Complexity of Regex Crosswords

Stephen Fenner Daniel Padé

February 2019

Abstract

In a regular expression crossword puzzle, one is given two non-empty lists $\langle\langle R_1, \ldots, R_m\rangle$ and $\langle C_1, \ldots, C_n\rangle\rangle$ 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 in the j^{th} column is in $L(C_j)$. 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.

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 the squares with letters 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.

Regex crosswords have enjoyed some recent popularity, having been discussed in several popular media sources [AA13, Bla13], and thanks to some websites where people can solve the puzzles online [AAa, AAb]. Some variants of the basic puzzle have also been posed [AAc].

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

In this paper, we consider two 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; and (2) a 2-player game where players take turns attempting to fill in successive rows and columns of the grid. 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 3 and 4 for precise definitions). Our main result 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 [Fen14],[2] but the polynomial reduction used there was indirect and needlessly complicated for its purpose. As a warm-up to our main result, we give a simple, straightforward polynomial reduction from **3SAT** to **RC**.

In the spirit of 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] Glen Takahashi posted this question to Stack Exchange in 2012 [Tak14a], but it has been asked by others independently.

[2] In the same paper, a restriction of **RC** where the unique row and column regexes are equal to *each other* was also shown **NP**-complete.

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" [Ber66], 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 [GR97]. 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 [GR97, 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 [GR92] (cf. [GR97, 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 $\mathcal{T}$. The existence problem for recognizable picture languages ("Given a tiling system, does it define a nonempty language?") is known to be undecidable ([GR97, Theorem 9.1]), and so, putting these results together, we get that the existence problem for unbounded (R, C)-crosswords is undecidable as well. A much more direct reduction from the halting problem to unbounded (R, C)-crossword existence was given in [Fen14], where it was also shown 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.

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. We give open problems in Section 5.

2 Preliminaries

We fix an alphabet Σ once and for all and assume it contains the symbols 0 and 1 at least. For the **NP**-completeness result of Section 3, one can assume that $\Sigma = \{0, 1\}$. For the **PSPACE**-completeness result of Section 4, it suffices that $\Sigma = \{0, 1, 2\}$.

2.1 3SAT

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

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

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

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

The question is, is φ true (is it *satisfied*) for some assignment of the variables. This is the canonical complete problem for **NP**. In Section 3 we show that the language **RC** — the language of (R, C)-crosswords — is **NP**-complete by giving reduction from **3SAT**.

2.2 TQBF

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

$$\varphi \coloneqq \exists x_0 \forall y_0 \cdots \exists x_{k-1} \forall y_{k-1} \exists x_k \tilde{\varphi}(x_0, y_0, \ldots, x_{k-1}, y_{k-1}, x_k) \tag{1}$$

where $\tilde{\varphi}$ is a quantifier-free Boolean formula which can be assumed to be in conjunctive normal form with c clauses and $2k+1$ variables, for some positive c and k.

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. [3] 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, **RCG** — the language of (R,C)-crossword games (defined below) with a winning strategy for the first player — is **PSPACE**-complete by reduction from **TQBF**.

3 (R,C)-crosswords

For two given regexes R and C over Σ, an (R, C)-*crossword solution* is a two-dimensional m by n grid of symbols from the alphabet. Interpreting rows and columns as strings, each row must match R and each column must match C.

An (R, C)-*crossword* is represented as a 4-tuple $\langle 0^m, 0^n, R, C\rangle$ where the number of rows and columns are given in unary as m and n, and R and C are row and column regexes over Σ (defined in the usual way, using the operators $\cup$, $\|$, $*$, where $\|$ or juxtaposition both indicate concatenation).

Definition 1. The language **RC** is the set of all (R, C)-crosswords for which there exists an (R, C)-crossword solution of the given dimensions.

RC was shown to be **NP**-complete in [Fen14] via an indirect, complicated reduction. In this section, we give a much more straightforward polynomial reduction from **3SAT** to **RC**.

3.1 The reduction

Given a Boolean formula φ with $k \geq 1$ variables and d clauses as defined in Section 2.1 above (where we can assume $d \geq 3$), we construct an instance $\langle 0^{d+1}, 0^{k+d}, R, C\rangle$ of **RC** as follows: For $1 \leq i \leq d$, we define t_i to be the regex

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

Then we define

$$\begin{aligned} S &= \mathbf{1}^d\mathbf{0}^* \\ R &= \left(\bigcup_{i=1}^{d} t_i R_i\right) \cup S \\ C &= \mathbf{1}\,(\mathbf{0}^*\mathbf{1}\mathbf{0}^*) \cup \mathbf{0}(\mathbf{0}^* \cup \mathbf{1}^*) \end{aligned}$$

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

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

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

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

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

We show that φ is satisfiable iff an (R, C)-crossword solution exists.

