, 2, 3, 4\}$. Then each cell of a crossword solution over Σ is encoded by a 15×15 square in the crossword solution over $\{0, 1\}$, as shown in Figure 8. Generally, for $0 \le c < k$ we define S_c be the $\ell \times \ell$ square whose ith row (starting with $i = 0$) is $s_{(3c+i) \bmod \ell}$. These squares are pairwise distinct, and we use S_c to encode the letter c. Notice that the S_c are symmetric (with respect to matrix transpose), and so the ith column of S_c is also $s_{(3c+i) \bmod \ell}$. In Figure 9, we show the encoding $\psi_k(X)$ of a sample 2×3 Σ-grid X. The top row and left column form the *alignment region*, and these two strings will both match $\mathbf{1}A$. The rest of the grid is made up of $(\ell \times \ell)$-size squares $Q_{t,u}$ for $t, u \ge 0$, with $Q_{0,0}$ being the top leftmost square, $Q_{0,1}$ immediately to its right, $Q_{1,0}$ immediately below it, etc. Squares of the form $Q_{0,u}$ and $Q_{t,0}$ form the *calibration region*, and, except for $Q_{0,0}$, all these squares are equal to S_0. The rows and columns making up this region all match $\mathbf{1}C$. The rest of the grid (squares $Q_{t,u}$ for $t, u \ge 1$) forms the *encoding region*, each square encoding a single corresponding entry in the Σ-grid. Rows and columns that intersect this region all match $\mathbf{0}(D \cup E)$.

Now the detailed description. Let X be *any* Σ-grid with m rows and n columns, where $m, n \ge 1$. For $1 \le t \le m$ and $1 \le u \le n$, let $x_{t,u}$ be the symbol in row t and column u of X. Then we define a $\{0,1\}$-grid $Y = \psi_k(X)$ as follows: Y has dimensions $((m+1)\ell+1) \times ((n+1)\ell+1)$, where $\ell = 3k$ as above. It will be convenient to index the rows of Y as $(-1), \ldots, (m+1)\ell - 1$ and the columns as $(-1), \ldots, (n+1)\ell - 1$. With this indexing, the alignment region comprises row (-1) and column (-1), and each square $Q_{t,u}$ (for $0 \le t \le m$ and $0 \le u \le n$) is the intersection of rows $t\ell, \ldots, (t+1)\ell - 1$ with columns $u\ell, \ldots, (u+1)\ell - 1$. We will define Y row by row, with rows $r_{-1}, \ldots, r_{(m+1)\ell-1}$, then discuss the columns. (It will help to refer back to Figure 9.)

- Set $r_{-1} := 1^{\ell+1}0^{n\ell}$. Then r_{-1} matches $\mathbf{1}A$.
- Set

$$
\begin{aligned}
r_0 &:= 10001^{\ell-3}(s_0)^n \ , \\
r_1 &:= 101^{\ell-1}(s_1)^n \ , \\
r_2 &:= 101^{\ell-1}(s_2)^n \ .
\end{aligned}
$$

  Then r_0, r_1, and r_2 match $\mathbf{1}C_0$, $\mathbf{1}C_1$, and $\mathbf{1}C_2$, respectively.
- For $3 \le i < \ell$, set $r_i := 1^{\ell+1}(s_i)^n$. Then r_i matches $\mathbf{1}C_i$.
- For $1 \le t \le m$, let $x := x_{t,1} \cdots x_{t,n}$. For $0 \le i < \ell$, set

$$
r_{t\ell+i} := 0 s_i s_{(3x_{t,1}+i) \bmod \ell} \cdots s_{(3x_{t,n}+i) \bmod \ell} \ .
$$

  Note that for $1 \le u \le n$, block u of $r_{t\ell+i}$ equals $\mathrm{RotL}_i(h(x_{t,u}))$. Also notice that if all the rows of X match some positive regex T over Σ, then all the $r_{t\ell}$ match $\mathbf{0}E^{(T)}$. The rest of the rows $r_{t\ell+i}$ match $\mathbf{0}D$; in particular, $r_{t\ell+i}$ matches $\mathbf{0}D_{i \bmod 3}$.

This completes the definition of the map ψ_k.

We have established that if the rows of X all match some positive regex T, then each row of Y matches $F = \mathbf{1}(A \cup C) \cup \mathbf{0}(D \cup E^{(T)})$, and from the arrangement of the rows, we can see by symmetry that if the columns of X all match some positive regex U over Σ, then each column of Y matches $\mathbf{1}(A \cup C) \cup \mathbf{0}(D \cup E^{(U)})$ in a similar manner:

- $c_{-1} = 1^{\ell+1}0^{m\ell}$, matching $\mathbf{1}A$.
- $c_0 = 10001^{\ell-3}(s_0)^m$, $c_1 = 101^{\ell-1}(s_1)^m$, and $c_2 = 101^{\ell-1}(s_2)^m$, matching $\mathbf{1}C_0$, $\mathbf{1}C_1$, and $\mathbf{1}C_2$, respectively.
- For $3 \le i < \ell$, $c_i = 1^{\ell+1}(s_i)^m$, matching $\mathbf{1}C_i$.
- For $1 \le u \le n$, letting $x := x_{1,u} \cdots x_{m,u}$, and for $0 \le i < \ell$, we have

$$
c_{u\ell+i} := 0 s_i s_{(3x_{1,u}+i) \bmod \ell} \cdots s_{(3x_{m,u}+i) \bmod \ell} \ .
$$

  That is, for $1 \le t \le m$, block t of $c_{u\ell+i}$ equals Since x matches U, we have that $c_{u\ell}$ matches $\mathbf{0}E^{(U)}$, and the rest of the $c_{u\ell+i}$ match $\mathbf{0}D$.

This establishes that, if X is a (T,U)-crossword solution, then $Y = \psi_k(X)$ is an (F,G)-crossword solution (of the correct size), where $F := f(k,T)$ and $G = f(k,U)$. It is also clear that, since Y has the original grid X completely encoded within it, ψ_k is a one-to-one map.

It remains to show that for any (F,G)-crossword solution Y, there is a (T,U)-crossword solution X such that $\psi_k(X) = Y$, where T, U, F, and G are as above. We establish this through a series of claims. Each claim is proved using "sudoku-like" arguments. Let Y be any (F, G)-crossword solution. First observe that any string w matching $A \cup C \cup D \cup E$ has length $v\ell$ for some $v \geq 2$, and so we can chop w into substrings of length ℓ that we call *blocks* (at least two), starting with block 0 through block $v-1$. This forces Y, minus its top row and left column, to be divided into $(\ell \times \ell)$-size squares $Q_{t,u}$ as described earlier, the rows and columns of each $Q_{t,u}$ being blocks in the rows and columns of Y that intersect $Q_{t,u}$. Y has squares $Q_{t,u}$ for each $0 \leq t \leq m$ and $0 \leq u \leq n$ for some $m, n \geq 1$. As before, we index the rows and columns of Y as $-1, \ldots, (m+1)\ell - 1$ and $-1, \ldots, (n+1)\ell - 1$, respectively.

