Computational Complexity of SwitchDoor Games

Yunus Aydın
Oriel College
University of Oxford

supervised by
Dr. Standa Živný

A project report submitted in partial fulfilment of the requirements for the degree of

MCompSci

Trinity 2020

Abstract

We define and study the computational complexity of a family of 1-player games on graphs, that we call SwitchDoor. The goal in this game is to reach some specified room by moving between neighbouring rooms via open doors and using switches to toggle the state of some doors. We show this game is PSPACE-complete even when each switch is wired to at most 4 doors and each door is wired to at most 1 switch. We also present a polynomial-time algorithm for a restricted variant of this game called PathSD.

Contents

1 Introduction

We study the computational complexity of several decision problems regarding a 1-player game. In this game, you're in a maze of rooms, where each room has a switch wired to some doors.[1] When a switch is pressed, the doors that are wired to it are toggled: open doors become closed, closed ones become open. Initially, some doors are open, some are closed, and you are in one of the rooms. At each move, you either move to a neighbouring room via an open door or press the switch in your current room. The goal is to reach some specified room.

We'll be interested in the natural decision problem regarding this game: "Given a level of this game, is it possible to win?" This is what we call the SwitchDoor problem.[2] We'll also focus on some of the variants of this problem, where either the structure of the rooms or the wiring is restricted in some way.

1.1 Related Work

Viglietta [4] presented several "metatheorems" that relates the computational complexity of video games with the presence of some common game elements such as collectible items, doors, buttons, pressure plates, destroyable paths etc. Specifically, our work is related to the Metatheorem 5 of [4], which establishes lower bounds on the computational complexity of a game that involves doors and buttons where the goal is to reach some exit location. This is indeed quite similar to what we do, although there is a single significant difference between these two games: our switches are only able to toggle doors, whereas Viglietta's buttons can be wired to doors in three different ways so that when pressed, a single door might open/close/toggle three disjoint subsets of doors. In fact, they exploit the fact that buttons are able to open and close doors in their reductions to construct gadgets that correspond to one-way paths, which is not possible to do in SwitchDoor since every move is reversible. It's not clear to us whether there is a way

[1] One important thing to note is that switches are not necessarily wired to the doors linked to their rooms, and they're not restricted to those doors either. They might be wired to any subset of doors globally. It's also possible for switches to be not wired to any door at all.

[2] We use the word SwitchDoor to denote both the decision problem and the 1-player game. This is almost always inferable from context, otherwise we'll explicitly call it the SwitchDoor problem/game.

to adapt their reduction to make it work for SwitchDoor, although our intractibility results apply directly to their game involving buttons and doors.

There is another similar work [1] by Gabor and Williams that focus on a video game called Switches[3]. Very much like SwitchDoor, Switches is a game with the goal of reaching some target cell that involves only doors and switches. Like SwitchDoor, its switches can only toggle doors, but there are also some important differences between these games:

- In SwitchDoor, rooms and doors can form any sort of graph, whereas Switches is a game played on a $r \times c$ grid. The corresponding SwitchDoor variant would be PlanarSD.

- Similarly, the wiring is restricted in Switches: for any combination, two switches either have the same subset of doors or disjoint subsets. This is because the wiring is defined implicitly in Switches by a colour assignment of doors and switches. Each switch toggles the doors with the same colour as its own. The corresponding SwitchDoor variant is going to defined as ColourSD in Chapter 2.

- In Switches, whenever the player moves to a cell containing a switch, that switch is automatically operated, which is exactly like the pressure plates in [4]. In SwitchDoor on the other hand, the player always has the option not to press; they behave like the buttons in [4].

As you'll see in the following chapters, at times we're going to mix and match some restrictions on SwitchDoor to create new variants. The variant ColourPlanarSD is almost the same game as Switches, except the difference in their corresponding switch mechanics. Note that -in general- buttons can be *simulated* by pressure plates [4]. Because of this, ColourPlanarSD is computationally easier than the Switches decision problem. We're going to use this fact to apply some of our intractibility results to the Switches game.

Although lacking in some detail, Gabor and Williams [1] demonstrate a proof sketch for the PSPACE-hardness of the Switches decision problem by a reduction from TQBF. As one sees after some investigation, their reduction doesn't actually depend on the fact their switches act like pressure plates; thus, almost the same reduction can be used to show ColourPlanarSD is PSPACE-hard, which then implies SwitchDoor is PSPACE-hard. They also present a polynomial algorithm for $r = 1$ case.

[3] You can play the game at here.

1.2 Contributions

In this work, we present an alternative proof for PSPACE-hardness of SwitchDoor by a reduction from a decision problem related to Nondeterministic Constraint Logic, which was introduced by Hearn and Demaine as a general framework for proving PSPACE-hardness [2].

The reduction in [1] uses arbitrarily many doors and arbitrarily many switches per colour. By applying our results to the Switches game, we show that the Switches decision problem is PSPACE-complete even when the maximum number of doors per colour is restricted to 4, and the maximum number of switches per colour is restricted to 1, which settles two open problems that were proposed in [1], regarding the computational complexity of the Switches variants with constant bounds on the number of doors/switches per colour.

We also define another variant called PathSD, which is similar to the $r = 1$ case of the Switches game. Although it's similar to the $r = 1$ case, there is no straight-forward way to reduce to it. This is because the Switches game has the additional restriction on the wiring and also because to *simulate* the button-like switches of SwitchDoor with the pressure-plate-like switches of the Switches game, we need two rows, but we only have a single row in the $r = 1$ case. Thus, it's unlikely that their polynomial-time algorithm can be adapted to work for PathSD. By reducing it to a system of linear equations over $GF(2)$, we prove that it's in P.

2 Definitions

In this section, we provide the formal definition of SwitchDoor and introduce some other definitions, notation related to this problem.

2.1 SwitchDoor

Rooms and doors between them will be represented by an undirected graph $G = (V, E)$, where V is the set of rooms and E is the set of doors. The set of initially open doors will be called $E_0 \subseteq E$. We'll also have a total function $w : V \to \mathcal{P}(E)$ to represent where each switch is wired to. $w(u)$ is allowed to be any subset of E as previously stated in Chapter 1. We'll call the pair (G, w) as a *maze*.

Definition 2.1. A SwitchDoor maze is a pair (G, w) consisting of a graph $G = (V, E)$ and a wiring function $w : V \to \mathcal{P}(E)$.

Since a state of this game depends only on the current room the player is in and the set of open doors at that moment, the set of possible states is $V \times \mathcal{P}(E)$. Furthermore, a valid move in this game will be defined as a relation on the set of possible states, namely $\to_{\mathcal{M}}$, which itself is a union of two other relations: $\rightsquigarrow_{\mathcal{M}}$ and $\dashrightarrow_{\mathcal{M}}$. These represent pressing a switch and moving to a neighbouring room, respectively.

Definition 2.2. Given a SwitchDoor maze $\mathcal{M} = ((V, E), w)$,

Let $S = V \times \mathcal{P}(E)$. Then

$$\begin{aligned}
\rightsquigarrow_{\mathcal{M}} &= \{((u, E_i), (v, E_{i+1})) \in S \times S \mid \quad u = v \quad \text{and} \quad E_i \oplus w(u) = E_{i+1}\} \\
\dashrightarrow_{\mathcal{M}} &= \{((u, E_i), (v, E_{i+1})) \in S \times S \mid \quad \{u, v\} \in E_i \quad \text{and} \quad E_i = E_{i+1}\} \\
\to_{\mathcal{M}} &= \rightsquigarrow_{\mathcal{M}} \cup \dashrightarrow_{\mathcal{M}}
\end{aligned}$$

$\to_{\mathcal{M}}$ corresponds to a single move in the game. To express zero or more consecutive moves, we'll get the transitive, reflexive closure of this relation.

Definition 2.3. For any relation R on S, we define *the transitive, reflexive closure* of R as the smallest reflexive and transitive relation on S that contains R. We denote this set as R^*.

Using this notion of a move, we get the definition of the decision problem modelling whether it's possible to win given a level of this game.

Problem 1. SwitchDoor
Instance: A 3-tuple $(\mathcal{M}, q_0, g)$ consisting of

- A SwitchDoor maze $\mathcal{M} = ((V, E), w)$,
- An initial state $q_0 = (s, E_0)$ such that $s \in V$ and $E_0 \subseteq E$,
- A goal room $g \in V$.

Question: Is there any $E' \subseteq E$ such that $q_0 = (s, E_0) \to_{\mathcal{M}}^* (g, E')$?

2.2 Restrictions on the Wiring

One of the interesting things we can ask about SwitchDoor is "How does the complexity of the game depend on the structure of the wiring?" There are many ways to restrict the wiring, one obvious variant is where some of

- the number of doors a switch is wired to
- the number of switches a door is wired to

are bounded by some constants. We define the following problem to address this.

Problem 2. (n, m)-SwitchDoor
Instance: $(\mathcal{M} = ((V, E), w), q_0, g)$ such that

$$\forall v \in V\colon |w(v)| \leqslant n,$$
$$\forall e \in E\colon |\{v \in V \mid e \in w(v)\}| \leqslant m.$$

Question: Is there any $E' \subseteq E$ such that $q_0 \to_{\mathcal{M}}^* (g, E')$?

Note that these problems don't restrict the degree of vertices in V. That would be a restriction on the graph, not the wiring.

We'll define another variant with a wiring restriction called ColourSD, as mentioned in the previous chapter.

Problem 3. ColourSD
Instance: $(\mathcal{M} = ((V, E), w), q_0, g)$ such that

$$\forall v_1, v_2 \in V\colon w(v_1) \cap w(v_2) \neq \emptyset \implies w(v_1) = w(v_2)$$

Question: Is there any $E' \subseteq E$ such that $q_0 \to^*_{\mathcal{M}} (g, E')$?

Notation. In some cases, it will be useful to have one of the restrictions, but not the other. In those cases, we'll put $\cdot$ in the place of the corresponding parameter. For instance, if we wanted to discuss a variant of SwitchDoor where each switch is wired to at most two doors, while there is no restriction on doors, we would refer to it by $(2,\cdot)$-SwitchDoor.

2.3 Simpler Graphs

Another way to restrict the problem is by restricting the class of graphs allowed. More specifically, we'll mainly focus on the cases where the given graph is a path graph or a planar graph.

Problem 4. PathSD
Instance: $(\mathcal{M} = ((V, E), w), E_0)$ such that

$$V = \{u_0, u_1, \ldots, u_{n-1}\},$$
$$E = \{\{u_i, u_{i+1}\} \mid i \in \{0, 1, .., n-2\}\}.$$

Question: Is there any $E' \subseteq E$ such that $(u_0, E_0) \to^*_{\mathcal{M}} (u_{n-1}, E')$?

Problem 5. PlanarSD
Instance: $(\mathcal{M} = ((V, E), w), q_0, g)$ such that (V, E) is a planar graph.
Question: Is there any $E' \subseteq E$ such that $q_0 \to^*_{\mathcal{M}} (g, E')$?

3 SwitchDoor

Let us first give an upper bound for the complexity of SwitchDoor. Is it in NP for instance? At first sight, it might seem like given a solution, a sequence of valid moves, to an instance of this game, it's easy to verify this proposed solution. The problem with this idea is it doesn't have to be the case that there are always some polynomial number of moves that wins the game if the instance is "winnable". Since the size of the state space has an exponential size in terms of the size of the instance, the player might need to do exponential number of moves to win the game in some cases. But using the fact NPSPACE = PSPACE, we can show it's in PSPACE.

Theorem 3.1. SwitchDoor *is in* PSPACE.

Proof. Let us first give an algorithm that decides SwitchDoor, to show SwitchDoor is in NPSPACE.