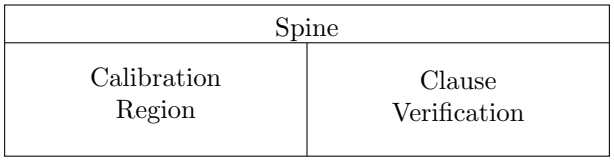
[3] More precisely, the question is whether the sentence $\exists x_0 \forall y_0 \cdots \exists x_{k-1} \forall y_{k-1} \exists x_k [\tilde{\varphi}(x_0, y_0, \ldots, x_{k-1}, y_{k-1}, x_k) = \text{TRUE}]$ holds in the two-element Boolean algebra $(\{\text{FALSE}, \text{TRUE}\}, \wedge, \vee, \neg)$.

First, assuming that φ is satisfiable, where $\langle z_1, \ldots, z_k \rangle$ is a satisfying assignment, then this sets up a $d+1$ by $d+k$ crossword solution of the following form:

	c_1	c_2	c_3	$\cdots$	c_d	c_{d+1}	$\cdots$	c_{d+k}
r_0	1	1	1	$\cdots$	1	0	$\cdots$	0
r_1	1	0	0	$\cdots$	0	z_1	$\cdots$	z_k
r_2	0	1	0	$\cdots$	0	z_1	$\cdots$	z_k
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	z_1	$\cdots$	z_k
r_c	0	0	0	$\cdots$	1	z_1	$\cdots$	z_k

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:



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 clear that if φ is satisfiable, then the (R, C)-crossword constructed above is solvable. In other words, there is a way to fill in the crossword such that all rows match the regular expression R, and all columns match the regular expression 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 rearranging of the (non-spine) rows gives another valid solution. Due to this fact it is guaranteed that for each i, some row matches $t_i R_i$, which is important for the converse below.

3.2 An (R, C)-crossword solution guarantees φ is satisfiable

To complete the proof, it must be shown that if the crossword is solvable, this implies that φ is satisfiable. We do this via a series of lemmas.

Here we assume an (R, C)-crossword solution exists with rows $\langle r_0, \ldots, r_d \rangle$ and columns $\langle c_1, \ldots, c_{d+k} \rangle$.

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.

Lemma 2. 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., $r_0 = 010 \cdots$:

	c_1	c_2	c_3	c_4	$\cdot$	c_d	$\cdot$	$\cdot$
r_0	0	1	0	0	$\cdot$	0		
$\vdots$								

From the definition of C, we see that c_i must match $\mathbf{1}(\mathbf{0}^*\mathbf{10}^*)$, 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 $c_i = c_2 = 10100\cdots0$:

	c_1	c_2	c_3	c_4	$\cdot$	c_d	$\cdot$	$\cdot$
r_0	0	1	0	0	$\cdot$	0		
r_1		0						
r_2		1						
r_3		0						
$\vdots$		$\vdots$						

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.

r_j **matches** t_iR_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.

This proves the lemma. □

By Lemma 2, the first d columns must match $\mathbf{1}(\mathbf{0}^*\mathbf{10}^*)$; we call such columns *calibration columns.*

Lemma 3. *No row other than r_0 matches S.*

Proof. Again assume this not the case. By the previous lemma, 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. □

Lemma 4. *For any i, $1 \leq i \leq d$, some row matches t_iR_i*

Proof. By Lemmas 2 & 3, we have that r_0 is the only row to match the spine S. Since $R = (\bigcup_{i=1}^{d} t_iR_i) \cup S$, it follows that each of the other rows matches t_iR_i for some i. For the purposes of contradiction, assume that there is some t_iR_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:

	c_1	$\cdot$	$c_{\ell-1}$	c_ℓ	$c_{\ell+1}$	$\cdot$	c_d	c_{d+1}	$\cdot$
r_0	1	$\cdot$	1	1	1	$\cdot$	1	$\cdot$	
$\vdots$				$\vdots$					
r_n	0	$\cdot$	0	1	0	$\cdot$	0	$\cdot$	
$\vdots$				$\vdots$					
r_m	0	$\cdot$	0	1	0	$\cdot$	0	$\cdot$	
$\vdots$									

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

Lemma 5. *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 Lemma 4, 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 $\langle z_1, \ldots, z_k \rangle$. □

4 (R,C)-crossword games

For two given regexes R and C over 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.[4]

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 (defined in the usual way, using the operators $\cup$, $\|$, $*$).