We extend the block concept to strings of length $v\ell + 1$, e.g., the rows and columns of an (F, G)-crossword solution, by ignoring the first symbol in the string, that is, block 0 starts with the second symbol of the string.

Claim 6.7. *Each square $Q_{t,0}$ and $Q_{0,u}$ of Y, for $1 \leq t \leq m$ and $1 \leq u \leq n$, has exactly two 1's in each of its rows and each of its columns, the rest of the entries being 0.*

Proof of Claim 6.7. Let w be any string matching $A \cup C \cup D \cup E$. Then each block of w, other than block 0, has at *most* two 1's; in particular, it is either 0^ℓ (if w matches A), or it is of the form s_i for some i (if w matches $C \cup D \cup E$). Moreover, block 0 of w has at *least* two 1's. Thus for $1 \leq u \leq n$, square $Q_{0,u}$ has each of its rows containing at most two 1's and each of its columns containing at least two 1's. The only way this can happen is if each row and column of $Q_{0,u}$ contains *exactly* two 1's. A similar argument shows that each row and column of $Q_{t,0}$ contains exactly two 1's, for $i \leq t \leq m$. $\square$

Claim 6.8. *No row of Y other than the topmost, and no column of Y other than the leftmost, matches $\mathbf{1}A$.*

Proof of Claim 6.8. Consider any row except the topmost. This row is either $0r$ or $1r$ for some string r matching $A \cup C \cup D \cup E$, and it intersects either $Q_{0,1}$ or else $Q_{t,0}$ for some $t \geq 1$. In the former case, block 1 of r (i.e., the block of r intersecting $Q_{0,1}$) has a 1, and so r cannot match A; in the latter case, block 0 of r has a 0, and so again, r cannot match A. (Both cases follow from Claim 6.7.) Thus the row in question cannot match $\mathbf{1}A$. The same argument applies to the columns except the leftmost; none of them can match $\mathbf{1}A$. $\square$

Claim 6.9. *The topmost row and leftmost column of Y each match $\mathbf{1}A$.*

Proof of Claim 6.9. Observe that any string w matching C must have *at least three* 1's in its block 0. Now consider any row of Y that intersects square $Q_{1,0}$. This row is of the form $0r$ or $1r$, for some r matching $A \cup C \cup D \cup E$. By Claim 6.8, this row does not match $\mathbf{1}A$, and so it must match $\mathbf{1}C \cup \mathbf{0}(D \cup E)$. However, r cannot match C because (by Claim 6.7) r has only two 1's in block 0. Thus the row must match $\mathbf{0}(D \cup E)$—in particular, it starts with 0. That means that the leftmost column (column (-1)) has all 0's in its block 1, and so it cannot match $\mathbf{1}C \cup \mathbf{0}(D \cup E)$, and thus it must match $\mathbf{1}A$. A similar, transposed argument shows that the topmost row must also match $\mathbf{1}A$. $\square$

Claim 6.10. *Rows $0, \ldots, \ell - 1$ and columns $0, \ldots, \ell - 1$ of Y each match $\mathbf{1}C$, and the rows and columns of Y starting with index ℓ each match $\mathbf{0}(D \cup E)$.*

Proof of Claim 6.10. By the previous claim, the topmost row and leftmost column of Y each match $\mathbf{1}A = \mathbf{1}^{\ell+1}(\mathbf{0}^\ell)^+$. Thus rows $0, \ldots, \ell - 1$ each start with 1, and the rows starting with index ℓ each start with 0. By Claim 6.8, none of these rows match $\mathbf{1}A$, so rows 0 through $\ell - 1$ all must match $\mathbf{1}C$ and the rest must match $\mathbf{0}(D \cup E)$. A similar argument holds for the columns. $\square$

Notice that each row and column of $Q_{0,0}$ matches $(\mathbf{000}\cup\mathbf{011}\cup\mathbf{111})\mathbf{1}^{\ell-3}$. For $0 \le i < (m+1)\ell$, let r_i denote the row of Y with index i, and for $0 \le j < (n+1)\ell$ let c_j denote the column of Y with index j. Rows $r_0, \dots, r_{\ell-1}$ and columns $c_0, \dots, c_{\ell-1}$ all match $\mathbf{1}C$ by Claim 6.10, and the rest match $\mathbf{0}(D \cup E)$.

Claim 6.11. *For all $0 \le i < \ell$, r_i and c_i both match $\mathbf{1}C_i$.*

Proof of Claim 6.11. First we show that r_0 and c_0 both match $\mathbf{1}C_0$. Suppose that r_0 does not match $\mathbf{1}C_0$ (the argument for c_0 is similar). Then (since r_0 matches $\mathbf{1}C$) r_0 matches $\mathbf{1}C_i$ for some $i \ge 1$, and so has a prefix matching $\mathbf{1}(\mathbf{0}\cup\mathbf{1})\mathbf{1}^{\ell-1}$, which makes $c_1, \dots, c_{\ell-1}$ all have 11 as a prefix. This in turn implies that each of these columns must match $\mathbf{1}C_j$ for some $j \ge 3$. Now notice that block 1 of any string x matching C_j is s_j, and so if $3 \le j < \ell$, then x must have 0 as the next to last symbol in its block 1. From these facts it follows that the next to last row of $Q_{1,0}$ (i.e., block 0 of $r_{2\ell-2}$) matches $(\mathbf{0}\cup\mathbf{1})\mathbf{0}^{\ell-1}$. But this is impossible, because this block must have two 1's by Claim 6.7.

