A Framework for Proving the Computational Intractability of Motion Planning Problems

by

Jayson Lynch

B.S., Massachusetts Institute of Technology (2013)
M.Eng., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Department of Electrical Engineering and Computer Science
August 28, 2020

Certified by .....
Erik D. Demaine
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

A Framework for Proving the Computational Intractability of Motion Planning Problems

by

Jayson Lynch

Submitted to the Department of Electrical Engineering and Computer Science
on August 28, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

This thesis develops a framework for proving computational complexity results about motion planning problems. The model captures reactive environments with local interaction. We introduce a motion planning problem involving one or more agents that move around a connection graph and through "gadgets" which are stateful parts of the environment whose state and traversability can change only in response to traversals of the agent within the gadget. The model includes variants for 0-player, 1-player, 2-player, and team imperfect information games.

This thesis considers various classes of gadgets and give both algorithms and hardness results ranging from NL-completeness to Undecidability. Full dichotomies are obtained for some classes including the natural class of gadgets which can be traversed a bounded number of times. For 1-player this gives a separation between containment in NL versus NP-completeness, for 2-player a separation between containment in P and PSPACE-completeness, and for team imperfect information games a separation between containment in P and NEXPTIME-completeness.

Our model builds on and generalizes several other proof techniques for motion planning problems and games. This thesis also provides examples of how this new framework can simplify many of those old results, as well as applying to many new hardness results for video games and variants of block pushing puzzles. New hardness results include PSPACE-hardness for Trainyard, Sokobond, The Legend of Zelda: Breath of the Wild, The Legend of Zelda: The Minish Cap, The Legend of Zelda: Oracle of Seasons, Captain Toad: Treasure Tracker, Super Mario Oddsey, Super Mario Galaxy 1 and 2, Super Mario Sunshine, and Super Mario 64.

Thesis Supervisor: Erik D. Demaine
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I'd like to thank my advisor, Erik Demaine, who I've been working with since I was an undergraduate. It has been a joy working together and I appreciate all the care, understanding, help, and insights he's provided over the years.

I would like to thank all of my many coauthors and collaborators. I'd especially like to thank those who've provided mentorship, advice, or guidance while I've been a student. Some of these people include: Zach Abel, Michael Bender, Marty Demaine, Sarah Eisenstat, Erik Klopfer, Matias Korman, Ara Kooser, Jason Ku, Tao B Schardl, Mark Smith, and Virginia Vassilevska Williams.

I would like to thank my coauthors and collaborators whose ideas, work, and writing is represented in this thesis: Joshua Ani, Sualeh Asif, Jeffrey Bosboom, Michael Coulombe, Yevhenii Diomidov, Jonathan Gabor, Isaac Grosof, Dylan Hendrickson, Ama Koranteng, Lorenzo Najt, Mikhail Rudoy, Sarah Scheffler, and Adam Suhl. From this perspective I would especially like to thank Erik Demaine, Isaac Grossof, and Dylan Hendrickson, without whom the motion planning with gadgets model would never have come about. I especially want to acknowledge Dylan Hendrickson, who is an author on almost all of the papers represented and has contributed a huge amount of work to this project. I would not be surprised if a plurality of the proofs here were primarily composed by Dylan. I also want to specifically acknowledge Joshua Ani, who has contributed significantly to the more recent work on this topic.

I would like to thank my thesis committee: Erik Demaine, Virginia Vassilevska Williams, and Ryan Williams, for their feedback and advice on this thesis and on other work.

I'd like to thank my family, especially my parents Anita Lynch-Gallagher, Erik Gallagher, Robert Lynch, and Le Anne Lynch, whose love and support has been very important my whole life.

I would like to thank Andrea Lincoln for being a great friend and collaborator throughout my time as a graduate student.

I would like to thank Krue for all their friendship and support. They have been an

important part of my life the entire time I've been a graduate student. I would also like to thank Krue's family for their friendship and support.

I would like to thank K. E. Peterson, who has been an important part of my life for the last few years, for her support, friendship, and for help in editing this thesis.