Note that the numbers m and n are given in *unary*.

Definition 6. The language **RCG** is the set of all (R, C)-games where Rose has a winning strategy.

4.1 RCG ∈ PSPACE

It is straightforward to observe that **RCG** ∈ **PSPACE**. This follows from the properties of (R, C)-games: Given an instance of **RCG** of size $N = m \cdot n$,

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

4.2 Hardness of RCG

Here is the main result of our paper.

Theorem 7. TQBF $\leq_p$ RCG.

[4] 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 row or column matches the corresponding regular expression.

To prove Theorem 7, our main result, 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. 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′**.

We show our main result in two steps: in Lemma 8 we show how to polynomially reduce **TQBF** to **RCG′**; then we give a polynomial reduction from **RCG′** to **RCG** (Theorem 9 below). In using **RCG′**, the goal is to first consider a 'simpler' game to verify that there is a correspondence between the formulæ in **TQBF** and the possible games in **RCG**.

Lemma 8. $\mathbf{TQBF} \leq_p \mathbf{RCG'}$.

Proof. Given an instance φ of **TQBF** as in Equation (1) of Section 2.2 with c clauses and $2k+1$ variables, we construct an equivalent instance of **RCG′** with $m \coloneqq k+c+1$ rows and $n \coloneqq k+c$ columns. The intersection of the first $k+1$ rows and first $k+1$ columns we will call the *variable region*. There are c rows below this region, one for each clause of $\tilde{\varphi}$, which we collectively call the *clause region*. The regular expressions for each player in **RCG′** are over the alphabet $\{0,1\}$ and are defined as follows (with an explanation afterward): for $1 \leq i \leq m$, we let

$$R_i \coloneqq \begin{cases} (\mathbf{0} \cup \mathbf{1})^* \mathbf{0}^{c-1} & \text{if } 1 \leq i \leq k+1, \\ (\mathbf{0} \cup \mathbf{1})^* \mathbf{1} (\mathbf{0} \cup \mathbf{1})^* \mathbf{0}^{c-1} & \text{if } k+2 \leq i \leq m, \end{cases}$$

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

$$C_i \coloneqq \begin{cases} \bigcup\limits_{a,b \in \{\mathbf{0},\mathbf{1}\}} (\mathbf{0} \cup \mathbf{1})^{i-1} ab (\mathbf{0} \cup \mathbf{1})^{k-i} \| S_{iab} & \text{if } 1 \leq i \leq k, \\ \bigcup\limits_{a \in \{\mathbf{0},\mathbf{1}\}} (\mathbf{0} \cup \mathbf{1})^{k} a \| T_a & \text{if } i = k+1, \\ \mathbf{0}^* & \text{if } k+2 \leq i \leq n, \end{cases}$$

where given $1 \leq i \leq k+1$, and $a, b \in \{\mathbf{0}, \mathbf{1}\}$, the regexes S_{iab} and T_a are defined as follows: First, for $1 \leq j \leq c$ let

$$u_j \coloneqq \begin{cases} \mathbf{0} & \text{if } x_{i-1} \text{ occurs negatively in the } j\text{th clause}, \\ \mathbf{1} & \text{if } x_{i-1} \text{ occurs positively in the } j\text{th clause}, \\ \bot & \text{if } x_{i-1} \text{ does not occur in the } j\text{th clause}. \end{cases} \qquad v_j \coloneqq \begin{cases} \mathbf{0} & \text{if } y_{i-1} \text{ occurs negatively in the } j\text{th clause}, \\ \mathbf{1} & \text{if } y_{i-1} \text{ occurs positively in the } j\text{th clause}, \\ \bot & \text{if } y_{i-1} \text{ does not occur in the } j\text{th clause}. \end{cases}$$

Now for $1 \leq j \leq c$, define

$$d_j \coloneqq \begin{cases} \mathbf{1} & \text{if } u_j = a \text{ or } v_j = b, \\ \mathbf{0} & \text{otherwise.} \end{cases} \qquad e_j \coloneqq \begin{cases} \mathbf{1} & \text{if } u_j = a, \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Finally, we let $S_{iab} \coloneqq d_1 \| \cdots \| d_n$ and $T_a \coloneqq e_1 \| \cdots \| e_n$.