Next we show that r_1 and c_1 match $\mathbf{1}C_1$ and r_2 and c_2 match $\mathbf{1}C_2$. By what we just showed, r_1 r_2 both have prefix 10 (because c_0 matches $\mathbf{1}C_0$), and so they each match $\mathbf{1}(C_0 \cup C_1 \cup C_2)$. Neither of them can match $\mathbf{1}C_0$, however: Consider the 2×2 square S forming the intersection of rows $1, 2$ with columns $1, 2$. If either r_1 or r_2 matches $\mathbf{1}C_0$, then S contains a 0, and hence at least one of the columns c_1 or c_2 must also match $\mathbf{1}C_0$, which implies that S contains *all* 0's, which means that both r_1 and r_2 match $\mathbf{1}C_0$. But this would make $c_{2\ell-1}$ have prefix 0111 putting three 1's in the last column of $Q_{0,1}$ and contradicting Claim 6.7. (By a similar argument, neither c_1 nor c_2 can match $\mathbf{1}C_0$.) Thus we have r_1 and r_2 both matching $\mathbf{1}(C_1 \cup C_2)$. Now r_2 cannot match $\mathbf{1}C_1$, for if it does, then $c_{2\ell-2}$ has prefix either 0101 or 0111, neither of which is possible because block 0 of $c_{2\ell-2}$ must be s_j for some j. Thus r_2 matches $\mathbf{1}C_2$. We have one more case to eliminate, i.e., showing that r_1 cannot match $\mathbf{1}C_2$. Suppose r_1 matches $\mathbf{1}C_2$. Then column $c_{2\ell-2}$ has prefix 0100, and the only way this can happen is if $c_{2\ell-2}$ has prefix $0s_{\ell-1}$. But that means that row $r_{\ell-1}$ has a 1 as the next to last symbol of its block 1. Since $r_{\ell-1}$ matches $\mathbf{1}C$, this can only happen if $r_{\ell-1}$ matches $\mathbf{1}(C_0 \cup C_1)$, whence it has 10 as a prefix. This puts a 0 as the last symbol of block 0 of c_0, but this is impossible, because c_0 matches $\mathbf{1}C_0$ and hence has $10001^{\ell-3}$ as a prefix. Thus r_1 cannot match $\mathbf{1}C_2$, and so it matches $\mathbf{1}C_1$. A symmetric argument holds for c_1 and c_2.

Finally, we show that r_i matches $\mathbf{1}C_i$ for $3 \le i < \ell$. This is by induction on i, starting with $i = 3$, with the inductive hypothesis that r_j matches $\mathbf{1}C_j$ for all $0 \le j < i$. We have then that $c_{2\ell-i-1}$ has prefix 0^i1 and $c_{2\ell-i}$ has prefix $0^{i-1}11$. Since both of these columns match $\mathbf{0}(D \cup E)$ and hence must each start with $0s_j$ for some j's, we can only have that $c_{2\ell-i-1}$ has prefix 0^i11 and $c_{2\ell-i}$ has prefix $0^{i-1}110$. Then block 1 of r_i must be s_i, and it follows that r_i matches $\mathbf{1}C_i$. □

Claim 6.12. *$Q_{t,0} = Q_{0,u} = S_0$ for all $1 \le t \le m$ and $1 \le u \le n$.*

Proof of Claim 6.12. This follows immediately from Claim 6.11. □

Claim 6.13. *For each $1 \le t \le m$ and each $1 \le u \le n$, $r_{t\ell}$ matches $\mathbf{0}E^{(T)}$ and $c_{u\ell}$ matches $\mathbf{0}E^{(U)}$.*

Proof of Claim 6.13. By assumption, all rows of Y match $F = \mathbf{1}(A \cup C) \cup \mathbf{0}(D \cup E^{(T)})$, and all columns of Y match $G = \mathbf{1}(A \cup C) \cup \mathbf{0}(D \cup E^{(U)})$. By Claim 6.12, each row $r_{t\ell}$ and each column $c_{u\ell}$ has prefix $0s_0$, and thus none can match $\mathbf{1}(A \cup C) \cup \mathbf{0}D$. Thus each such row must match $\mathbf{0}E^{(T)}$, and each such column matches $\mathbf{0}E^{(U)}$. □

Claim 6.14. *For all t, u with $1 \le t \le m$ and $1 \le u \le n$, there exists a unique $x_{t,u} \in \Sigma$ such that $Q_{t,u} = S_{x_{t,u}}$.*

Proof of Claim 6.14. For simplicity, we will assume $t = u = 1$; the same argument works for any t, u. By Claim 6.10, rows $r_\ell, \ldots, r_{2\ell-1}$ and columns $c_\ell, \ldots, c_{2\ell-1}$ all match $\mathbf{0}(D \cup E)$. By Claim 6.12, the ith row of $Q_{1,0}$ (i.e., block 0 of $r_{\ell+i}$) is s_i, for $0 \le i < \ell$. We have r_ℓ matching $\mathbf{0}E$ by Claim 6.13. For $0 \le j < \ell$, let b_j be block 1 of $r_{\ell+j}$ (i.e., the jth row of $Q_{1,1}$), and let b'_j be block 1 of $c_{\ell+j}$ (i.e., the jth column of $Q_{1,1}$). Row r_ℓ matching $\mathbf{0}E$ makes $b_0 = s_{3x}$ for some unique $0 \le x < k$. For $1 \le i < \ell$, row $r_{\ell+i}$, having s_i as its block 0, cannot match $\mathbf{0}E$. Thus $r_{\ell+i}$ matches $\mathbf{0}D$, and in fact, it must match $\mathbf{0}D_{i \bmod 3}$, owing to its block 0, and this makes $b_i = s_{(3v+i) \bmod \ell}$ for some $0 \le v < k$. The same goes for the columns of $Q_{1,1}$. Furthermore, notice that D and E ensure that the columns b'_j and $b'_{(j-1) \bmod \ell}$ are distinct for any $0 \le j < \ell$, because j and $j-1$ have different remainders modulo 3.

We show by induction on $1 \le i < \ell$ that $b_i = s_{(3x+i) \bmod \ell}$, and this will imply that $Q_{1,1} = S_x$, finishing the proof of the claim. Now assume (inductive hypothesis) that $b_{i-1} = s_{(3x+i-1) \bmod \ell}$ (we have established this for $i = 1$). We have $b_i = s_{3v+i}$ for some $0 \le v < k$, and so it suffices to show that $v = x$. Suppose $v \ne x$. Then there is no position where the strings b_i and b_{i-1} share a 1 in common. The two 1's in b_{i-1} occur in columns b'_{z_1} and b'_{z_2} of $Q_{1,1}$, where $z_1 := (-3x - i) \bmod \ell$ and $z_2 := (z_1 - 1) \bmod \ell = (-3x - i - 1) \bmod \ell$, and by assumption, these 1's are then immediately followed by 0's in their respective columns. Since $b'_{z_1} = s_{j_1}$ and $b'_{z_2} = s_{j_2}$ for some $0 \le j_1, j_2 < \ell$, and they share the substring 10 in the same position in each, it must be that $j_1 = j_2$. But this contradicts what we said above about columns being distinct. Therefore, $v = x$, and we are done. $\square$

Claim 6.15. *For all $1 \le t \le m$ and $1 \le u \le n$, let $x_{t,u} \in \Sigma$ be the unique symbol such that $Q_{t,u} = S_{x_{t,u}}$ (cf. Claim 6.14). Then the $m \times n$ array X whose (t,u)th entry is $x_{t,u}$ forms a (T,U)-crossword solution.*