Algorithm 1 A Nondeterministic Poly-Space Decision Procedure for SwitchDoor

```
1:  procedure SwitchDoor(M, q_0, g)
2:      ((V, E), w) ← M
3:      (u_0, E_0) ← q_0

4:      for i ← 0 to |V × P(E)| − 1 do
5:          if u_i = g then
6:              return true
7:          end if

8:          M ← {q | (u_i, E_i) →_M q}                ▷ valid moves from (u_i, E_i)
9:          (u_{i+1}, E_{i+1}) ← pick nondeterministically from M
10:     end for
11:     return false
12: end procedure
```

We need to show that Algorithm 1 decides SwitchDoor, which is to say it's sound, complete and always halts. We also need to show it has a polynomial space complexity.

Termination and soundness are straightforward. It always halts after at most $|V \times \mathcal{P}(E)|$ iteration of the loop. At each iteration it picks a valid move, so there must be a valid

path from q_0 to (g, E') for some E' if it returns **true**. It's also complete because there is no need to look for paths with cycles. If there was a valid path with some cycles, we could just remove them to get another valid path without any cycles. Since any path of length more than $|V \times \mathcal{P}(E)|$ has to contain some cycle (simply by pigeonhole principle), we don't need to look for those.

$|V \times \mathcal{P}(E)|$ is an exponential number in terms of the size of the instance, so i can be stored in polynomial space. (u_i, E_i) takes polynomial space as well. Note that we don't actually store each (u_i, E_i) seperately. We can always overwrite (u_{i+1}, E_{i+1}) in place of (u_i, E_i) because we won't need the previous variables. In other words, here indexing is just used to describe the change over time, not to describe an array. Since there are at most $|E| + 1$ valid moves at each step, M also takes polynomial space. All in all, the algorithm has a polynomial space complexity.

Algorithm 1 is a nondeterministic poly-space algorithm that decides SwitchDoor, so SwitchDoor must be in NPSPACE.

By Savitch's theorem [3], we know NPSPACE = PSPACE; thus, SwitchDoor is also in PSPACE. □

3.1 (1,·)-SwitchDoor

This is the variant where each switch is only wired to at most one door. The key idea here is that we never need to toggle a door from the open state to the closed one. We only need to press the switches that open a door.

Theorem 3.2. $(1, \cdot)$-SwitchDoor *is in* P.

Proof. As stated above, we're only going to search for rooms that are reachable with a sequence of moves that never closes a door. If there was a valid sequence of moves that contains moves that close a door, we could get rid of them while maintaining the sequence's validity. To do so, we first pick the first move that closes a door. Either that door is never opened again in the sequence -thus never used-, in which case we can remove that move from the sequence without effecting the following moves. Or that door is opened again at some point, in which case we remove both moves from the sequence. This way, neither the moves in between or the moves after the second one gets affected. We can keep on this process to eliminate all moves that close a door.

To explore the rooms that are reachable with a sequence that never closes a door, we can use a standard graph search algorithm with a slight modification.

Algorithm 2 Poly-Time Decision Procedure for $(1,\cdot)$-SwitchDoor

```
1:  procedure SwitchDoor(M, q_0, g)
2:      ((V, E), w) ← M
3:      (s, E') ← q_0

4:      R ← {}
5:      Q ← queue({s})

6:      while ¬empty?(Q) do
7:          u ← pop(Q)
8:          if u ∈ R then
9:              continue
10:         end if

11:         {{v_1, v_2}} ← w(u)
12:         if {v_1, v_2} ∉ E' then
13:             E' ← E' ∪ {{w_1, w_2}}
14:             if v_1 ∈ R then
15:                 push(Q, v_2)
16:             end if
17:             if v_2 ∈ R then
18:                 push(Q, v_1)
19:             end if
20:         end if

21:         for v adjacent to u in (V, E') do
22:             push(Q, v)
23:         end for
24:         R ← R ∪ {u}
25:     end while

26:     if g ∈ R then
27:         return true
28:     else
29:         return false
30:     end if
31: end procedure
```

The part from line 12 to 20 is what makes this different from a standard graph search algorithm like BFS/DFS. This is where we possibly add a new edge to the graph depending on whether the current switch is opening a door. Whenever a new edge is added to the graph, we check the incident vertices to see if they were already visited; if

so, we add the opposing vertex to the queue because v_1 and v_2 are connected after the switch press, but they were not connected when the vertex was visited.

By pressing the switches in the same order as this algorithm, we could open all the doors in the final state of E'. After that, we would be able to move to any room in R. Thus, the algorithm is sound.

For completeness, we want to prove that

$$(u_0, E_0) \to_{\mathcal{M}} (u_1, E_1) \to_{\mathcal{M}} \cdots \to_{\mathcal{M}} (u_n, E_n) \text{ for some } E_0 \subseteq E_1 \subseteq \cdots \subseteq E_n$$
$$\implies u_n \text{ is visited at some point in the search with the initial state } q_0 = (u_0, E_0).$$

We can do so by an induction on n.

Base case. When $n = 0$, RHS is true because u_0 is going to be pushed to the queue initially.

Induction step. Assuming the induction hypothesis is true for all $n \leqslant k$,

We want to show u_{k+1} gets visited if there is some path

$$(u_0, E_0) \to_{\mathcal{M}} (u_1, E_1) \to_{\mathcal{M}} \cdots \to_{\mathcal{M}} (u_k, E_k) \to_{\mathcal{M}} (u_{k+1}, E_{k+1})$$
$$\text{with } E_0 \subseteq E_1 \subseteq \cdots \subseteq E_k \subseteq E_{k+1}.$$

We know u_k is going to be visited by the induction hypothesis. If u_k and u_{k+1} are equal, we can directly apply the induction hypothesis. If not, we consider two cases: $\{u_k, u_{k+1}\} \in E_0$ and $\{u_k, u_{k+1}\} \notin E_0$.

If $\{u_k, u_{k+1}\} \in E_0$, the set of open doors E' in the algorithm initially contains $\{u_k, u_{k+1}\}$. Since E' only gets larger, $\{u_k, u_{k+1}\}$ must still be in E' when u_k is visited; therefore, u_{k+1} gets pushed to the queue and gets visited at some point.

If $\{u_k, u_{k+1}\} \notin E_0$, for some i smaller than k, the switch at u_i must open the door $\{u_k, u_{k+1}\}$. In other words, for some i smaller than k, $w(u_i)$ must equal to $\{\{u_k, u_{k+1}\}\}$. Otherwise, E_k couldn't contain $\{u_k, u_{k+1}\}$, and the move from u_k to u_{k+1} would be invalid. Since $i < k$, we can apply the induction hypothesis. We know the algorithm is going to visit u_i, so $\{u_k, u_{k+1}\}$ must be added to E' at some point: either by some other vertex that gets visited before u_i or by u_i. We also know u_k is going to be visited by the induction hypothesis. If u_k isn't visited yet when $\{u_k, u_{k+1}\}$ is being added, E' is going to contain $\{u_k, u_{k+1}\}$ when u_k gets visited, and u_{k+1} is going to be pushed to the queue as wanted. If u_k is already visited when $\{u_k, u_{k+1}\}$ is being added, u_{k+1} is going to be pushed to the queue -just after E' is updated- by the lines 14 - 19.

The time complexity analysis of Algorithm 2 is more or less the same as the analysis of DFS/BFS. New edges might be added to the graph during the search, but each edge is added at most once, and every added edge must be one of the edges from E, so it still takes linear time -assuming we have linear space-.

Algorithm 2 is a linear-time algorithm that decides $(1,\cdot)$-SwitchDoor. Because of this, $(1,\cdot)$-SwitchDoor must be in P. □

3.2 Nondeterministic Constraint Logic

The following reduction in this section are going to be from a decision problem related to Nondeterministic Constraint Logic, which was introduced by Hearn and Demaine as a general framework for proving PSPACE-hardness [2].

NCL "machine", as they call it, is specified by a *constraint graph*, which is a graph with a weight assignment to its vertices and edges.

Definition 3.3. A constraint graph $\mathcal{G}$ is a pair (G, f) consisting of an undirected simple graph $G = (V, E)$ and a weight assignment $f : V \cup E \to \mathbb{N}$. The weight of a vertex is called its *minimum inflow*.

In fact, we'll be focusing on a more specific variant called an *AND/OR constraint graph*.

Definition 3.4. An AND/OR constraint graph is a constraint graph $((V, E), f)$ where each vertex of V is either an AND vertex or an OR vertex:

- Both AND and OR vertices have a degree of 3 and a minimum inflow of 2.
- An AND vertex's incident edges have weights of 1, 1 and 2, whereas an OR vertex's incident edges all have a weight of 2.

A *configuration* C of the constraint graph $\mathcal{G}$ is an assignment of orientation (direction) to each edge in E. This describes the state of the machine. And a configuration is *legal* if and only if each vertex v in V satisfies its minimum inflow constraint, which is to say v's inflow, the sum of the weights of the edges pointing into v, is greater than or equal to v's minimum inflow.

Definition 3.5. Given a constraint graph $\mathcal{G} = ((V, E), f)$,

The set of all legal configurations of $\mathcal{G}$

$$S_{\mathcal{G}} = \{C \subseteq V \times V \mid |\{(u,v),(v,u)\} \cap C| = 1 \text{ for all } \{u,v\} \in E \text{ and } \sum_{(u,v)\in C} f(\{u,v\}) \geqslant f(v) \text{ for all } v \in V\}.$$

Initially, the NCL machine has a legal configuration. At each step, the machine can do an *edge reversal*, which reverses the orientation of an edge. An edge can be reversed only when the resulting configuration is still legal. The edge reversal is defined as a relation on $S_{\mathcal{G}}$.

Definition 3.6. Given a constraint graph $\mathcal{G} = ((V,E),f)$,

$$r(\{u,v\},C) = C \oplus \{(u,v),(v,u)\}$$
$$\rightarrow_{\mathcal{G}} = \{(C_i, C_{i+1}) \in S_{\mathcal{G}} \times S_{\mathcal{G}} \mid r(e,C_i) = C_{i+1} \text{ for some } e \in E\}$$

We're interested in the configuration-to-edge problem on AND/OR constraint graphs, that is: given an AND/OR constraint graph, its initial configuration and some target edge, is there a sequence of edge reversals that reverses that edge? [1]

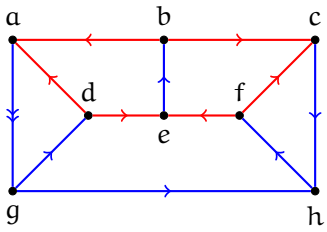

Figure 3.1: An example instance of NCL

Problem 6. NCL
Instance: A 3-tuple $(\mathcal{G}, C_0, t)$ consisting of

- An AND/OR constraint graph $\mathcal{G} = ((V,E),f)$,
- The initial configuration $C_0 \in S_{\mathcal{G}}$,
- The target edge $t \in E$.

Question: Is there any configuration $C \in S_{\mathcal{G}}$ such that $C_0 \rightarrow_{\mathcal{G}}^* C \rightarrow_{\mathcal{G}} r(t,C)$?

[1] For our convenience, we'll simply call it NCL instead of "configuration-to-edge problem on AND/OR constraint graphs".

Theorem 3.7 ([2])**.** NCL *is* PSPACE*-complete.*

Figure 3.1 above presents an example instance of NCL. Note that in the context of AND/OR constraint graphs, the colour blue represents a weight of 2, whereas red represents 1. The arrows given visulise the initial configuration, and the edge with the double arrow specifies the target edge. For this instance, the answer is yes because we are able to reverse the target edge {a,g} by first reversing {g,h} and then reversing {a,g}. That's also the shortest sequence because we're not able to reverse {a,g} directly. That wouldn't result with a legal configuration, since g wouldn't satisfy its minimum inflow constraint.

3.3 (4,1)-SwitchDoor

Here we show the PSPACE-hardness of (4,1)-SwitchDoor. To show it is PSPACE-hard, we're going to demonstrate a polynomial-time many-one reduction from NCL, which is PSPACE-complete, to (4,1)-SwitchDoor.

In our construction, we first create an *edge gadget* for each edge of the constraint graph in the NCL instance. Then, we *connect* these to get an instance of (4,1)-SwitchDoor.