Each of the first $k+1$ rows and columns corresponds to the truth value (0 or 1) of one or two particular variables in the original formula, as depicted in Figure 2. The remainder of the rows (c of them) correspond to the clauses of φ.

x_0	?	?	$\cdot$	?
y_0	x_1	?	$\ddots$	?
?	y_1	x_2	$\ddots$	?
$\vdots$	$\ddots$	$\ddots$	$\ddots$	$\vdots$
?	?	$\cdot$	y_{k-1}	x_k

Figure 2: *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. When Rose plays the ith row (for $1 \leq i \leq k+1$) she is able to choose the truth value of x_{i-1} by placing a 0 or 1 in the corresponding square in Figure 2 (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 ith column according to C_i, he can similarly choose the truth value of y_{i-1} by placing a 0 or 1 in the corresponding square. However, because of the S_{iab} component of C_i, Colin is forced to then place a 1 in each of the last c positions corresponding to a clause that is satisfied by the truth settings of these two variables. (The minor exception is the $(k+1)$st column, where there is only the variable x_k to consider.)

Also note that in order for Rose to complete the board, there must be a 1 in at least one of the first $k+1$ 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 $\tilde{\varphi}$, making the two games equivalent. Our construction is clearly polynomial time, which finishes the proof. □

4.3 Constraining the Players

Theorem 9. **RCG′** $\leq_p$ **RCG**.

The rest of this section is a proof of Theorem 9. To reduce from **RCG′** to **RCG** we need to provide a method to consolidate the families of regular expressions into one regex per player. Here, we present a generic construction that forces the players to play in order, which can be applied to any **RCG′** game — forcing each player to play their families of regexes in index order.

Given an arbitrary instance $G \coloneqq \langle\langle R_1, \ldots, R_m\rangle, \langle C_1, \ldots, C_n\rangle\rangle$ of **RCG′**, we construct an equivalent instance of **RCG**. Our construction requires the alphabet Σ 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 given **RCG′** grid 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 $R_i \coloneqq \mathbf{1}^*$ for $m < i \leq n$,

yielding an evidently equivalent $n \times n$ game. We can do something similar if $m > n$. The instance of **RCG** we construct from G will then be a $(2n+1) \times (2n+1)$ game $H \coloneqq \langle 0^{2n+1}, 0^{2n+1}, R, C\rangle$. We may also assume that $n \geq 2$.

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

$$R := \mathbf{210}^* \quad \cup \tag{2}$$

$$\underbrace{\bigcup_{i=0}^{n-2} \mathbf{0}^i\mathbf{1}^3\mathbf{0}^{n-i-2}}_{\text{I}} \| \underbrace{\mathbf{0}^i\mathbf{10}^{n-i-1}}_{\text{II}} \quad \cup \tag{3}$$

$$\underbrace{\mathbf{00}^{n-2}\mathbf{11}}_{\text{Ir}} \| \underbrace{\mathbf{0}^{n-1}\mathbf{1}}_{\text{II}} \quad \cup \tag{4}$$

$$\bigcup_{i=1}^{n} \underbrace{\mathbf{0}^i\mathbf{10}^{n-i}}_{\text{III}} \| R_i \tag{5}$$

(a) Rose's regular expression. Regex (2) is the 'spine', while regexes (3–4) define the 'calibration' region (I, II). Regex (5) continues calibration in region III while also including the row regexes from G.

$$C := \mathbf{210}^* \quad \cup \tag{6}$$

$$\underbrace{\bigcup_{i=0}^{n-2} \mathbf{0}^i\mathbf{1}^3\mathbf{0}^{n-i-2}}_{\text{I}} \| \underbrace{\mathbf{0}^i\mathbf{10}^{n-i-1}}_{\text{III}} \quad \cup \tag{7}$$

$$\underbrace{\mathbf{00}^{n-2}\mathbf{11}}_{\text{Ic}} \| \underbrace{\mathbf{0}^{n-1}\mathbf{1}}_{\text{III}} \quad \cup \tag{8}$$