Proof of Claim 6.15. For $1 \le t \le m$, let $d_t := x_{t,1} \cdots x_{t,n}$, and for $1 \le u \le n$, let $e_u := x_{1,u} \cdots x_{m,u}$. We show that the d_t all match T and the e_u all match U. We have

$$r_{t\ell} = 0 s_0 s_{3x_{t,1}} \cdots s_{3x_{t,n}} = 0 s_0 (h(d_t)) ,$$

and because of the symmetry of the squares $Q_{t,u}$, we also have

$$c_{u\ell} = 0 s_0 s_{3x_{1,u}} \cdots s_{3x_{m,u}} = 0 s_0 (h(e_u)) ,$$

for all $1 \le t \le m$ and $1 \le u \le n$. By Claim 6.13, $r_{t\ell}$ matches $\mathbf{0}E^{(T)} = \mathbf{0}s_0(h(T))$ and $c_{u\ell}$ matches $\mathbf{0}E^{(U)} = \mathbf{0}s_0(h(U))$. Then because h is clearly a one-to-one map, it must be that d_t matches T and e_u matches u. $\square$

Finally, if X is as defined in Claim 6.15, then is clear by our definition of ψ_k above that $Y = \psi_k(X)$. This ends the proof of Lemma 6.6. $\square$

Proof of Theorem 6.5. Let $G := f(k, C)$, where C is as in Theorem 5.7, and k is the size of the alphabet used in that proof. $\mathsf{W}_{\{0,1\}}(G)$ is clearly c.e. For the other direction, we m-reduce from the problem $\mathsf{W}_\Sigma(C)$ of Theorem 5.7 via the map $f(k, \cdot)$. Given any positive regex R over a size-k alphabet, which we can assume is $\{0, \ldots, k-1\}$, we set $F := f(k, R)$. Then an (F, G)-crossword solution exists if and only if an (R, C)-crossword solution exists, by Lemma 6.6. $\square$

The next theorem is another corollary of Lemma 6.6. It strengthens Theorem 6.2. The problem $\mathsf{UR}_=\mathsf{C}_\Sigma$ was defined and shown undecidable for any alphabet Σ in Theorem 6.2.

Theorem 6.16. $\mathsf{UR}_=\mathsf{C}_{\{0,1\}}$ *is m-equivalent to the Halting Problem.*

Proof. This works as in the proof of Theorem 6.5. $\mathsf{UR}_=\mathsf{C}_{\{0,1\}}$ is evidently c.e. Conversely, we m-reduce from $\mathsf{UR}_=\mathsf{C}_\Sigma$ of Theorem 6.2. Given a positive regex E, we can effectively determine the size k of E's alphabet. Then adjusting the alphabet to $\{0,\ldots,k-1\}$, we let $E' := f(k,E)$, where f is the function of Lemma 6.6. Then E' is positive, and an (E',E')-crossword solution exists if and only if an (E,E)-crossword solution exists. $\square$

6.4 Making crosswords square

An $m \times n$ Σ-grid is *square* iff $m = n$. In this section, we explain briefly why the complexities of all our problems are unaffected by restricting all crossword solutions to be square.

First, in the proof of Lemma 5.6 in Appendix A, the R and C we construct are such that if an $m \times n$ (R,C)-crossword solution exists, then $m \geq n$. This is because each row records a configuration of the machine M, and each column records a tape cell *that is scanned at least once*, and M can only scan at most as many different tape cells as there are configurations. Thus to allow a square (R,C)-crossword solution, we only need to pad with (blank) cells that are never scanned. Letting $C' := C \cup [B]^+$, we get that an (R,C)-crossword solution exists if and only if an (R,C')-crossword solution exists, if and only if a square (R,C')-crossword solution exists.

Next, the map ρ of Lemma 6.1 clearly preserves squareness: every $m \times n$ (R,C)-crossword solution (for $m,n \geq 2$) maps to an $(m+1)\times(n+1)$ (E,E)-crossword solution and vice versa. Finally, the maps ψ_k of Lemma 6.6 also preserve squareness. An $m \times n$ (T,U)-crossword solution maps under ψ_k to a $(3k(m+1)+1)\times(3k(n+1)+1)$ (F,G)-crossword solution and vice versa.

7 Further Results

7.1 Controlling the number of solutions

Using the apparatus of Section 5, we can obtain a reduction from $\mathsf{3SAT}$ to RC that gives a one-to-one correspondence between satisfying assignments to a Boolean formula and solutions to the corresponding crossword. The following lemma tightens Lemma 3.3. Its proof is given in Appendix B.

Lemma 7.1. *There exist a polynomial p, a polynomial-time computable function r, and a positive regular expression C' over Σ such that, for any Boolean formula φ,*

1. *$R' := r(\varphi)$ is a positive regular expression over $\{0,1\}$,*
2. *(R',C') is plural,*
3. *every (R',C')-crossword is $q \times q$, where $q := q(|\varphi|)$, and*
4. *the number of (R',C')-crosswords is equal to the number of satisfying truth assignments to φ.*

Since the reductions of Lemma 3.3 and 7.1 control not just the existence but the *number* of crossword solutions, we can get more information out of them. We list a few other results here that follow easily from Lemma 3.3 or Lemma 7.1 or both.

- Counting the number of (R,C)-crossword solutions of given dimensions (given in unary) is polynomially equivalent to counting the number of satisfying assignments to a Boolean formula, and hence is complete for the class $\#\mathbf{P}$ [15]. More precisely, let $\#\mathsf{RC}$ be the function that takes an (R,C)-crossword as input and returns the number of solutions. Lemmas 3.3 and 7.1 imply $\#\mathsf{RC}$ is hard for $\#\mathbf{P}$. Since $\#\mathsf{RC} \in \#\mathbf{P}$, we have that $\#\mathsf{RC}$ is complete for $\#\mathbf{P}$.

- As with sudoku puzzles, someone who wants to solve a regex crossword puzzle (found online or in a newspaper, say) should reasonably expect that a solution exists and is unique. Does the promise of a unique solution make solving the puzzle any easier in the worst case? The answer is no, at least with respect to randomized polynomial reductions. Consider the following search problem:

  Input: Regular expressions R and C, and integers $m, n \geq 1$ in unary.
  Promise: A unique $m \times n$ (R, C)-crossword solution exists.
  Ouput: The $m \times n$ (R, C)-crossword solution.

  Lemma 7.1 and its proof says that this problem is polynomially equivalent to finding the unique satisfying assignment to a Boolean formula with the promise that it is uniquely satisfiable. The latter problem is known to be **NP**-hard with respect to randomized polynomial reductions [16].

- Shifting perspective from the last item, a regex crossword puzzle *maker* may want a test to determine, given regular expressions R and C and $m, n \geq 1$ in unary, whether or not a unique solution exists. Lemma 7.1 says that this is polynomially equivalent to USAT, the language of all uniquely satisfiable Boolean formulas. USAT is known to be **NP**-hard (it is in the class $\mathbf{D}^p$, the first level of the difference hierarchy over **NP**).