Edge Gadget. Given an AND/OR constraint graph $\mathcal{G} = ((V, E), f)$, our motivation with edge gadgets is for them to *simulate* the edge reversals done in the constraint graph $\mathcal{G}$. More specifically, for each edge e in E of $\mathcal{G}$, we're going to create a gadget $R_{\mathcal{G}}(e) = (V_e, E_e, l_e)$ that contains a room g_e with a switch such that pressing it corresponds to reversing the edge $e$in the constraint graph. Thus, we want that switch to be reachable when and only when the reversal of e is a legal move in the constraint graph. To accomplish this, we're going to label each door in our SwitchDoor instance with an edge of the constraint graph combined with its orientation (with the l_e function of the $R_{\mathcal{G}}(e)$) and make the wiring according to these labels. The *why* and *how* of all these ideas are going to become more and more apparent as we progress along this section; now it's time to get our hands dirty and actually define the gadget.

Since we have an AND/OR constraint graph, there are only 4 possible cases for the weights of $e = \{u, v\}$ and its incident edges on the constraint graph. If e is red, then both u and v must be an AND-vertex. If e is blue, then all three combinations are possible. We're going to define $R_{\mathcal{G}}$ case by case:

1. *Blue-OR-AND*: This is the case where u is an OR-vertex, v is an AND-vertex, and e is blue. More formally, this is the case where the following holds:

   - $f(\{v, y\}) = f(\{v, z\}) = 1$

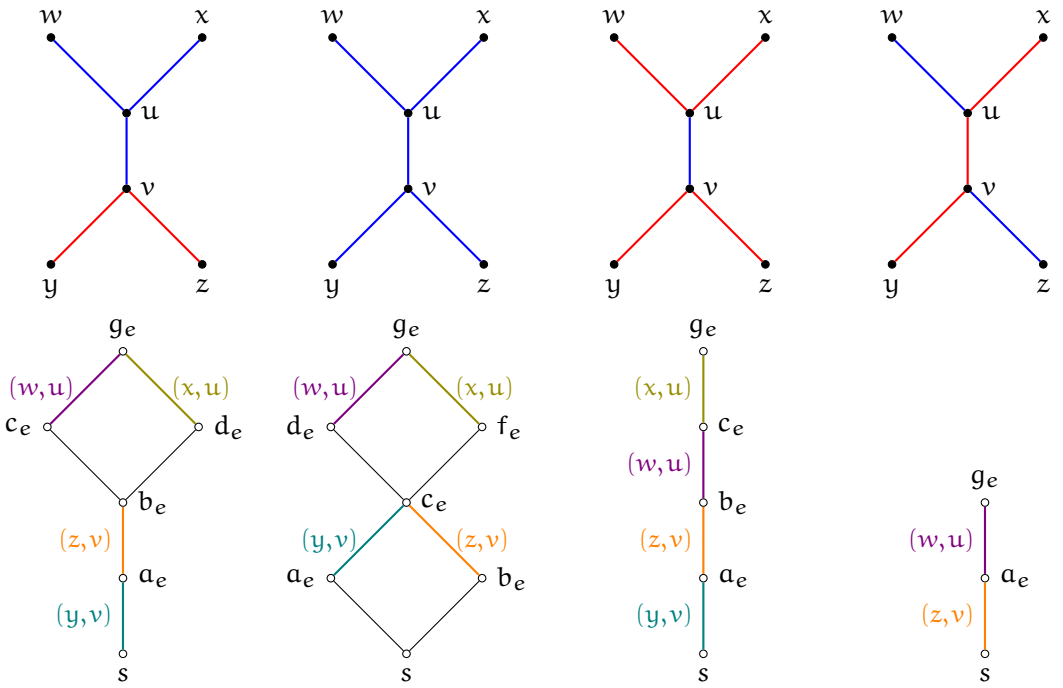

Figure 3.2: 4 possible cases used in $R_{\mathcal{G}}$'s definition and the corresponding gadgets[2]

- $f(\{w,u\}) = f(\{x,u\}) = f(\{u,v\}) = 2$
- $adj_G(u) = \{w,x,v\}$ and $adj_G(v) = \{u,y,z\}$[3]

In this case, $R_{\mathcal{G}}(e) = (V_e, E_e, l_e)$ where

- $V_e = \{s, g_e, a_e, b_e, c_e, d_e\}$
- $E_e = \{\{s,a_e\},\{s,b_e\},\{a_e,c_e\},\{b_e,c_e\},\{c_e,d_e\},\{d_e,g_e\}\}$
- And $l_e : E_e \to V \times V$ is defined as

$$l_e(x) = \begin{cases} (y,v) & \text{if } x = \{s,a_e\} \\ (z,v) & \text{if } x = \{a_e,b_e\} \\ (w,u) & \text{if } x = \{c_e,g_e\} \\ (x,u) & \text{if } x = \{d_e,g_e\} \end{cases}$$

As we're going to prove in (3.1) part of Lemma 3.13, the reversal of e is a legal

[3] $adj_G(u)$ here represents the adjacent vertices of u in the graph G.

move if and only if both u and v satisfy their minimum inflow constraints when the edge e is ignored (assuming we have a legal configuration to start with). For the *Blue-OR-AND* case, this implies the reversal of e is a legal move for the configuration C if and only if both (y,v) and (z,v) is in C and at least one of (w,u) and (x,u) is in C. We need both (y,v) and (z,v) for v to satisfy its minimum inflow constraint because their weights are 1 (and v's minimum inflow is 2), whereas one of (w,u) and (x,u) is enough since each has a weight of 2. Since we want g_e to be reachable from s if and only if the reversal of e is a legal move, we design the gadget so that the doors with labels (y,v) and (z,v) are *connected in series*, whereas the doors with labels (w,u) and (x,u) are *connected in parallel*. The edge gadget for other cases are constructed in a similar fashion.

2. *Blue-OR-OR*: This is the case where both u and v are OR-vertices, and e is blue. More formally, this is the case where the following holds:

   - $f(\{w,u\}) = f(\{x,u\}) = f(\{v,y\}) = f(\{v,z\}) = f(\{u,v\}) = 2$
   - $adj_G(u) = \{w,x,v\}$ and $adj_G(v) = \{u,y,z\}$

   In this case, $R_{\mathcal{G}}(e) = (V_e, E_e, l_e)$ where

   - $V_e = \{s, g_e, a_e, b_e, c_e, d_e, f_e\}$
   - $E_e = \{\{s,a_e\},\{s,b_e\},\{a_e,c_e\},\{b_e,c_e\},\{c_e,d_e\},\{c_e,f_e\},\{d_e,g_e\},\{f_e,g_e\}\}$
   - And $l_e : E_e \to V \times V$ is defined as

$$l_e(x) = \begin{cases} (y,v) & \text{if } x = \{a_e,c_e\} \\ (z,v) & \text{if } x = \{b_e,c_e\} \\ (w,u) & \text{if } x = \{d_e,g_e\} \\ (x,u) & \text{if } x = \{f_e,g_e\} \end{cases}$$

3. *Blue-AND-AND*: This is the case where both u and v are AND-vertices, and e is blue. More formally, this is the case where the following holds:

   - $f(\{w,u\}) = f(\{x,u\}) = f(\{v,y\}) = f(\{v,z\}) = 1$
   - $f(\{u,v\}) = 2$
   - $adj_G(u) = \{w,x,v\}$ and $adj_G(v) = \{u,y,z\}$

   In this case, $R_{\mathcal{G}}(e) = (V_e, E_e, l_e)$ where

- $V_e = \{s, g_e, a_e, b_e, c_e\}$
- $E_e = \{\{s, a_e\}, \{a_e, b_e\}, \{b_e, c_e\}, \{c_e, g_e\}\}$
- And $l_e : E_e \to V \times V$ is defined as

$$l_e(x) = \begin{cases} (y, v) & \text{if } x = \{s, a_e\} \\ (z, v) & \text{if } x = \{a_e, b_e\} \\ (w, u) & \text{if } x = \{b_e, c_e\} \\ (x, u) & \text{if } x = \{c_e, g_e\} \end{cases}$$

4. *Red-AND-AND*: This is the case where both u and v are AND-vertices, and e is red. More formaly, this is the case where the following holds:
   - $f(\{x, u\}) = f(\{u, v\}) = f(\{v, y\}) = 1$
   - $f(\{w, u\}) = f(\{v, z\}) = 2$
   - $adj_G(u) = \{w, x, v\}$ and $adj_G(v) = \{u, y, z\}$

   In this case, $R_G(e) = (V_e, E_e, l_e)$ where
   - $V_e = \{s, g_e, a_e\}$
   - $E_e = \{\{s, a_e\}, \{a_e, g_e\}\}$
   - And $l_e : E_e \to V \times V$ is defined as

$$l_e(x) = \begin{cases} (z, v) & \text{if } x = \{s, a_e\} \\ (w, u) & \text{if } x = \{a_e, g_e\} \end{cases}$$

Putting It All Together. Now that we know how to construct an edge gadget for any edge of the constraint graph of the NCL instance, we need to show how to combine those gadgets to complete the construction of a SwitchDoor maze. Fortunately, it's actually quite straightforward to do so. Most of the work is done by the edge gadget. Since (in NCL) it's allowed to reverse any edge as long as the resulting configuration is legal, the player in the constructed SwitchDoor maze should be able to move from any edge gadget to another. We actually hinted the solution in the gadget construction. All the gadgets are going to have one single shared vertex s so that the player can move from any gadget to another. So, to construct the vertices and edges of the SwitchDoor maze, it's enough to compute the vertices and edges of each gadget and get the union of those sets. For the wiring, remember that our goal was to simulate the reversal of e

(in the constraint graph) with the switch press at g_e (in the SwitchDoor maze). Since the reversal of e affects exactly (u,v) and (v,u) ($r(e,C) = C \oplus \{(u,v),(v,u)\}$), the switch at g_e should be wired to doors with labels (u,v) and (v,u).

Definition 3.8. Given an AND/OR constraint graph $\mathcal{G} = ((V,E),F)$,

$$l_{\mathcal{G}} = \bigsqcup_{e \in E} l_e$$

Definition 3.9. Given an AND/OR constraint graph $\mathcal{G} = ((V,E),f)$, the corresponding SwitchDoor maze $R(\mathcal{G})$ is the pair $((V',E'),w)$ where

$$V' = \bigcup_{e \in E} V_e, \quad E' = \bigsqcup_{e \in E} E_e,$$

$$w(x) = \begin{cases} \{y \in E' \mid l_{\mathcal{G}}(y) = (u,v) \text{ or } l_{\mathcal{G}}(y) = (v,u)\} & \text{if } x = g_{\{u,v\}} \text{ for some } \{u,v\} \in E \\ \emptyset & \text{otherwise} \end{cases}$$

Not surprisingly, the set of initially open doors is going to depend on the initial configuration of the constraint graph: for each directed edge in the initial configuration of NCL, we're going to open the doors that have that edge as a label.

Definition 3.10. Given a configuration $C \subseteq V \times V$ of some constraint graph $\mathcal{G} = ((V,E),f)$, we define the corresponding set of edges in $R(\mathcal{G}) = ((V',E'),w)$ as

$$h_{\mathcal{G}}(C) = \{e \in E' \mid l_{\mathcal{G}}(e) \in C\}.$$

All in all, given any instance of NCL $(\mathcal{G}, C_0, t)$, we can reduce the problem to (4,1)-SwitchDoor by computing $(R(\mathcal{G}), h_{\mathcal{G}}(C_0), s, g_t)$. An example construction is demonstrated in Figure 3.3 below. As you can see, we are able to reverse the target edge $\{a,g\}$ in the NCL instance by first reversing $\{h,g\}$ then reversing $\{a,g\}$. Similarly, we are able to get to the goal room $g_{\{a,g\}}$ in SwitchDoor instance by pressing the switch at room $g_{\{h,g\}}$ then moving to $g_{\{a,g\}}$.