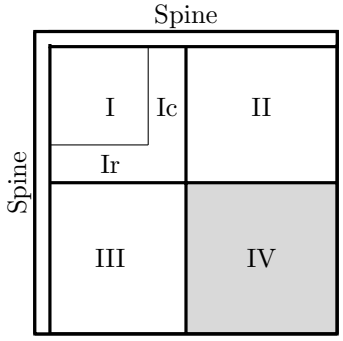
$$\underbrace{\bigcup_{i=1}^{n} \mathbf{0}^i\mathbf{10}^{n-i}}_{\text{II}} \| C_i \quad \cup \tag{9}$$

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

(b) Colin's regular expression. Regex (6) is the 'spine', regexes (7–8) are the calibration region (I and III), regex (9) continues calibration in region II while also including the column regexes from G, and regex (10) is a 'bomb' to punish Rose for cheating.

Figure 3: *The regular expressions wrapping games in* **RCG**. *Regexes are bracketed with the regions they describe, illustrated in Figure 4a.*

Figure 4a illustrates how H 'wraps' around the game G: players first fill in the spine, then regions I, II, and III before simulating the game G in the lower right square (light grey).



(a) Regions of the board

(b) An example of normal play

Figure 4: *Regions to constrain the players. Each 'block' is a $n \times n$ square.*

4.4 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 is in three stages:

Spine: In round 0, both players play the spine, i.e., a string matching $\mathbf{210}^*$.

Calibration: In round i, where $1 \leq i \leq n$, Rose and Colin each play a 'calibration string,' i.e. either the string matching $\mathbf{0}^i\mathbf{1}^3\mathbf{0}^{n-i-2}\|\mathbf{0}^i\mathbf{10}^{n-i-1}$ (if $i < n$) or $\mathbf{00}^{n-2}\mathbf{11}\|\mathbf{0}^{n-1}\mathbf{1}$ (if $i = n$).

Simulation: Rose and Colin now simulate the given $\mathbf{RCG}'$ game: In round $(n + i)$, for $1 \leq i \leq n$, Rose plays a string matching $\mathbf{0}^i\mathbf{10}^{n-i-1}\|R_i$ (if she can), and Colin plays a string matching $\mathbf{0}^i\mathbf{10}^{n-i-1}\|C_i$ (if he can).

Figure 4b 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 playing a *bomb*, i.e., a string matching $(\mathbf{0} \cup \mathbf{1} \cup \mathbf{100} \cup \mathbf{00}^*\mathbf{10})\mathbf{2}^*$, once or twice.

Lemma 10. *In round* 0*, if Rose does not play the spine, then Colin can win; otherwise, Colin must also play the spine.*

Proof. If Rose does not play $\mathbf{210}^*$, she has two choices for the first character. If she chooses 0, say, then Colin has a quick kill by playing a bomb (see Figure 5), with similar results if she cheats with a 1.

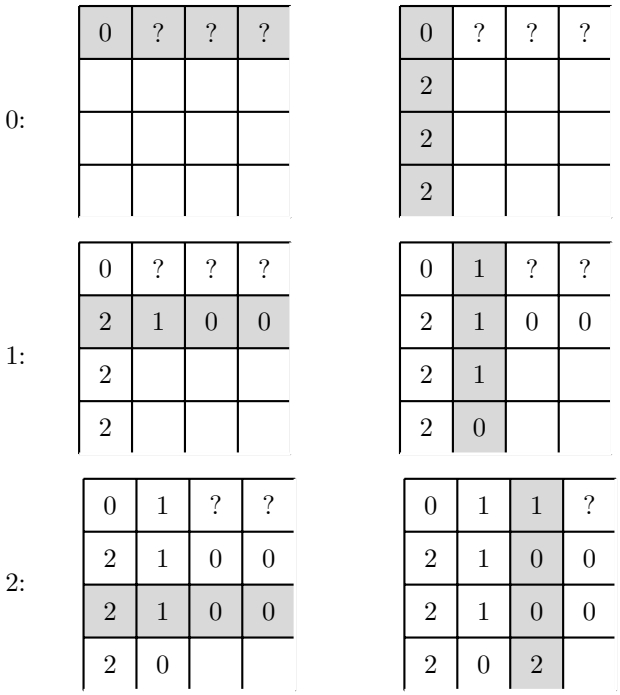


Figure 5: *Each round when Rose cheats with* $0\cdots$ *in her first move and Colin plays a bomb. Note that Rose has no regex to match the prefix* 20*. We replace a '?' with a* 1 *in round* 1 *to show the worst case, where Colin must survive through round* 2 *(not required in the* 0 *case).*