Finally, the techniques of Section 5 can be modified easily to show that if the dimensions of the crossword are both given in *binary* instead of unary, then the (R, C)-crossword solution existence problem is complete for **NEXP** (nondeterministic exponential time) under polynomial reductions. If one of the dimensions is given in unary and the other in binary, then the problem becomes **PSPACE**-complete. (**PSPACE**-hardness follows from Lemma 5.6; membership in **PSPACE** follows by modifying slightly the proof of Theorem 6.4.)

7.2 String-based puzzles

In a standard regex crossword puzzle, each cell of the grid contains a single letter from the alphabet. A variant puzzle allows each cell to contain an arbitrary string of characters. Then, the concatenation of the strings along each row must match the corresponding row regex, and similarly for the columns. One of course can consider the various alterations on this puzzle we have described previously: bounded versus unbounded, limits on the alphabet size, equality of regexes, and puzzle versus two-player game. In this section we give upper and lower bounds on the complexity of an unrestricted (but still bounded) version of the puzzle.

Definition 7.2. Let Σ be an alphabet. A *string-based regex crossword* over Σ is pair

$$P := \langle\langle R_1, \ldots, R_m\rangle, \langle C_1, \ldots, C_n\rangle\rangle$$

for some $m, n \geq 1$ and regexes $R_1, \ldots, R_m$ and $C_1, \ldots, C_n$ over Σ. A *solution* to P is a map $s : [m] \times [n] \to \Sigma^*$ such that, for all $1 \leq i \leq m$ and $1 \leq j \leq n$,

- $s(i, 1) \| \cdots \| s(i, n)$ matches R_i, and
- $s(1, j) \| \cdots \| s(m, j)$ matches C_j.

We define StrRC_Σ to be the language of all solvable string-based regex crosswords over Σ.

Proposition 7.3. $\mathsf{StrRC}_\Sigma \in \mathbf{PSPACE}$ *for any alphabet* Σ.

Proof sketch. Given regexes $R_1, \ldots, R_m$ and $C_1, \ldots, C_n$, we first convert them to equivalent ϵ-NFAs $\tilde{R}_1, \ldots, \tilde{R}_m$ and $\tilde{C}_1, \ldots, \tilde{C}_n$, respectively. We then nondeterministically guess strings for each cell in the following (row-major) order: $w_{11}, w_{12}, \ldots, w_{1n}, w_{21}, w_{22}, \ldots, w_{2n}, \ldots, w_{m1}, \ldots, w_{mn}$, where w_{ij} is the contents of the cell at row i column j. While guessing strings, we simulate the NFAs using the standard set-of-states method. We do this in the following way: On string w_{ij}, we simulate $\tilde{R}_i$ and $\tilde{C}_j$. If $i = 1$, we simulate $\tilde{C}_j$ from the start; if $j = 1$, we simulate $\tilde{R}_i$ from the start. If $i > 1$, we continue to simulate $\tilde{C}_j$ from where we left off after guessing $w_{i-1,j}$, and if $j > 1$, we simulate $\tilde{R}_i$ starting from where we left off after guessing $w_{i,j-1}$. In any case, we always save the simulation results for future use. After guessing w_{in} (respectively, w_{mj}) we check whether $\tilde{R}_i$ (respectively, $\tilde{C}_j$) accepts. If either reject, then we reject; if no NFAs have rejected after guessing w_{mn}, then we accept.

This approach decides membership in StrRC_Σ, and it can be done in nondeterministic polynomial space, because we only need to keep track of the sets of states of the various NFAs. Further, the string w_{ij} need be no longer than $2^{r_i+c_j}$, where r_i and c_j are the sizes of the state sets of $\tilde{R}_i$ and $\tilde{C}_j$, respectively, and thus we can stop guessing w_{ij} and move on to the next string after at most this many steps. This length bound suffices to allow any combination of state sets of the two automata.

The proposition then follows by Savitch's theorem. □

8 Open Problems

The most immediate question arising from our work is whether RCG is **PSPACE**-hard restricted to a binary alphabet. Our proof shows only that it is **PSPACE**-hard for a ternary alphabet. Doing without the third symbol "2" in the alphabet currently seems like a daunting task, despite the fact that under normal play, that symbol appears only once in the upper left-hand corner.

Another question is whether we still get **PSPACE**-hardness if we restict the regexes R and C to be equal to each other. If one can show **PSPACE**-hardness for RCG' restricted so that $R_i = C_i$ for all i, then it may be easy to get $R = C$ for the constructed instance of RCG, since these two latter regexes are close to being equal anyway.

Theorem 5.7 gives undecidability for a particular fixed expression C. One may ask more generally: For which C is the corresponding problem undecidable? How hard is it to determine, given a C, whether the corresponding problem is decidable? We conjecture that this latter question is m-complete for Σ_3, the third Σ-level of the arithmetic hierarchy (see, e.g., [13]). Similar questions can be asked about Proposition 3.10. For example: For which C is the question (i) **NP**-hard; (ii) in **P**?

8.1 Variants of two-player regex crossword games

One can imagine a variety of two-player games involving regex crosswords besides the ones considered in this paper, and some of these may actually be fun to play. Recall the RCG' game described in Section 4.2:

> A blank $m \times n$ grid is given to start, along with regexes $R_1, \ldots, R_m$ and $C_1, \ldots, C_n$. Player 1 (Rose) fills in the first row to match R_1, then Player 2 (Colin) fills in the rest of the first column so that it matches C_1, then Rose fills in the rest of row 2 so that it matches R_2, then Colin column 2, etc.

For example:

1. Same as the RCG' game above, but each player can choose an incomplete row (respectively column) to fill in on each turn.

2. Same as the RCG' game, but both players alternately fill in rows in order, and a move is legal iff each column can be completed to match its corresponding C_j (this may or may not be easy to determine).

3. Same as in the last item, but a player can choose a row to fill in on their turn.

In all these games, the last player able to make a legal move wins. We conjecture that for all these games, determining whether Rose has a winning strategy is **PSPACE**-hard, even if all the R_i are equal and all the C_j are equal and independent of the input, or if all the R_i and C_j are equal to each other. (It is straightforward to prove that all these problems are in **PSPACE**.)

One might also consider some unbounded versions of these games:

1. Positive regexes R and C are given, but the size of the grid is not. Rose first chooses an *arbitrary* string r_1 matching R for the first row of the grid (thus fixing the number of columns). Colin then chooses an arbitrary string c_1 matching C for the first column of the grid (except the first symbol of c_1 must equal that of r_1), thus fixing the number of rows. Players then proceed as in the games mentioned previously.

2. Same as the last item, but on their first move, each player chooses a string r (respectively c) and says which row (respectively column) this string is to fill.

The first two moves in each of these games is unbounded, but thereafter, the grid dimensions are fixed, and so determining the winner under optimal play is decidable, *given the first two moves*. The problem of determining if Rose wins *without* knowing the first two moves is then in the class Σ_2, the second Σ-level of the arithmetic hierarchy (i.e., it is c.e. relative to the Halting Problem). We conjecture that it is m-complete for this class.