Proof of Correctness. There are two things we need to prove. Firstly, it must be the case that $R(\mathcal{G})$ and $h_{\mathcal{G}}(C_0)$ is computable in polynomial time. And most importantly, reduction itself must be correct, namely

$$(\mathcal{G}, C_0, t) \in \text{NCL} \iff (R(\mathcal{G}), h_{\mathcal{G}}(C_0), s, g_t) \in (4,1)\text{-SwitchDoor}.$$

But let's not worry about the wiring constraints for now and only prove it for SwitchDoor first.

[4] For the NCL instance, the arrows represent the initial configuration, and the target edge is specified by the double arrows. For the SwitchDoor instance on the other hand, solid and dashed edges represent initially open and closed doors respectively, and the wiring is visualised by the colours assigned to switches and doors.

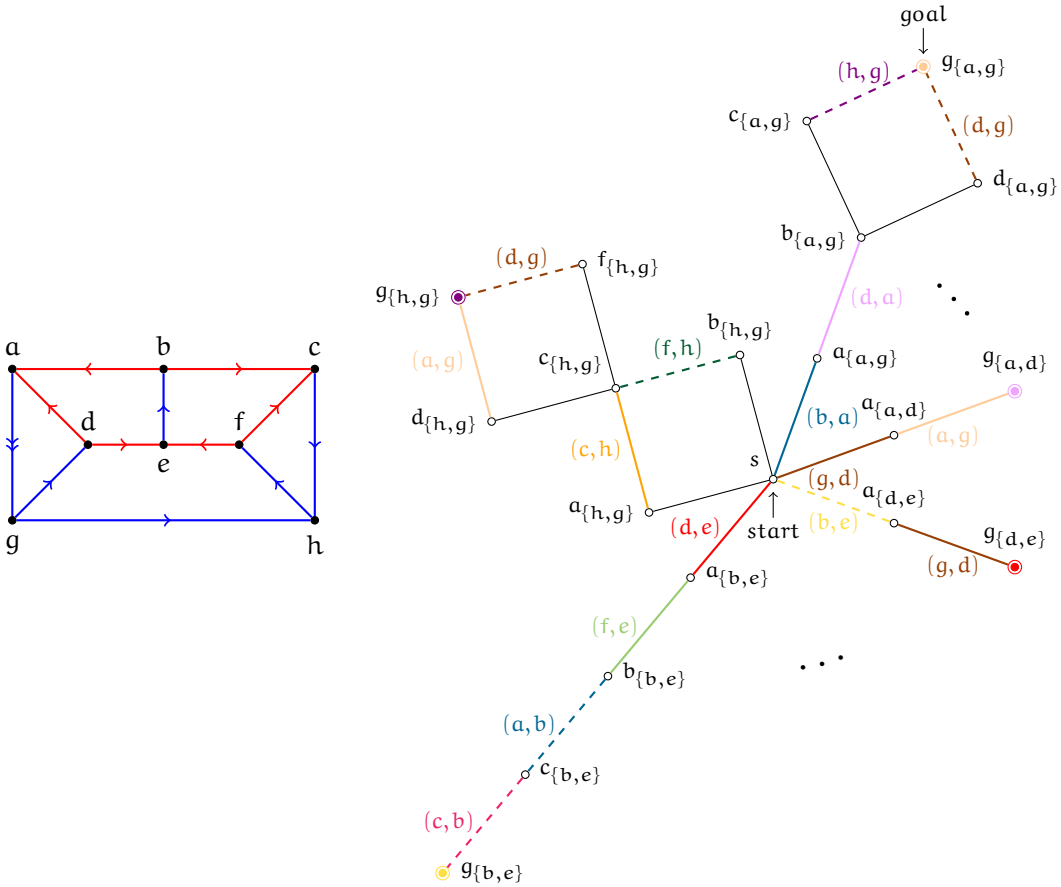

Figure 3.3: An example construction of the reduction from NCL to (4,1)-SwitchDoor[4]

Our proof has two main parts. We'll first define a specific sequence of moves in SwitchDoor as a new relation $\curvearrowright$ and show (in Lemma 3.13) a relation-preserving homomorhism between the relation representing a valid move in NCL and this new relation representing this specific sequence of moves in SwitchDoor. The second part of the proof, which consists of Lemma 3.14, 3.15 and 3.16, is going to demonstrate that this subset of moves is actually enough: if there is any solution to the given instance of SwitchDoor, there will be a solution consisting of $\curvearrowright$'s.

$\curvearrowright$ is going to be the relation that represents the sequence of moves that moves the player from s to some g_e, presses the switch at g_e and moves it back to s.

Definition 3.11. Given a SwitchDoor maze $\mathcal{M}$,

$$\curvearrowright_{\mathcal{M}} = \{((u,E),(v,E')) \in \rightarrow_{\mathcal{M}}^{*} \mid \quad u = v = s \quad \text{and} \quad (u,E) \dashrightarrow_{\mathcal{M}}^{*} (g_e,E) \rightsquigarrow_{\mathcal{M}} (g_e,E') \dashrightarrow_{\mathcal{M}}^{*} (v,E') \text{ for some } e\}$$

Lemma 3.12. *Given an AND/OR constraint graph* $\mathcal{G} = ((V,E),f)$, $C \subseteq V \times V$ *and* $\{u,v\} \in E$,

Let $R(\mathcal{G}) = ((V', E'), w)$. *Then*

$$h_{\mathcal{G}}(C) \oplus w(g_{\{u,v\}}) = h_{\mathcal{G}}(r(\{u,v\}, C))$$

Proof.

$$\begin{aligned}
& h_{\mathcal{G}}(C) \oplus w(g_{\{u,v\}}) \\
&= h_{\mathcal{G}}(C) \oplus \{e \in E' \mid l_{\mathcal{G}}(e) = (u,v) \text{ or } l_{\mathcal{G}}(e) = (v,u)\} \\
&= h_{\mathcal{G}}(C) \oplus \{e \in E' \mid l_{\mathcal{G}}(e) \in \{(u,v),(v,u)\}\} \\
&= \{e \in E' \mid l_{\mathcal{G}}(e) \in C\} \oplus \{e \in E' \mid l_{\mathcal{G}}(e) \in \{(u,v),(v,u)\}\} \\
&= \{e \in E' \mid l_{\mathcal{G}}(e) \in C \oplus \{(u,v),(v,u)\}\} \\
&= \{e \in E' \mid l_{\mathcal{G}}(e) \in r(\{u,v\}, C)\} \\
&= h_{\mathcal{G}}(r(\{u,v\}, C))
\end{aligned}$$

□

Lemma 3.13. *Given an AND/OR constraint graph* $\mathcal{G} = ((V, E), f)$,

Assuming $C \in S_{\mathcal{G}}$,

$$C \to_{\mathcal{G}} C' \iff (s, h_{\mathcal{G}}(C)) \curvearrowright_{R(\mathcal{G})} (s, h_{\mathcal{G}}(C')).$$

Proof. Assuming $C \to_{\mathcal{G}} C'$, we know $C' = r(e, C)$ for some $e \in E$ by Definition 3.6. Similarly, $C' = r(e, C)$ for some $e \in E$ if $(s, h_{\mathcal{G}}(C)) \curvearrowright_{R(\mathcal{G})} (s, h_{\mathcal{G}}(C'))$. This is by Lemma 3.12. Thus, we suppose $C' = r(e, C)$. If that was not the case, both LHS and RHS would be false as wanted.

Let $R(\mathcal{G}) = ((V', E'), w)$, $e = \{u, v\}$, $adj_G(u) = \{w, x, v\}$ and $adj_G(v) = \{u, y, z\}$. We're going to prove the lemma in three parts.

$$C \to_{\mathcal{G}} C' \iff \begin{cases} \sum_{(u',u) \in C \cap \{(w,u),(x,u)\}} f(\{u', u\}) \geqslant f(u) \\ \sum_{(u',v) \in C \cap \{(y,v),(z,v)\}} f(\{u', v\}) \geqslant f(v) \end{cases} \tag{3.1}$$

$$\iff (s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (g_e, h_{\mathcal{G}}(C)) \tag{3.2}$$

$$\iff (s, h_{\mathcal{G}}(C)) \curvearrowright_{R(\mathcal{G})} (s, h_{\mathcal{G}}(C')) \tag{3.3}$$

(3.3) is going to be provided by Lemma 3.14, therefore here we only prove (3.1) and (3.2).

(3.1). For $\implies$, we're going to use the fact both C and C' are in S_G to show u and v satisfy their minimum inflow constraint even if we ignored the edge e.

$$\begin{aligned} & C \to_G C' \\ \implies & C, C' \in S_G \\ \implies & \begin{cases} \sum_{(u',u)\in C} f(\{u',u\}) \geqslant f(u) \\ \sum_{(u',u)\in C'} f(\{u',u\}) \geqslant f(u) \end{cases} \end{aligned}$$

Because $adj_G(u) = \{w, x, v\}$,

$$\iff \begin{cases} \sum_{(u',u)\in C\cap\{(w,u),(x,u),(v,u)\}} f(\{u',u\}) \geqslant f(u) \\ \sum_{(u',u)\in C'\cap\{(w,u),(x,u),(v,u)\}} f(\{u',u\}) \geqslant f(u) \end{cases}$$

If $(v,u) \notin C$, $C \cap \{(w,u),(x,u),(v,u)\} = C \cap \{(w,u),(x,u)\}$ as wanted; otherwise, $(v,u) \notin C'$, hence $C' \cap \{(w,u),(x,u),(v,u)\} = C' \cap \{(w,u),(x,u)\} = C \cap \{(w,u),(x,u)\}$.

$$\implies \sum_{(u',u)\in C\cap\{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u)$$

Here we showed u would satisfy its minimum inflow constraint even without e. The same argument applies to v as well. That concludes the $\implies$ part of (3.1).

For the $\impliedby$ part of the proof, we're going to use the RHS and the premise that C is a legal configuration to reach the conclusion that C' is also a legal configuration. For C' to be a legal configuration, each vertex needs to satisfy its minimum inflow constraint, and each edge needs to have exactly one orientation.

For all vertices except u and v, their minimum inflow constraint are going to be satisfied by the fact that they were already satisfied in C and that the reversal of $\{u,v\}$ doesn't affect the inflow of those vertices. Similarly, the reversal of an edge maintains the property that each edge has exactly one orientation.

$$\begin{aligned} & C \in S_G \\ \iff & \begin{cases} |\{(a,b),(b,a)\} \cap C| = 1 & \text{for all } \{a,b\} \in E \\ \sum_{(u',a)\in C} f(\{u',a\}) \geqslant f(a) & \text{for all } a \in V \end{cases} \\ \implies & \begin{cases} |\{(a,b),(b,a)\} \cap C| = 1 & \text{for all } \{a,b\} \in E \\ \sum_{(u',a)\in C} f(\{u',a\}) \geqslant f(a) & \text{for all } a \in V \setminus \{u,v\} \end{cases} \end{aligned}$$

Since the reversal of $\{u,v\}$ doesn't affect the inflow of any vertex $a \in V \setminus \{u,v\}$ and maintains the property that each edge has exactly one orientation,

$$\begin{aligned}
&\iff \begin{cases} |\{(a,b),(b,a)\} \cap r(\{u,v\},C)| = 1 & \text{for all } \{a,b\} \in E \\ \sum\limits_{(u',a) \in r(\{u,v\},C)} f(\{u',a\}) \geqslant f(a) & \text{for all } a \in V \setminus \{u,v\} \end{cases} \\
&\iff \begin{cases} |\{(a,b),(b,a)\} \cap C'| = 1 & \text{for all } \{a,b\} \in E \\ \sum\limits_{(u',a) \in C'} f(\{u',a\}) \geqslant f(a) & \text{for all } a \in V \setminus \{u,v\} \end{cases} \qquad (3.4)
\end{aligned}$$

Using the RHS we can show u and v also satisfy their minimum inflow constraint.