In either case, Rose would quickly lose. If Rose does play $\mathbf{210}^*$ on her first turn, Colin must play a string prefixed with 2, his only option matching the regex $\mathbf{210}^*$. □

Lemma 11. *After normal play through round* $(i-1)$ *for* $1 \le i \le n$*, Rose prefers regex (3) to regex (5) in round* i*.*

Proof. If $i = 1$, then Rose must play a string with prefix 1, and so she must play a string matching regex (3). Now suppose $i \ge 2$, and consider the following portion of the board at the start of round i when both players have been playing normally:

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

Rose has a choice of regexes (3) or (5), as each can match a string prefixed by $0^{i-1}1$. Say Rose chooses regex (5), thus playing a string matching $\mathbf{0}^{i-1}\mathbf{10}^{n-i}\|R_{i-1}$. Colin can then respond with a bomb:

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

(a) Rose plays regex (5)

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

(b) Colin plays regex (10), Rose loses

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

Lemma 12. *Colin cannot cheat in rounds* 1 *through* n*.*

Proof. By Lemma 10, we begin round 1 with the spine having been played by both players. Rose is then forced to play a string prefixed with 1, the only matching regex being regex (3) with $i = 0$: $\mathbf{1110}^{n-2}\|\mathbf{10}^{n-1}$. From this point on through round n, assuming Rose plays normally, Colin will be faced with prefix $0^{i-1}11$ in round i, and thus must play a string matching regex (7) or (8), i.e., play normally. □

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).

Lemma 13. *Assume normal play through round* n*. For* $1 \le i \le n$*, in round* $(n+i)$*, Rose must play a string matching* $\mathbf{0}^i\mathbf{10}^{n-i}\|R_i$ *and Colin must play a string matching* $\mathbf{0}^i\mathbf{10}^{n-i}\|C_i$*.*

Proof. In round $(n+i)$, Rose and Colin are both faced with prefix 0^i10^{n-i}, and the only regexes that this matches are the respective regexes given above for Rose and Colin. □

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 Theorem 9.

5 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.

Acknowledgments

We would like to thank Thomas Thierauf for several interesting discussions on this topic and to Joshua Cooper for finding for us a particularly challenging and fun regex crossword puzzle. We are also grateful to Klaus-Jörn Lange for suggesting the connection between our work and the theory of picture languages.

References

[AAa] http://regexcrossword.com.

[AAb] http://www.regexcrosswords.com.

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

[AAd] Royal dinner. http://regexcrossword.com/challenges/experienced/puzzles/1.

[AA13] February 2013. Slashdot discussion, http://games.slashdot.org/story/13/02/13/2346253/can-you-do-the-regular-expression-crossword.

[Ber66] Robert Berger. *The Undecidability of the Domino Problem.* Number 66 in Memoirs of the American Mathematical Society. American Mathematical Society, Providence, Rhode Island, 1966. MR0216954.

[Bla13] Lucy Black. 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.

[Fen14] S. Fenner. The complexity of some regex crossword problems, 2014.

[GR92] D. Giammarresi and A. Restivo. Recognizable picture languages. *International Journal of Pattern Recognition and Artificial Intelligence*, pages 31–46, 1992.

[GR97] D. Giammarresi and A. Restivo. Two-dimensional languages. In A. Salomaa and G. Rosenberg, editors, *Handbook of Formal Languages*, volume 3, chapter 96, pages 215–267. Springer-Verlag, 1997.

[LS97] M. Latteux and D. Simplot. Recognizable picture languages and domino tiling. *Theoretical Computer Science*, 178(1-2):275–283, 1997. Note.

[RR79] Azriel Rosenfeld and Werner Rheinboldt. *Picture Languages. Formal Models for Picture Recognition.* Computer science and applied mathematics. Elsevier Inc, Academic Press Inc, 1979.

[Tak14a] Glen Takahashi. Are regex crosswords NP-hard?, September 2014. CS Stack Exchange question 30143, answered by FrankW; http://cs.stackexchange.com/questions/30143/are-regex-crosswords-np-hard.

[Tak14b] Glen Takahashi. Are regex crosswords NP-hard? CS Stack Exchange question 30143, answered by FrankW, 2014.