Acknowledgments

We would like to thank Joshua Cooper for finding for us a particularly challenging and fun three-way regex crossword puzzle in [5]. We also thank Klaus-Jörn Lange, who pointed out the connection between our work and the theory of two-dimensional picture languages, and George McNulty, who gave helpful suggestions for improving the proof and presentation of our main result regarding (R, C)-games. We are also grateful for a number of students in the first author's Theory of Computation class who (independently) suggested the variation of the regex crossword puzzle given in Definition 7.2. The first author also thanks Jason O'Kane for first suggesting to him the **NP**-completeness question for regex crosswords as an exercise. Much of this work was done at the Dagstuhl seminar 14391, "Algebra in Computational Complexity." Some of this work was also done while the first author visited the third author at the University of Ulm (Germany), and the first author would like to thank the Dagstuhl organizers and the University of Ulm for their hospitality.

References

[1] Https://regexcrossword.com

[2] MIT Mystery Hunt, http://www.mit.edu/ puzzle

[3] (February 2013), slashdot discussion, http://games.slashdot.org/story/13/02/13/2346253/can-you-do-the-regular-expression-crossword

[4] Berger, R.: The Undecidability of the Domino Problem. No. 66 in Memoirs of the American Mathematical Society, American Mathematical Society, Providence, Rhode Island (1966), mR0216954

[5] Black, L.: Can you do the regular expression crossword? I Programmer (February 2013), http://www.i-programmer.info/news/144-graphics-and-games/5450-can-you-do-the-regular-expression-crossword.html

[6] Fenner, S.: The complexity of some regex crossword problems (2014)

[7] Fenner, S., Padé, D.: Complexity of regex crosswords. In: Martín-Vide, C., Okhotin, A., Shapira, D. (eds.) Language and Automata Theory and Applications. pp. 215–230. Springer International Publishing, Cham (2019)

[8] Giammarresi, D., Restivo, A.: Recognizable picture languages. International Journal of Pattern Recognition and Artificial Intelligence pp. 31–46 (1992)

[9] Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Salomaa, A., Rosenberg, G. (eds.) Handbook of Formal Languages, vol. 3, chap. 96, pp. 215–267. Springer-Verlag (1997)

[10] Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Pearson, 3rd edn. (2007)

[11] Latteux, M., Simplot, D.: Recognizable picture languages and domino tiling. Theoretical Computer Science **178**(1-2), 275–283 (1997), note

[12] Sipser, M.: Introduction to the Theory of Computation. Cengage Learning, 3rd edn. (2013)

[13] Soare, R.I.: Recursively Enumerable Sets and Degrees. Perspectives in Mathematical Logic, Springer-Verlag, Berlin (1987)

[14] Takahashi, G.: Are regex crosswords NP-hard? CS Stack Exchange question 30143, answered by FrankW (2014)

[15] Valiant, L.: The complexity of computing the permanent. Theor. Comput. Sci. **8**, 189–201 (1979)

[16] Valiant, L., Vazirani, V.: NP is as easy as detecting unique solutions. Theor. Comput. Sci. **47**, 85–93 (1986)

A Proof of Lemma 5.6

In this appendix, we prove Lemma 5.6 from Section 5:

Lemma A.1. *Let M be a Turing machine (as described above). There exists an alphabet Σ and a regex $C := C(M)$ over Σ (Σ and C both depending on M), and for any input string w there exists a regex $R := R(M, w)$ over Σ (depending on M and w) such that (R, C) is plural, and M halts on input w if and only if an (R, C)-crossword solution exists, and if this is the case, then*

- *the (R, C)-crossword solution is unique, and*

- *there is a constant c, independent of M and w, such that the unique solution is a grid with between $t + 2|w|$ and $t + 2|w| + c$ rows and between* $\max(s, |w|)$ *and* $\max(s, |w|) + c$ *columns, where t (respectively s) is the number of steps M takes (respectively, the number of cells M ever scans) on input w.*

Furthermore, R is computable from M and w in polynomial time, and C is computable from M.

Recall that our computational model is that of a deterministic Turing machine with a unique halting state (distinct from the start state) and a single one-way infinite tape whose initial contents starts with blank symbols in the two left-most cells, followed by an input string w of nonblank symbols, followed on the right with blank tape. In each step, the tape head must move either left or right by one cell. We view a computational tableau with the initial configuration on the top row and time moving downward.

Proof of Lemma 5.6. Let $M := (Q, \Gamma, \delta, q_0, q_{\text{halt}}, B)$, where

- Q is the (finite) state set,
- $q_0 \in Q$ is the start state,
- $q_{\text{halt}} \in Q$ is the halting state, different from q_0 (M halts just when this state is entered),
- Γ is the tape alphabet,
- $B \in \Gamma$ is the blank symbol, and
- $\delta : (Q \setminus \{q_{\text{halt}}\}) \times \Gamma \to Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\}$ is the transition function. The left and right head directions are indicated by L and R, respectively.

Given some input string $w \in (\Gamma \setminus \{B\})^*$, we construct the two regexes R and C over an alphabet Σ (defined below). The expression C only depends on M and not on w. For technical convenience and without loss of generality, we will modify the state set Q and transition function δ of M if necessary to obtain a Turing machine $\widehat{M}$ with the following three properties: $\widehat{M}$'s first computational step is governed by the transition $\delta(q_0, B) = (q_1, B, \mathsf{L})$ for some state $q_1 \neq q_0$ (that is, $Bq_0Bw \mapsto q_1BBw$); $\widehat{M}$ then scans across the entire input w and back (going through configuration $BBwq_2B$ and ending at Bq_3Bw for some states q_2 and q_3), at which point it simulates M step for step; $\widehat{M}$ never re-enters state q_0 after its first step, nor attempts to move left when scanning the leftmost cell of the tape (it might write a special symbol in the leftmost cell to keep itself from doing this). Clearly, $\widehat{M}$ can be constructed from M in polynomial time, and on any input, $\widehat{M}$'s halting *versus* non-halting behavior is the same as M's and its time and space usage are roughly the same as M's.[7] We do not refer to the original M again in the rest of the proof, so we will re-use Q and δ to denote respectively the state set and transition function of $\widehat{M}$ without risking of ambiguity.

To avoid confusion, we will call the elements of the alphabet Σ *markers*, reserving the word *symbol* to refer to elements of Γ. The markers in Σ are of the following three disjoint types:

Unscanned tape cell markers: For all $a \in \Gamma$, the marker $[a]$ is in Σ. Each of these markers is used to depict a cell of the tape containing the symbol a and which is scanned neither

[7] $\widehat{M}$ scans the entire input w twice before simulating M, hence the appearance of $t + 2|w|$ in the expression for the number of rows of the crossword solution.

currently nor in the next time step. We let $U := \{[a] : a \in \Gamma\}$ denote the set of all unscanned tape cell markers.[8]

Scanned tape cell markers: For all $a \in \Gamma$ and all $q \in Q$, the marker $[a, q]$ is in Σ. Each of these depicts a cell of the tape containing a that is currently being scanned, and $\widehat{M}$'s current state is also included in the marker.