$$\begin{cases} \sum\limits_{(u',u) \in C \cap \{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u) \\ \sum\limits_{(u',v) \in C \cap \{(y,v),(z,v)\}} f(\{u',v\}) \geqslant f(v) \end{cases}$$

Because $C \cap \{(w,u),(x,u)\} = r(\{u,v\},C) \cap \{(w,u),(x,u)\}$ and $C \cap \{(y,v),(z,v\} = r(\{u,v\},C) \cap \{(y,v),(z,v)\}$,

$$\begin{aligned}
&\iff \begin{cases} \sum\limits_{(u',u) \in C' \cap \{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u) \\ \sum\limits_{(u',v) \in C' \cap \{(y,v),(z,v)\}} f(\{u',v\}) \geqslant f(v) \end{cases} \\
&\implies \begin{cases} \sum\limits_{(u',u) \in C'} f(\{u',u\}) \geqslant f(u) \\ \sum\limits_{(u',v) \in C'} f(\{u',v\}) \geqslant f(v) \end{cases} \qquad (3.5)
\end{aligned}$$

By combining (3.4) and (3.5), we get the desired result.

$$\begin{aligned}
&\begin{cases} |\{(a,b),(b,a)\} \cap C'| = 1 & \text{for all } \{a,b\} \in E \\ \sum\limits_{(u',a) \in C'} f(\{u',a\}) \geqslant f(a) & \text{for all } a \in V \setminus \{u,v\} \\ \sum\limits_{(u',u) \in C'} f(\{u',u\}) \geqslant f(u) & \\ \sum\limits_{(u',v) \in C'} f(\{u',v\}) \geqslant f(v) & \end{cases} \\
&\iff \begin{cases} |\{(a,b),(b,a)\} \cap C'| = 1 & \text{for all } \{a,b\} \in E \\ \sum\limits_{(u',a) \in C'} f(\{u',a\}) \geqslant f(a) & \text{for all } a \in V \end{cases} \\
&\iff C' \in S_{\mathcal{G}}
\end{aligned}$$

Because $C \in S_{\mathcal{G}}$ and $C' = r(e, C)$,

$$\iff C \to_{\mathcal{G}} C'$$

(3.2). For this proof, we consider each possible type of e (as stated in Figure 3.2):

1. *Blue-OR-AND*:

$$\begin{cases} \sum_{(u',u) \in C \cap \{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u) = 2 \\ \sum_{(u',v) \in C \cap \{(y,v),(z,v)\}} f(\{u',v\}) \geqslant f(v) = 2 \end{cases}$$

   $f(\{w,u\}) = 1$ and $f(\{x,u\}) = 1$, so both (w,u) and (x,u) must be in C, otherwise u's inflow would be less than 2. This is also sufficient for u's inflow to be greater than or equal to 2. By a similar argument, we know at least one of (y,v) and (z,v) is in C if and only if v's inflow is greater than or equal to 2.

$$\begin{aligned} &\iff \begin{cases} (w,u) \in C \\ (x,u) \in C \\ (y,v) \in C \vee (z,v) \in C \end{cases} \\ &\iff \begin{cases} (a_e, c_e) \in h_{\mathcal{G}}(C) \vee (b_e, c_e) \in h_{\mathcal{G}}(C) \\ (c_e, d_e) \in h_{\mathcal{G}}(C) \\ (d_e, g_e) \in h_{\mathcal{G}}(C) \end{cases} \\ &\iff \begin{cases} (a_e, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (c_e, h_{\mathcal{G}}(C)) \vee (b_e, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (c_e, h_{\mathcal{G}}(C)) \\ (c_e, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (d_e, h_{\mathcal{G}}(C)) \\ (d_e, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (g_e, h_{\mathcal{G}}(C)) \end{cases} \\ &\iff (s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (g_e, h_{\mathcal{G}}(C)) \end{aligned}$$

2. *Blue-OR-OR*:

$$
\begin{aligned}
&\begin{cases} \sum\limits_{(u',u)\in C\cap\{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u) = 2 \\ \sum\limits_{(u',v)\in C\cap\{(y,v),(z,v)\}} f(\{u',v\}) \geqslant f(v) = 2 \end{cases} \\
\iff &\begin{cases} (y,v) \in C \vee (z,v) \in C \\ (w,u) \in C \vee (x,u) \in C \end{cases} \\
\iff &\begin{cases} (a_e, c_e) \in h_{\mathcal{G}}(C) \vee (b_e, c_e) \in h_{\mathcal{G}}(C) \\ (d_e, g_e) \in h_{\mathcal{G}}(C) \vee (f_e, g_e) \in h_{\mathcal{G}}(C) \end{cases} \\
\iff &\begin{cases} (a_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (c_e, h_{\mathcal{G}}(C)) \vee (b_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (c_e, h_{\mathcal{G}}(C)) \\ (d_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (g_e, h_{\mathcal{G}}(C)) \vee (f_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (g_e, h_{\mathcal{G}}(C)) \end{cases} \\
\iff &(s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{\mathcal{R}(\mathcal{G})} (g_e, h_{\mathcal{G}}(C))
\end{aligned}
$$

3. *Blue-AND-AND*:

$$
\begin{aligned}
&\begin{cases} \sum\limits_{(u',u)\in C\cap\{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u) = 2 \\ \sum\limits_{(u',v)\in C\cap\{(y,v),(z,v)\}} f(\{u',v\}) \geqslant f(v) = 2 \end{cases} \\
\iff &\begin{cases} (y,v) \in C \\ (z,v) \in C \\ (w,u) \in C \\ (x,u) \in C \end{cases} \\
\iff &\begin{cases} (s, a_e) \in h_{\mathcal{G}}(C) \\ (a_e, b_e) \in h_{\mathcal{G}}(C) \\ (b_e, c_e) \in h_{\mathcal{G}}(C) \\ (c_e, g_e) \in h_{\mathcal{G}}(C) \end{cases} \\
\iff &\begin{cases} (s, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (a_e, h_{\mathcal{G}}(C)) \\ (a_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (b_e, h_{\mathcal{G}}(C)) \\ (b_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (c_e, h_{\mathcal{G}}(C)) \\ (c_e, h_{\mathcal{G}}(C)) \dashrightarrow_{\mathcal{R}(\mathcal{G})} (g_e, h_{\mathcal{G}}(C)) \end{cases} \\
\iff &(s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{\mathcal{R}(\mathcal{G})} (g_e, h_{\mathcal{G}}(C))
\end{aligned}
$$

4. *Red-AND-AND*:

$$\begin{aligned}
&\begin{cases} \sum_{(u',u)\in C\cap\{(w,u),(x,u)\}} f(\{u',u\}) \geqslant f(u) = 2 \\ \sum_{(u',v)\in C\cap\{(y,v),(z,v)\}} f(\{u',v\}) \geqslant f(v) = 2 \end{cases} \\
\iff &\begin{cases} (z,v) \in C \\ (w,u) \in C \end{cases} \\
\iff &\begin{cases} (s,a_e) \in h_{\mathcal{G}}(C) \\ (a_e,g_e) \in h_{\mathcal{G}}(C) \end{cases} \\
\iff &\begin{cases} (s,h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (a_e,h_{\mathcal{G}}(C)) \\ (a_e,h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (g_e,h_{\mathcal{G}}(C)) \end{cases} \\
\iff &(s,h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (g_e,h_{\mathcal{G}}(C))
\end{aligned}$$

□

Lemma 3.14. *Given an AND/OR constraint graph $\mathcal{G}$,*

$$(s,h_{\mathcal{G}}(C)) \curvearrowright_{R(\mathcal{G})} (s,h_{\mathcal{G}}(r(e,C))) \iff (s,h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (g_e,h_{\mathcal{G}}(C))$$

Proof. Let $e = \{u,v\}$. We'll prove this by showing whenever we can go from s to g_e, we could also come back to s if we were to press the switch at g_e.

$$(s,h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (t_e,h_{\mathcal{G}}(C))$$

Since (for any SwitchDoor maze $\mathcal{M}$) $q \dashrightarrow_{\mathcal{M}} q' \iff q' \dashrightarrow_{\mathcal{M}} q$ by Definition 2.2,

$$\iff (g_e,h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (s,h_{\mathcal{G}}(C))$$

Because none of the edges in $R_{\mathcal{G}}(\{u,v\})$ have either the label (u,v) or (v,u),

$$\begin{aligned}
&\iff (g_e,h_{\mathcal{G}}(C) \oplus \{(u,v),(v,u)\}) \dashrightarrow^*_{R(\mathcal{G})} (s,h_{\mathcal{G}}(C) \oplus \{(u,v),(v,u)\}) \\
&\iff (g_e,h_{\mathcal{G}}(C) \oplus w(g_e)) \dashrightarrow^*_{R(\mathcal{G})} (s,h_{\mathcal{G}}(C) \oplus w(g_e)) \\
&\iff (g_e,h_{\mathcal{G}}(r(e,C))) \dashrightarrow^*_{R(\mathcal{G})} (s,h_{\mathcal{G}}(r(e,C))) \qquad \text{By Lemma 3.12}
\end{aligned}$$

Using this biconditional we achieve the desired proposition.

$$
\begin{aligned}
& (s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (t_e, h_{\mathcal{G}}(C)) \\
\iff & \begin{cases} (s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (t_e, h_{\mathcal{G}}(C)) \\ (g_e, h_{\mathcal{G}}(r(e, C))) \dashrightarrow^*_{R(\mathcal{G})} (s, h_{\mathcal{G}}(r(e, C))) \end{cases} \\
\iff & (s, h_{\mathcal{G}}(C)) \dashrightarrow^*_{R(\mathcal{G})} (t_e, h_{\mathcal{G}}(C)) \rightsquigarrow_{R(\mathcal{G})} (g_e, h_{\mathcal{G}}(r(e, C))) \dashrightarrow^*_{R(\mathcal{G})} (s, h_{\mathcal{G}}(r(e, C))) \\
\iff & (s, h_{\mathcal{G}}(C)) \curvearrowright_{R(\mathcal{G})} (s, h_{\mathcal{G}}(r(e, C)))
\end{aligned}
$$

□

Lemma 3.15. *Given an AND/OR constraint graph $\mathcal{G}$,*

$$(s, E) \curvearrowright^*_{R(\mathcal{G})} (s, E') \iff (s, E) \to^*_{R(\mathcal{G})} (s, E')$$

Proof. Since $\curvearrowright_{R(\mathcal{G})}$ is a subset of $\to^*_{R(\mathcal{G})}$, $\implies$ part of the claim is correct. For the converse, we're going to partition $\to^*_{R(\mathcal{G})}$ according to $\rightsquigarrow_{R(\mathcal{G})}$ moves, then show each part is equivalent to a $\curvearrowright_{R(\mathcal{G})}$.

Note that if $E = E'$, both sides are correct since those are transitive, **reflexive** closures. So we don't need to deal with that case; we can assume $E \neq E'$ (and there must be at least one $\rightsquigarrow_{R(\mathcal{G})}$ move since $E \neq E'$).

Let's prove it by induction on the minimum number of $\rightsquigarrow_{R(\mathcal{G})}$ moves required to reach the state (s, E') from the state (s, E).

Base case. When the minimum number of $\rightsquigarrow_{R(\mathcal{G})}$ moves is 1,

$$
\begin{aligned}
& (s, E) \to^*_{R(\mathcal{G})} (s, E') \\
\implies & (s, E) \dashrightarrow^*_{R(\mathcal{G})} (g_e, E) \rightsquigarrow_{R(\mathcal{G})} (g_e, E') \dashrightarrow^*_{R(\mathcal{G})} (s, E') && \text{for some } e \\
\iff & (s, E) \curvearrowright_{R(\mathcal{G})} (s, E') \\
\implies & (s, E) \curvearrowright^*_{R(\mathcal{G})} (s, E')
\end{aligned}
$$

Induction step. When the minimum number of $\rightsquigarrow_{R(\mathcal{G})}$ moves is $k+1$, and the induction hypothesis holds for k,

$$
\begin{aligned}
& (s, E) \to^*_{R(\mathcal{G})} (s, E') \\
\iff & (s, E) \dashrightarrow^*_{R(\mathcal{G})} (g_e, E) \rightsquigarrow_{R(\mathcal{G})} (g_e, E'') \dashrightarrow^*_{R(\mathcal{G})} (s, E'') \to^*_{R(\mathcal{G})} (s, E') && \text{for some } e, E'' \\
\iff & (s, E) \curvearrowright_{R(\mathcal{G})} (s, E'') \to^*_{R(\mathcal{G})} (s, E') && \text{for some } E'' \\
\implies & (s, E) \curvearrowright_{R(\mathcal{G})} (s, E'') \curvearrowright^*_{R(\mathcal{G})} (s, E') && \text{for some } E'' \quad \text{By IH} \\
\iff & (s, E) \curvearrowright^*_{R(\mathcal{G})} (s, E')
\end{aligned}
$$

□

Lemma 3.16. *Given an AND/OR constraint graph* $\mathcal{G} = ((V, E), f)$*,*

Assuming $C \in S_{\mathcal{G}}$*,*

$$(u, h_{\mathcal{G}}(C)) \to_{R(\mathcal{G})} (v, E') \implies \textit{there is some } C' \in S_{\mathcal{G}} \textit{ such that } h_{\mathcal{G}}(C') = E'.$$

Proof. Suppose $(u, h_{\mathcal{G}}(C)) \to_{R(\mathcal{G})} (v, E')$. By Definition 2.2, this implies one of two things:

$$(u, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})} (v, E') \tag{3.6}$$
$$(u, h_{\mathcal{G}}(C)) \rightsquigarrow_{R(\mathcal{G})} (v, E') \tag{3.7}$$

If (3.6) is true, then $E' = h_{\mathcal{G}}(C)$ as wanted.

If (3.7) is true, then $u = g_e$ and $E' = h_{\mathcal{G}}(C) \oplus w(g_e)$ for some e. By Lemma 3.12, we know $E' = h_{\mathcal{G}}(C) \oplus w(g_e) = h_{\mathcal{G}}(r(e, C))$. We also know $r(e, C) \in S_{\mathcal{G}}$ becase $C \in S_{\mathcal{G}}$; therefore, proposition holds again. □

Lemma 3.17.

$$(\mathcal{G}, C_0, t) \in \text{NCL} \iff (R(\mathcal{G}), h_{\mathcal{G}}(C_0), s, g_t) \in \textsc{SwitchDoor}$$

Proof.

$$\begin{aligned}
& (\mathcal{G}, C_0, t) \in \text{NCL} \\
\iff & C_0 \to_{\mathcal{G}}^* C \to_{\mathcal{G}} r(t, C) && \text{for some } C \\
\iff & (s, h_{\mathcal{G}}(C_0)) \curvearrowright_{R(\mathcal{G})}^* (s, h_{\mathcal{G}}(C)) \curvearrowright_{R(\mathcal{G})} (s, h_{\mathcal{G}}(r(t, C))) && \text{for some } C \quad \text{By Lemma 3.13} \\
\iff & (s, h_{\mathcal{G}}(C_0)) \curvearrowright_{R(\mathcal{G})}^* (s, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})}^* (g_e, h_{\mathcal{G}}(C)) && \text{for some } C \quad \text{By Lemma 3.14} \\
\iff & (s, h_{\mathcal{G}}(C_0)) \to_{R(\mathcal{G})}^* (s, h_{\mathcal{G}}(C)) \dashrightarrow_{R(\mathcal{G})}^* (g_t, h_{\mathcal{G}}(C)) && \text{for some } C \quad \text{By Lemma 3.15}
\end{aligned}$$

For any e, any path arriving room g_e with some set of open doors E' must pass s one last time with the same set of open doors E' just before arriving g_e and then never press a switch until arriving g_e. Thus,

$$\begin{aligned}
\iff & (s, h_{\mathcal{G}}(C_0)) \to_{R(\mathcal{G})}^* (g_t, h_{\mathcal{G}}(C)) && \text{for some } C \\
\iff & (s, h_{\mathcal{G}}(C_0)) \to_{R(\mathcal{G})}^* (g_t, E') && \text{for some } E' \quad \text{By Lemma 3.16} \\
\iff & (R(\mathcal{G}), h_{\mathcal{G}}(C_0), s, g_t) \in \textsc{SwitchDoor}
\end{aligned}$$

□

Lemma 3.18. *There are polynomial-time algorithms for computing* $R(\mathcal{G})$ *and* $h_{\mathcal{G}}(C_0)$*.*

Proof. Let $R(\mathcal{G}) = ((V', E'), w)$.

Since we're only doing a constant amount of work constructing the gadget for each edge, the time needed is linear in terms of the size of $\mathcal{G}$. We then compute the union of those, which would just be a list concatenation if we were to implement sets with a list data structure for instance. And to compute $h_{\mathcal{G}}(C_0)$, even when it's done naively we would only need to iterate over $E' \times C_0$. Since E' has polynomial size in terms of the input, that takes polynomial time. Thus, the time complexity overall is polynomial. □

Theorem 3.19. NCL $\leqslant_P$ SwitchDoor

Proof. By Lemma 3.17 and Lemma 3.18, $(\mathcal{G}, E_0, t) \mapsto (R(\mathcal{G}), h_{\mathcal{G}}(E_0), s, g_e)$ is a polynomial-time many-one reduction from NCL to SwitchDoor. □

Corollary 3.19.1. NCL $\leqslant_P$ (4,1)-SwitchDoor

Proof. Let $\mathcal{G} = ((V, E), f)$ and $R(\mathcal{G}) = ((V', E'), w)$.

Here we show the instances of SwitchDoor produced by this reduction have switches wired up to 4 doors and doors wired up to 1 switch, thereby showing they're also intances of (4,1)-SwitchDoor.

For any door e in E', w.l.o.g. suppose that it has the label (u, v). We know that there is only one switch that could be wired to e: $t_{\{u,v\}}$. This is because all of the switches not in the form $t_{\{x,y\}}$ are not wired to anything, and out of the ones in the form $t_{\{x,y\}}$ only $t_{\{u,v\}}$ could be wired to e since its index *matches* e's label (u, v). Now the remining thing to show is that each switch is wired up to 4 doors.

For any switch $t_{\{u,v\}}$ in V' (we don't care about other switches because they are not connected to any door), it's wired to the door e in E' if and only if $l_{\mathcal{G}}(e) = (u, v)$ or $l_{\mathcal{G}}(e) = (v, u)$ which can only happen when e is part of some gadget $R_{\mathcal{G}}(e')$ where e' and $\{u, v\}$ are incident edges in the constraint graph. And because AND/OR constraint graphs are 3-regular, each edge has exactly 4 incident edges. We also know that no label or its *reverse* occur more than once in total in a gadget. Thus, the number of occurrences of both (u, v) and (v, u) in total can't exceed 4. □

Corollary 3.19.2. NCL $\leqslant_P$ (4,1)-ColourPlanarSD

Proof. Since each door is wired to at most one switch in our construction, for different u and v, $w(u)$ and $w(v)$ are always disjoint. It's also straightforward to see that each gadget we construct is planar, and since we're only joining them at a single vertex, the whole graphs is also planar. □

Theorem 3.20. (4,1)-SwitchDoor *is* PSPACE-*complete.*

Proof. This follows from Theorem 3.7, Corollary 3.19.1 and Theorem 3.1. □

Theorem 3.21. (4,1)-ColourPlanarSD *is* PSPACE-*complete.*

Proof. This follows from Theorem 3.7, Corollary 3.19.2 and Theorem 3.1. □

4 PathSD

In many parts of this chapter we're mostly going to care about the rooms where we press a switch and the order of those rooms. So, having a succint(er) notation to express that is useful. $\rightarrow_{\mathcal{M}}^{A}$ represents the sequence of moves that presses the switches in the order given by sequence A. Before defining that, let us first define some functions and notation related to sequences.

Definition 4.1. Given sequences $A = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in \mathbb{N}^n$ and $B = \langle b_0, b_1, \ldots, b_{m-1} \rangle \in \mathbb{N}^m$,

- AB is the concatenation of A and B.

$$AB = \langle a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{m-1} \rangle$$

- $\mathrm{min}(A)$ is the minimum element of A.
- $\mathrm{max}(A)$ is the maximum element of A.
- For $k \in \mathbb{N}$, $A_{\leqslant k}$ is the subsequence of A, consisting exactly of A's elements that are less than or equal to k.

$$A_{\leqslant k} = \begin{cases} \langle\rangle & \text{if } A = \langle\rangle \\ \langle a_0 \rangle \langle a_1, \ldots, a_{n-1} \rangle_{\leqslant k} & \text{if } a_0 \leqslant k \\ \langle a_1, \ldots, a_{n-1} \rangle_{\leqslant k} & \text{otherwise} \end{cases}$$

- $w(A)$ is the set of edges representing the effect of pressing all the switches in A.

$$w(A) = \bigoplus_{0 \leqslant i < n} w(u_{a_i})$$

Definition 4.2. Given a PathSD maze $\mathcal{M} = ((V, E), w)$ and a sequence $A = \langle a_0, a_1, \ldots, a_{m-1} \rangle \in \mathbb{N}^m$,

Let $S = V \times \mathcal{P}(E)$. Then

$$\rightarrow_{\mathcal{M}}^{A} = \begin{cases} \dashrightarrow_{\mathcal{M}}^{*} & \text{if } A = \langle\rangle \\ \{((u_i, E'), (u_j, E'')) \in S \times S \mid (u_i, E') \dashrightarrow_{\mathcal{M}}^{*} (u_{a_0}, E') \rightsquigarrow_{\mathcal{M}} \\ \qquad (u_{a_0}, E' \oplus w(u_{a_0})) \rightarrow_{\mathcal{M}}^{\langle a_1, \ldots, a_{m-1} \rangle} (u_j, E'')\} & \text{otherwise} \end{cases}$$

Note that for PathSD, this relation is equivalent to the original definition of multiple moves. Formally,

Lemma 4.3. *Given a* PathSD *maze* $\mathcal{M}$,

$$q \to^*_{\mathcal{M}} q' \iff q \to^A_{\mathcal{M}} q' \text{ for some } A$$

Proof. For any $\mathcal{M}$ and A,$\to^A_{\mathcal{M}}$ is a subset of $\to^*_{\mathcal{M}}$. Thus, we only need to show the $\implies$ direction. To do so, we can do a straight-forward induction on the minimum number of $\rightsquigarrow_{\mathcal{M}}$ moves required to go from q to q'.

Base case. When it's possible to from q to q' without any $\rightsquigarrow_{\mathcal{M}}$ moves,

$$\begin{aligned} & q \to^*_{\mathcal{M}} q' \\ \implies & q \dashrightarrow^*_{\mathcal{M}} q' \\ \iff & q \to^{\langle\rangle}_{\mathcal{M}} q' \end{aligned}$$

Induction step. When it's possible go from q to q' with $k+1$ $\rightsquigarrow_{\mathcal{M}}$ moves, and the induction hypothesis holds for k $\rightsquigarrow_{\mathcal{M}}$'s,

$$\begin{aligned} & q \to^*_{\mathcal{M}} q' \\ \implies & q \dashrightarrow^*_{\mathcal{M}} (u_i, E') \rightsquigarrow_{\mathcal{M}} (u_i, E' \oplus w(E')) \to^*_{\mathcal{M}} q' && \text{for some } i, E' \\ \iff & q \dashrightarrow^*_{\mathcal{M}} (u_i, E') \rightsquigarrow_{\mathcal{M}} (u_i, E' \oplus w(u_i)) \to^{A'}_{\mathcal{M}} q' && \text{for some } i, E', A' && \text{By IH} \\ \iff & q \to^{\langle i\rangle A'}_{\mathcal{M}} q' && \text{for some } i, A' \end{aligned}$$

□

In fact, there is no need to have sequences that contain numbers larger than $|V|-2$.