State transmission markers: For all $a \in \Gamma$ and all $q \in Q \setminus \{q_0\}$, the marker $[a, \downarrow q]$ is in Σ. These markers depict tape cells that are currently unscanned but will be scanned in the next time step (and so they always appear horizontally adjacent to scanned tape markers for nonhalting states). $\widehat{M}$'s state in the *next* time step is also included in the marker.

To summarize: At each time step of $\widehat{M}$'s computation, the tape cell scanned by the head is recorded in the crossword solution by the corresponding scanned tape cell marker, which includes $\widehat{M}$'s current state. All the unscanned cells of $\widehat{M}$'s tape are recorded in the solution by their corresponding unscanned tape cell markers with one exception: the unscanned tape cell that will become scanned in the next time step will be recorded by a state transmission marker, which includes $\widehat{M}$'s state in the next time step.

Here are two typical examples. Suppose $\widehat{M}$'s current state is q and it is scanning a b on the tape, with a to the left and c to the right. The corresponding configuration is traditionally denoted $\cdots aqbc \cdots$. If $\delta(q, b) = (r, x, \mathsf{R})$, then the part of the crossword solution corresponding to the transition $aqbc \mapsto axrc$ looks like this:

$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$[a]$	$[b, q]$	$[c, \downarrow r]$	$\cdots$
$\cdots$	$[a]$	$[x]$	$[c, r]$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

If instead, $\delta(q, b) = (s, y, \mathsf{L})$, then we get this for the transition $aqbc \mapsto sayc$:

$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$[a, \downarrow s]$	$[b, q]$	$[c]$	$\cdots$
$\cdots$	$[a, s]$	$[y]$	$[c]$	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

The one exception to this rule is a halting configuration, say $\cdots aq_{\text{halt}}bc \cdots$, which is represented in the crossword solution thus:

$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\cdots$	$[a]$	$[b, q_{\text{halt}}]$	$[c]$	$\cdots$

We will guarantee that there can be no rows of the solution below this one.

The regex R

R ensures that all the rows of the crossword solution look like they should. First we define a regex giving the initial configuration of $\widehat{M}$ on input w: Let $w = w_1 w_2 \cdots w_n$, where $n \geq 0$ and each w_i is in $\Gamma \setminus \{B\}$. Define

$$I_w := [B, \downarrow q_1][B, q_0][w_1][w_2] \cdots [w_n][B]^+ \,. \tag{20}$$

[8] From now on, we will identify a finite set of strings with the regex that matches exactly the strings in the set. For example, we use U as a regex in the sequel.

This is the only component of our construction that depends on the string w. Since in its first step $\widehat{M}$'s head moves left and its state becomes q_1, this is the correct description of the first row. Since there are no cells further to the left, we can take $[B, \downarrow q_1]$ to start I_w. Next, we define strings of markers indicating configurations beyond the initial one. Set

$$\begin{aligned}
T_{\mathsf{L}} &:= \{[b, \downarrow r][a, q] : a, b \in \Gamma \wedge q \in Q \setminus \{q_0, q_{\text{halt}}\} \wedge (\exists c \in \Gamma)\delta(q, a) = (r, c, \mathsf{L})\} , \\
T_{\mathsf{R}} &:= \{[a, q][b, \downarrow r] : a, b \in \Gamma \wedge q \in Q \setminus \{q_0, q_{\text{halt}}\} \wedge (\exists c \in \Gamma)\delta(q, a) = (r, c, \mathsf{R})\} , \\
T &:= T_{\mathsf{L}} \cup T_{\mathsf{R}} \cup \{[a, q_{\text{halt}}] : a \in \Gamma\} ,
\end{aligned}$$

describing portions of the tape in the vicinity of the scanned cell, undergoing transitions. Then finally we define the row regex

$$R := I_w \cup U^* T U^* ,$$

where we recall that U matches any single unscanned tape cell marker. Note that R requires each row to include exactly one scanned tape cell marker. If the corresponding state is nonhalting, then it is adjacent to some state transmission marker (and this is the only place the latter marker can appear in the row). If the corresponding state is halting, then there is no state transmission marker on the row.

Clearly, R is positive and computable in polynomial time given w and a description of $\widehat{M}$.

The regex C

C ensures that all the columns of the crossword look like they should. We define

$$C := S \cap W$$

as the intersection of two subexpressions: S ensures that each tape cell stays constant ("static")—except just after it is scanned by $\widehat{M}$'s head—and that when a cell becomes scanned, the new state information is faithfully copied from the previous time step (via the state transmission marker in the previous row); W (for "written") ensures that the correct symbol is written into a scanned cell on the next time step.

For S we define

$$\begin{aligned}
D &:= \bigcup_{a \in \Gamma,\, q \in Q \setminus \{q_0\}} [a]^* [a, \downarrow q][a, q] , \\
E &:= \bigcup_{a \in \Gamma,\, q \in Q \setminus \{q_0\}} [a]^+ [a, \downarrow q][a, q] , \\
F &:= \bigcup_{a \in \Gamma} [a]^* , \\
S &:= (E \cup [B, q_0] \cup [B, \downarrow q_1][B, q_1]) \, D^* F .
\end{aligned}$$

A string matching $E \cup [B, q_0] \cup [B, \downarrow q_1][B, q_1]$ gives the contents of a tape cell starting at the beginning up through the first time it is scanned. Thereafter, each string matching D represents a time interval ending with the cell being scanned again. F is matched by the cell's contents after the last time it is scanned. Note that S is positive, and hence C is positive.

For W we define (with explanation afterwards)

$$\begin{aligned}
X &:= \{[a,q][b] : a \in \Gamma \wedge q \in Q \setminus \{q_{\text{halt}}\} \wedge (\exists r \in Q)(\exists d \in \{\mathsf{L},\mathsf{R}\})[\delta(a,q) = (r,b,d)]\} \ , \\
Y &:= \{[a,q][b,{\downarrow}s] : a \in \Gamma \wedge q \in Q \setminus \{q_{\text{halt}}\} \wedge s \in Q \setminus \{q_0\} \wedge (\exists r \in Q)(\exists d \in \{\mathsf{L},\mathsf{R}\})[\delta(a,q) = (r,b,d)]\} \ , \\
H &:= \{[a,q_{\text{halt}}] : a \in \Gamma\} \ , \\
Z &:= \Sigma \setminus \{[a,q] : a \in \Gamma \wedge q \in Q\} \ , \\
W &:= Z^*(XZ^* \cup Y)^*H? \ .
\end{aligned}$$

X and Y both match a tape cell's contents in two adjacent time steps, starting when the cell is being scanned. The difference is that X must be used for the case where the tape head moves away *but does not immediately return to the cell in the next step*; Y is used for the case where the head moves away then immediately reverses direction back to the cell (hence the state transmission marker). Z matches any marker except a scanned tape cell marker. Thus, $XZ^* \cup Y$ depicts an interval of time starting when a cell is scanned up until, but not including, the next time it is scanned (or else through the end of the computation). H is used only if the cell is scanned when $\widehat{M}$ halts.