Lemma 4.4. *Given a* PathSD *maze* $\mathcal{M} = ((V, E), w)$ *and a sequence* $A \in \mathbb{N}^m$,

$$\begin{aligned} & (u_0, E_0) \to^A_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A)) \\ & \qquad \implies (u_0, E_0) \to^B_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(B)) \text{ for some } B \in \{0, 1, \ldots, |V|-2\}^k \end{aligned}$$

Proof. If the sequence A doesn't contain any $n-1$, then the claimed proposition holds. Otherwise, we partition the sequence into three contiguous subsequences $B\langle n-1\rangle C = A$ such that $n-1 \notin B$. Since the player need to visit u_{n-1} after B, we know $(u_0, E_0) \to^B_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(B))$. In other words, if there was a path from u_0 to u_{n-1} that used the switch at u_{n-1}, there would also be another path from u_0 to u_{n-1} that stopped just after arriving at u_{n-1} for the first time, not pressing that switch at all. □

Before tackling the actual problem, let us do some wishful thinking and ask some simpler questions with the hope that it will give us an insight about the original problem PathSD.

- *Instead of trying to search for any path, what would happen if we looked ones for that never backtrack?*
- *Would that assumption about the elements in our search make the problem any easier?*
- *How would we even check such a path to see if it's a solution to our instance?*

Let's start with the latter question, since that seems the easiest. In this situation we're given an instance of PathSD $(\mathcal{M}, E_0)$ where $\mathcal{M} = ((V, E), w)$ with $|V| = n$, and we want to check whether a sequence of press switches $A = \langle a_0, a_1, \ldots, a_{m-1} \rangle \in \{0, 1, \ldots, |V| - 2\}^m$, that we know is strictly increasing, is a solution to this instance. In other words, we would like to know whether

$$(u_0, E_0) \to_{\mathcal{M}}^{A} (u_{n-1}, E_0 \oplus w(A)).$$

What would be a necessary condition for A to satisfy the previous statement? For instance, it would need to be the case that any door, $\{u_i, u_{i+1}\}$, becomes open after switches (out of the switches in A) on its left, $A_{\leqslant i}$, are pressed. In fact, this is a sufficient condition. Formally, the condition is the following.

$$\forall \{u_i, u_{i+1}\} \in E \colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i})$$

Let's prove this is both necessary and sufficient.

Lemma 4.5. *Assuming A is strictly increasing,*

$$\forall \{u_i, u_{i+1}\} \in E \colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i}) \iff (u_0, E_0) \to_{\mathcal{M}}^{A} (u_{n-1}, E_0 \oplus w(A))$$

Proof.

$$
\begin{aligned}
&\forall \{u_i, u_{i+1}\} \in E\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i}) \\
\iff &\begin{cases}
\forall i \in \{0, 1, \ldots, a_0 - 1\}\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i}) \\
\forall i \in \{a_0, a_0 + 1, \ldots, a_1 - 1\}\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i}) \\
\vdots \\
\forall i \in \{a_{m-1}, a_{m-1} + 1, \ldots, n - 2\}\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i})
\end{cases} \\
\iff &\begin{cases}
\forall i \in \{0, 1, \ldots, a_0 - 1\}\colon \{u_i, u_{i+1}\} \in E_0 \\
\forall i \in \{a_0, a_0 + 1, \ldots, a_1 - 1\}\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(u_{a_0}) \\
\vdots \\
\forall i \in \{a_{m-1}, a_{m-1} + 1, \ldots, n - 2\}\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(u_{a_0}) \oplus w(u_{a_1}) \oplus \cdots \oplus w(u_{a_{m-1}}))
\end{cases} \\
\iff &\begin{cases}
(u_0, E_0) \dashrightarrow^*_{\mathcal{M}} (u_{a_0}, E_0) \\
(u_{a_0}, E_0 \oplus w(u_{a_0})) \dashrightarrow^*_{\mathcal{M}} (u_{a_1}, E_0 \oplus w(u_{a_0})) \\
\vdots \\
(u_{a_{m-1}}, E_0 \oplus w(A)) \dashrightarrow^*_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A))
\end{cases} \\
\iff &(u_0, E_0) \to^{\mathcal{A}}_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A))
\end{aligned}
$$

□

In fact, for our convenience we're going to call this *permissibility*.

Definition 4.6. A is *permissible* if and only if

$$\forall \{u_i, u_{i+1}\} \in E\colon \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i}).$$

Please note that for A to be permissible, A doesn't have to be strictly increasing. It's just that given that A is strictly increasing, saying A is permissible is the same thing as saying it's a valid solution because of Lemma 4.5.

One nice thing about permissibility (for strictly increasing A) is that it can be formulated as a system of linear equations over GF(2), the finite field with two elements.

$$W\mathbf{a} = \mathbf{e}$$

The ith row here is going to describe the constraint $\{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i})$. Thus, we construct $W \in \mathbb{Z}_2^{n-1} \times \mathbb{Z}_2^{n-1}$ and $\mathbf{e} \in \mathbb{Z}_2^{n-1}$ as follows.

$$W_{i,j} = \begin{cases} 1 & \text{if } \{u_i, u_{i+1}\} \in w(u_j) \text{ and } j \leqslant i \\ 0 & \text{otherwise} \end{cases}$$

$$e_i = \begin{cases} 1 & \text{if } \{u_i, u_{i+1}\} \notin E_0 \\ 0 & \text{otherwise} \end{cases}$$

Let's prove the correctness of this construction by showing a solution of this equation, $\mathbf{a}$, is going to uniquely determine a strictly increasing and permissible A and vice versa.

Lemma 4.7.

$\exists \boldsymbol{a} \in \mathbb{Z}_2^{n-1} : W\boldsymbol{a} = \boldsymbol{e} \iff \exists m \in \mathbb{N}, A \in \{0,1,2,\ldots,n-2\}^m :$ *A is strictly increasing and permissible*

Proof. To uniquely determine a sequence from a solution of the equations and vice versa, we define a bijection f between $\mathbb{Z}_2^{n-1}$ and subsequences of $\{0,1,2,\ldots,n-2\}$:

$f(\mathbf{a})$ is the subsequence of $\{0,1,2,\ldots,n-2\}$ such that $i \in f(\mathbf{a})$ if and only if $a_i = 1$.

Now that we've decided what $\mathbf{a}$ represents as a sequence, the remaining thing to show is

$$W\mathbf{a} = \mathbf{e} \iff f(\mathbf{a}) \text{ is permissible.}$$

To do so, we're going to consider each edge separately and show that ith edge satisfies the corresponding condition of permissibility if and only if ith row of the system of equations is satisfied. This is enough because if every row is satisfied then the corresponding conditions hold for each edge (so $f(\mathbf{a})$ is permissible) and vice versa.

Let $f(\mathbf{a}) = \langle b_0, b_1, \ldots, b_{m-1} \rangle$ and $f(\mathbf{a})_{\leqslant i} = \langle b_0, b_1, \ldots, b_{k_i} \rangle$. So for every $\{u_i, u_i + 1\} \in E$,

$$\begin{aligned}
& \{u_i, u_{i+1}\} \in E_0 \oplus w(f(\mathbf{a})_{\leqslant i}) \\
\iff & \{u_i, u_{i+1}\} \in E_0 \oplus w(u_{b_0}) \oplus w(u_{b_1}) \oplus \cdots \oplus w(u_{b_{k_i}}) \\
\iff & (1 + e_i) + W_{i,b_0} + W_{i,b_1} + \cdots + W_{i,b_{k_i}} && \equiv 1 \pmod 2 \\
\iff & e_i + W_{i,b_0} + W_{i,b_1} + \cdots + W_{i,b_{k_i}} && \equiv 0 \pmod 2 \\
\iff & W_{i,b_0} + W_{i,b_1} + \cdots + W_{i,b_{k_i}} && \equiv e_i \pmod 2 \\
\iff & W_{i,b_0} + W_{i,b_1} + \cdots + W_{i,b_{m-1}} && \equiv e_i \pmod 2 \\
\iff & a_0 W_{i,0} + a_1 W_{i,1} + \cdots + a_{n-2} W_{i,n-2} && \equiv e_i \pmod 2
\end{aligned}$$

□

So far, we have an incomplete algorithm to decide PathSD. Given an instance of PathSD, we can construct the described linear equations in poly-time and then decide whether it has a solution by Gaussian-Jordan elimination in poly-time. But it's incomplete. By Lemma 4.7, 4.5, 4.4 and 4.3, we know that the answer is yes for PathSD if there is a solution for the linear equations, **but not vice versa**. At this point, a key observation about PathSD comes to our rescue: for *any* sequence of switch presses, A, that takes us from u_0 to u_{n-1}, A must be permissible.

Lemma 4.8.

$$(u_0, E_0) \to^{A}_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A)) \implies A \textit{ is permissible}$$

Proof. Given that $(u_0, E_0) \to^{A}_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A))$, we want to show A is permissible, which is to say for any $\{u_i, u_{i+1}\} \in E$, $\{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i})$.

For any $\{u_i, u_{i+1}\} \in E$, we're going to consider three cases depending on i's relative position to $min(A)$ and $max(A)$.

1. When $i < min(A)$,

   The door $\{u_i, u_{i+1}\}$ is positioned left to the room where we press a switch for the first time, u_{a_0}. This means, this door must be open initially, namely $\{u_i, u_{i+1}\} \in E_0$. It's still open after all the switches in A left to this door are pressed, since there is no such switch in A ($i < min(A)$), and it was initially open.

   $$\begin{aligned}
   & (u_0, E_0) \to^{A}_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A)) \\
   \iff & (u_0, E_0) \dashrightarrow^{*}_{\mathcal{M}} (u_{a_0}, E_0) \to^{A}_{\mathcal{M}} (u_j, E_0 \oplus w(A)) \\
   \implies & (u_0, E_0) \dashrightarrow^{*}_{\mathcal{M}} (u_i, E_0) \dashrightarrow_{\mathcal{M}} (u_{i+1}, E_0) \dashrightarrow^{*}_{\mathcal{M}} (u_{a_0}, E_0) \\
   \implies & \{u_i, u_{i+1}\} \in E_0 \\
   \iff & \{u_i, u_{i+1}\} \in E_0 \oplus \emptyset \\
   \iff & \{u_i, u_{i+1}\} \in E_0 \oplus w(\langle\rangle) \\
   \iff & \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant k})
   \end{aligned}$$

2. When $max(A) \leqslant i$,

   The door $\{u_i, u_{i+1}\}$ is positioned right to the room where we press a switch for the last time, $u_{a_{m-1}}$. This means, this door must be open after all the switches in A are pressed, namely $\{u_i, u_{i+1}\} \in E_0 \oplus w(A)$. Furthermore, even the right-most switch in A is on this door's left ($max(A) \leqslant i$), so the last statement is equivalent to saying the door must be open after all the switches in $A_{\leqslant k}$ are pressed.

   $$\begin{aligned}
   & (u_0, E_0) \to^{A}_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A)) \\
   \iff & u_0, E_0) \to^{A}_{\mathcal{M}} (u_{a_{m-1}}, E_0 \oplus w(A)) \dashrightarrow^{*}_{\mathcal{M}} (u_{n-1}, E_0 \oplus w(A)) \\
   \implies & (u_{a_{m-1}}, E_0 \oplus w(A)) \dashrightarrow^{*}_{\mathcal{M}} (u_i, E_0 \oplus w(A)) \dashrightarrow_{\mathcal{M}} (u_{i+1}, E_0 \oplus w(A)) \dashrightarrow^{*}_{\mathcal{M}} (u_j, E_0 \oplus w(A)) \\
   \implies & \{u_i, u_{i+1}\} \in E_0 \oplus w(A) \\
   \iff & \{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant k})
   \end{aligned}$$

3. When $min(A) \leqslant i < max(A)$,

We're going to partition the switches pressed, A, into contiguous subsequences, by their relative position to the door $\{u_i, u_{i+1}\}$: depending on they're on the left of this door or otherwise.

Let $A = A_0 A_1 A_2 \ldots A_{2k+1}$ such that

$$\forall j \in \{0, 2, \ldots, 2k\}, a \in A_j : a \leqslant i \text{ and}$$
$$\forall j \in \{1, 3, \ldots, 2k+1\}, a \in A_j : a > i.$$

Let's denote the first and the last element of A_j as f_j and l_j. Note that A_0 and A_{2k+1} might be empty, but none of the others are not.

As with previous two cases, our goal here is to show pressing all the switches in $A_{\leqslant i}$ opens the door $\{u_i, u_{i+1}\}$. But since $A_{\leqslant i}$ is just the concatenation of $A_0, A_2, \ldots, A_{2k}$, one way to approach this problem is to examine each A_{2j} separately (for $0 \leqslant j \leqslant k$) to see how they affect the door. Two key observations here are that A_0 opens the door and that none of the others affect the door.

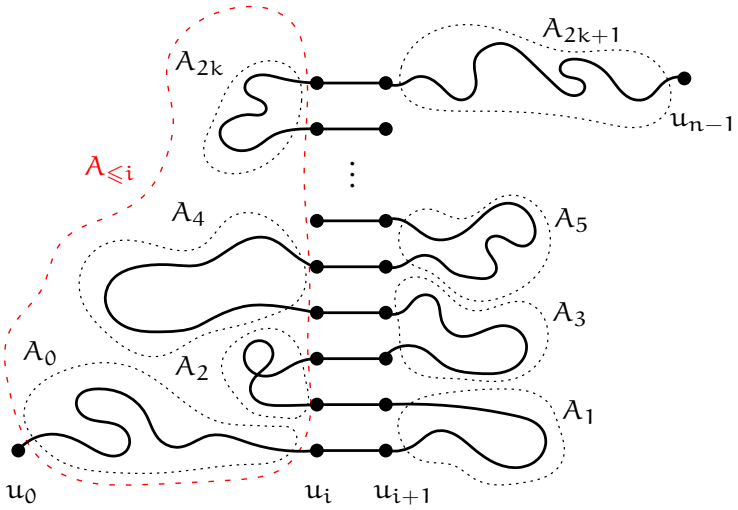

Figure 4.1: The partition of A, $A_0 A_1 A_2 \ldots A_{2k+1}$, where A is a sequence of switch presses representing a path from u_0 to u_{n-1}.

We first show, whether A_0 is empty or not, $\{u_i, u_{i+1}\} \in E_0 \oplus w(A_0)$, which is to say the door $\{u_i, u_{i+1}\}$ must be open when we first encounter it during our path from u_0 to u_{n-1}.

As stated before, either $A_0 = \langle\rangle$ or $A_0 \neq \langle\rangle$.

If $A_0 = \langle\rangle$,

$$
\begin{aligned}
& (u_0, E_0) \to_{\mathcal{M}}^{A} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0 A_1 \ldots A_{2k+1}} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0} (u_0, E_0) \dashrightarrow_{\mathcal{M}}^{*} (u_{f_1}, E_0) \to_{\mathcal{M}}^{A_1 A_2 \ldots A_{2k+1}} (u_{n-1}, E_0 \oplus w(A)) \\
\implies & (u_0, E_0) \dashrightarrow_{\mathcal{M}}^{*} (u_i, E_0) \dashrightarrow_{\mathcal{M}} (u_{i+1}, E_0) \dashrightarrow_{\mathcal{M}}^{*} (u_{f_1}, E_0) \\
\implies & \{u_i, u_{i+1}\} \in E_0 \\
\iff & \{u_i, u_{i+1}\} \in E_0 \oplus \emptyset \\
\iff & \{u_i, u_{i+1}\} \in E_0 \oplus w(\langle\rangle) \\
\iff & \{u_i, u_{i+1}\} \in E_0 \oplus w(A_0)
\end{aligned}
$$

If $A_0 \neq \langle\rangle$,

$$
\begin{aligned}
& (u_0, E_0) \to_{\mathcal{M}}^{A} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0 A_1 \ldots A_{2k+1}} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0} (u_{l_0}, E_0 \oplus w(A_0)) \dashrightarrow_{\mathcal{M}}^{*} (u_{f_1}, E_0 \oplus w(A_0)) \to_{\mathcal{M}}^{A_1 A_2 \ldots A_{2k+1}} (u_{n-1}, E_0 \oplus w(A)) \\
\implies & (u_{l_0}, E_0 \oplus w(A_0)) \dashrightarrow_{\mathcal{M}}^{*} (u_i, E_0 \oplus w(A_0)) \dashrightarrow_{\mathcal{M}} (u_{i+1}, E_0 \oplus w(A_0)) \dashrightarrow_{\mathcal{M}}^{*} (u_{f_1}, E_0 \oplus w(A_0)) \\
\implies & \{u_i, u_{i+1}\} \in E_0 \oplus w(A_0)
\end{aligned}
$$

In either case, $\{u_i, u_{i+1}\} \in E_0 \oplus w(A_0)$ as wanted. Note that we're exploiting the way the partition is defined: switches in A_0 are situated on the left side of the door $\{u_i, u_{i+1}\}$, whereas switches in A_1 are on the right, which means we have to pass through that door in between those sequences.

Now we show, $\forall j \in \{2,4,\ldots,2k\}$: $\{u_i, u_{i+1}\} \notin w(A_j)$, which means -except A_0- none of the *left subsequences* has any effect on the door.

Here again we deal with two cases: A_{j+1} is empty, or not.

If it's not empty, then we can safely assume we're passing through the door $\{u_i, u_{i+1}\}$ once from $u_{l_{j-1}}$ to u_{f_j}, and once again from u_{l_j} to $u_{f_{j+1}}$. The symmetric difference of the set of open doors during those passes is going to be $w(A_j)$, which means the

door $\{u_i, u_{i+1}\}$ could not have been affected by the switches in A_j (collectively).

$$
\begin{aligned}
& (u_0, E_0) \to_{\mathcal{M}}^{A} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0 A_1 \ldots A_{2k+1}} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0 A_1 \ldots A_{j-1}} (u_{l_{j-1}}, E') \dashrightarrow_{\mathcal{M}}^{*} \\
& (u_{f_j}, E') \to_{\mathcal{M}}^{A_j} (u_{l_j}, E' \oplus w(A_j)) \dashrightarrow_{\mathcal{M}}^{*} \\
& (u_{f_{j+1}}, E' \oplus w(A_j)) \to_{\mathcal{M}}^{A_{j+1} A_{j+2} \ldots A_{2k+1}} (u_{n-1}, E') \\
\implies & (u_{l_{j-1}}, E') \dashrightarrow_{\mathcal{M}}^{*} (u_{f_j}, E') \\
& (u_{l_j}, E' \oplus w(A_j)) \dashrightarrow_{\mathcal{M}}^{*} (u_{f_{j+1}}, E' \oplus w(A_j)) \\
\implies & \{u_i, u_{i+1}\} \in E' \\
& \{u_i, u_{i+1}\} \in E' \oplus w(A_j) \\
\implies & \{u_i, u_{i+1}\} \notin w(A_j)
\end{aligned}
$$

where $E' = E_0 \oplus w(A_0 A_1 \ldots A_{j-1})$.

If it's empty, then it must be the case that $j = 2k$. Because of that, even though A_{j+1} is empty and there is no $u_{f_{j+1}}$, we must pass through $\{u_i, u_{i+1}\}$ once from $u_{l_{j-1}}$ to u_{f_j}, and once again from u_{l_j} to u_{n-1}. Thus, the same type of argument applies in this case as well.

$$
\begin{aligned}
& (u_0, E_0) \to_{\mathcal{M}}^{A} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0 A_1 \ldots A_{2k+1}} (u_{n-1}, E_0 \oplus w(A)) \\
\iff & (u_0, E_0) \to_{\mathcal{M}}^{A_0 A_1 \ldots A_{j-1}} (u_{l_{j-1}}, E') \dashrightarrow_{\mathcal{M}}^{*} (u_{f_j}, E') \\
& \quad \to_{\mathcal{M}}^{A_j} (u_{l_j}, E' \oplus w(A_j)) \dashrightarrow_{\mathcal{M}}^{*} (u_{n-1}, E' \oplus w(A_j)) \\
\implies & (u_{l_{j-1}}, E') \dashrightarrow_{\mathcal{M}}^{*} (u_{f_j}, E') \\
& (u_{l_j}, E' \oplus w(A_j)) \dashrightarrow_{\mathcal{M}}^{*} (u_{n-1}, E' \oplus w(A_j)) \\
\implies & \{u_i, u_{i+1}\} \in E' \\
& \{u_i, u_{i+1}\} \in E' \oplus w(A_j) \\
\implies & \{u_i, u_{i+1}\} \notin w(A_j)
\end{aligned}
$$

where $E' = E_0 \oplus w(A_0 A_1 \ldots A_{j-1})$.

So far we have

$$
\{u_i, u_{i+1}\} \in E_0 \oplus w(A_0) \qquad \text{and}
$$
$$
\forall j \in \{2, 4, \ldots, 2k\} \colon \{u_i, u_{i+1}\} \notin w(A_j).
$$

By combining these, we get

$$\begin{aligned}
&\{u_i, u_{i+1}\} \in E_0 \oplus w(A_0) \oplus w(A_2) \oplus \cdots \oplus w(A_{2k}) \\
\iff &\{u_i, u_{i+1}\} \in E_0 \oplus w(A_0 A_2 \ldots A_{2k}) \\
\iff &\{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i}).
\end{aligned}$$

As can be seen, independent of $\{u_i, u_{i+1}\}$'s relative position, all three cases resulted with $\{u_i, u_{i+1}\} \in E_0 \oplus w(A_{\leqslant i})$. Thus, any valid sequence A from u_0 to u_{n-1} must also be permissible. □

Lemma 4.9. *Assuming* A *is permissible, then there is a strictly increasing sequence* A' *that is also permissible.*

Proof. Note that the definition of permissibility doesn't depend on the order of A because $\oplus$ is commutative. We can order A however we like, and it would still be permissible. Specifically, we're going to order it non-decreasingly. Let's call this A''.

Secondly, we can observe that any consecutive pair of equal elements in A'' could be removed from A'' pair by pair until there is no duplicate left in it. Each removal step here doesn't affect the permissibility since every set is its own inverse under $\oplus$. After there is no duplicate left in A'', it must be strictly increasing. □

Theorem 4.10. PathSD *is in* P.

Proof. Previously, we had an incomplete algorithm to decide PathSD. Given an instance of PathSD, we were constructing the system of linear equations, $W\mathbf{a} = \mathbf{e}$, in poly-time and then were deciding whether it has a solution by Gaussian-Jordan elimination in poly-time.

At that point, we weren't sure what to do if the system of linear equations doesn't have a solution since we only knew

$$\exists \mathbf{a} \in \mathbb{Z}_2^{n-1} : W\mathbf{a} = \mathbf{e} \implies \exists E' \subseteq E : (u_0, E_0) \to_{\mathcal{M}}^{*} (u_{n-1}, E').$$

In fact, there is no need to extend the algorithm since the converse is also true by Lemma 4.3, 4.4, 4.8, 4.9 and 4.7. □

5 Conclusion

In this work, we have shown by a reduction from NCL that $(4,1)$-SwitchDoor is PSPACE-complete. We have also showed that $(1,\cdot)$-SwitchDoor is solvable in linear-time. It's an open question whether $(k,\cdot)$-SwitchDoor is PSPACE-hard for $k = 2,3$. Since our construction was also an instance of $(4,1)$-ColourPlanarSD, we were able to demonstrate the intractibility of some Switches variants, that were previously unknown.

We've also focused on another variant called PathSD and have showed that it's in P by reducing it to a system of equations.

Bibliography

[1] Jonathan Gabor and Aaron Williams. Switches are pspace-complete. In *CCCG*, 2018.

[2] Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1):72 – 96, 2005. Game Theory Meets Theoretical Computer Science.

[3] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

[4] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.