We have that W matches all strings in which any occurrence of a non-halting scanned tape cell marker is immediately followed by either an unscanned tape cell marker—or state transmission marker—giving the cell's correct contents after the corresponding transition of $\widehat{M}$.

Notice that C is computable from $\widehat{M}$ alone and does not depend on the input string w at all. Note that we are *not* asserting that C is computable in polynomial time. Our description of C includes the intersection operator $\cap$, which is not part of the formal syntax of regexes. As we mentioned, one can effectively compute an equivalent regex without the $\cap$ operator, but it may be exponentially larger.

Correctness

One direction of the lemma is now fairly clear from the previous discussion: If $\widehat{M}$ halts starting with w on its tape, then an (R,C)-crossword solution exists. Such a solution reflects the computational trace of $\widehat{M}$ on input w.

For the other direction, suppose X is an (R,C)-crossword solution. Let $r_1,\dots,r_m \in \Sigma^*$ and $c_1,\dots,c_n \in \Sigma^*$ be the rows and columns of X, respectively, for some $m,n \geq 1$. S ensures that r_1 matches $(U \cup [B,q_0] \cup [B,{\downarrow}q_1])^*$, and since R forces r_1 to contain a scanned tape cell marker somewhere, that marker must be $[B,q_0]$. It follows that r_1 does not match U^*TU^*, and so it matches I_w, providing the right starting configuration for $\widehat{M}$ (and ensuring that $n \geq 2$). We also have $m \geq 2$, ensured by S because r_1 contains $[B,{\downarrow}q_1]$. Thus (R,C) is plural. Subsequent rows must then conform to $\widehat{M}$'s computation, as was described previously.

We claim that the last row r_m must contain a marker of the form $[a,q_{\text{halt}}]$ for some $a \in \Gamma$, indicating that $\widehat{M}$ halts. This is because R ensures that r_m contains some scanned tape cell marker, and supposing this marker is of the form $[a,q]$ for some $q \neq q_{\text{halt}}$, there must be a state transmission marker on either side of it in r_m, whence S ensures that this latter marker is followed by a scanned tape cell marker in its column, which means r_m could not have been the last row.

Finally, as we showed that any solution corresponds to the (unique) halting computation of $\widehat{M}$ on input w, we establish uniqueness of the solution by observing that the dimensions of the solution are uniquely determined by this computation: R makes sure that each row contains *exactly* one scanned tape cell marker, and S makes sure that every column contains *at least* one scanned tape cell marker, and so the columns of the solution exactly correspond to the tape cells that are scanned at least once by $\widehat{M}$ (which, by construction, include the entire input string w). Furthermore, any row containing a marker of the form $[a, q_{\mathrm{halt}}]$ (for some $a \in \Gamma$) must be the last row—this is enforced by W. It follows from all this that the dimensions of the solution are uniquely determined by $\widehat{M}$'s computation and are as given in the lemma: those dimensions reflect the time and space usage of $\widehat{M}$ up to an additive constant. $\square$

B Proof of Lemma 7.1

Proof. We modify slightly the proof of Lemma 5.6 applied to a Turing machine M such that, on any input w of length n:

1. M's tape alphabet contains (at least) the nonblank symbols 0 and 1 and blank symbol B,
2. M's computation satisfies the technical conditions given at the start of that proof with respect to w,
3. if w encodes some Boolean formula φ with variables $x_0, \ldots, x_{k-1}$ for some $k \le n$, then for any $a \in \{0,1\}^k$, with wBa initially on its tape, M scans wBa in its entirety and halts if and only if a is a satisfying truth assignment for φ, and
4. if M halts, then it halts after *exactly* $p(n) - 1$ many steps (thus including $p(n)$ many configurations), for some appropriately chosen polynomial p with integer coefficients, independent of w, such that $p(n) \ge 2n + 3$ for all $n \ge 0$.

Such a machine M and polynomial p clearly exist. Under these assumptions, we can change the definition of I_w in Equation 20 to accommodate the presence of a on the tape:

$$I_w := [B, \downarrow q_1][B, q_0][w_1] \cdots [w_n][B]([0] \cup [1])^k [B]^{p(n)-n-k-3} ,$$

provided $w = w_1 \cdots w_n$ encodes a Boolean formula with $k \le n$ variables. Note that I_w is only matched by strings of length $p(n)$. The rest of the definition of R remains the same. We also modify C just as we did in Section 6.4: $C'' := C \cup [B]^+$, where C is as in the proof Lemma 5.6. Under these modifications, both R and C'' remain positive. Now setting $p := p(n)$, we observe that for any w encoding a Boolean formula φ with $k \le n$ variables,

$$\begin{aligned} \varphi \text{ is satisfiable} &\iff M \text{ halts on } wBa \text{ for some } a \in \{0,1\}^k \\ &\iff \text{an } (R, C'')\text{-crossword exists,} \end{aligned}$$

and if such is the case, then owing to the determinism and running time of M, the (R, C'')-crossword is unique, is of size $p \times p$, and both w and a are easily recoverable from it, which implies that the number of (R, C'')-crosswords is equal to the number of satisfying assignments to φ. Also by Lemma 5.6, given φ we can compute R, C, and 0^p all in polynomial time.

Finally, we apply the function f of Lemma 6.6 to both R and C''. Let Σ be the alphabet of R and C'' (cf. Lemma 5.6). By renaming if necessary, we may assume that $\Sigma = \{0, \ldots, \ell - 1\}$ for some ℓ. Then we set

$$\begin{aligned} q &:= 3\ell(p+1) + 1 , \\ R' &:= f(\ell, R) , \\ C' &:= f(\ell, C'') , \\ r(\varphi) &:= R' . \end{aligned}$$

Any (R', C')-crossword thus has exactly $q = 3\ell(p + 1) + 1$ rows and columns. The expressions R' and C' are both positive by Lemma 6.6, and so (R', C') is plural, because $q \geq 2$. Finally, since f is polynomial-time computable (with constant ℓ), so is r, and since f preserves the number of crosswords, the number of $q \times q$ (R', C')-crosswords equals the number of $p \times p$ (R, C'')-crosswords, which equals the number of assignments satisfying φ. $\square$