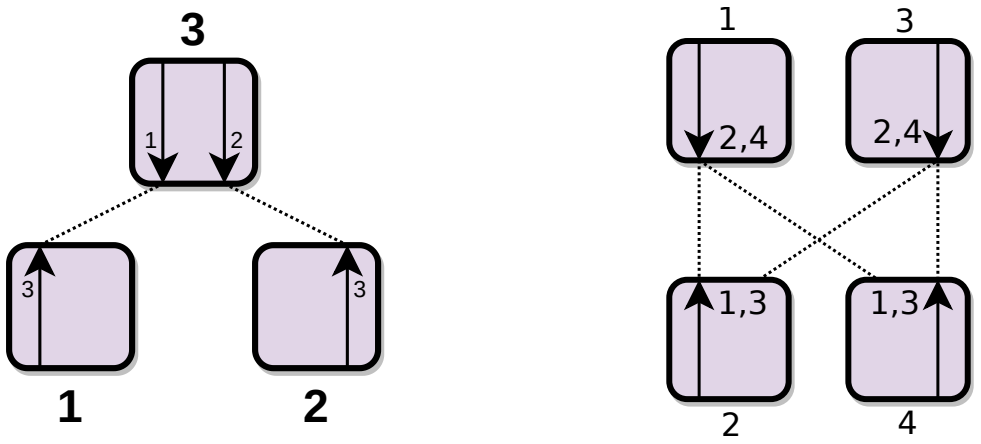
I would like to thank all of my friends and colleagues who have made MIT a wonderful place to be over the last twelve years. You all have created a home for me as well as a great environment to learn and do research.

Chapter 1

Introduction

Many aspects of the world we live in are well modeled by a semi-static environment, i.e. one which only changes in response to actions we or other agents perform upon it. We expect inanimate objects like ladders, keys, and cups of tea to stay in the same places we last left them. Given how common and mundane this experience of the world is, it is not surprising that this assumption of stasis between actions shows up in many abstract models of the world, whether they be models for robotic planning, simplifying assumptions to algorithms, or aspects of video games. Once we start considering abstract models, it is very natural to ask about the computational complexity of motion planning in such an environment. This thesis describes a general model to help understand the computational complexity of motion planning in these semi-static environments. It provides a formalization and generalization of prior techniques used to prove hardness results about agent based motion planning problems.

Our model additionally has the property of interaction being local, captured by restricting changes in the environment to be constant-sized ***gadgets***, where the gadget changes state when it is traversed by an agent according to a general transition function, enabling and/or disabling certain traversals in the future. In general, we model a ***gadget*** as consisting of a finite number of ***locations*** (entrances/exits) and a finite number of ***states***; see Figure 1-1 for two examples. An ***agent*** has a ***location*** and is able to take ***transitions*** in a gadget, changing the gadget's state and the agent's location. Each state s of the gadget defines a



(a) Locking 2-toggle

(b) Nondeterministic locking 2-toggle

Figure 1-1: State diagrams for the Locking 2-Toggle and the Non-deterministic Locking 2-Toggle.

labeled directed graph on the locations, where a directed edge (a, b) with label s' means that an agent can enter the gadget at location a and exit at location b, and that such a traversal forcibly changes the state of the gadget to s'. These gadgets are further connected together by a ***connection graph*** among their locations. The agent is able to freely move along the connection graph outside of the gadgets.

We study this model primarily from the single-robot (one-player) perspective, but we also introduce variations that look at two robots or teams of robots competing to reach their respective goals and a 0-player model where the agent's next location is fully deterministic. This model's simplicity and locality typically allow for "local replacement" [40] or "gadget based" reductions, allowing us to simplify several prior hardness proofs. Yet this model is also general enough to capture a range of situations and to generate interesting complexity questions in its own right. Applications of this model include hardness for block-pulling puzzles [11,23], models of microassembly with global control [11,15], generalizations of switching graphs [33], and generalizations of well known video games including Mario Kart, Zelda, and Portal (Section 5).

Our objective in studying these gadgets is twofold. First, we wish to characterize the computational complexity of motion planning problems whose input is world comprised of a system of such gadgets and a starting and goal location for an agent, and whose output is

Table 1.1: Table of complexity class containment based on the type of game and whether the number of moves is bounded.

	0-Player	1-Player	2-Player	Team
Polynomially Bounded	**P-complete**	**NP-complete**	**PSPACE-complete**	**NEXPTIME-complete**
Polynomially Unbounded	**PSPACE-complete**	**PSPACE-complete**	**EXPTIME-complete**	**RE-complete (Undecidable)**

whether the agent can reach the target location from the starting location. This ambitious goal would ideally generate a full characterization similar in spirit to Schaefer's Dichotomy Theorem for boolean CSPs [55]. Second, we want to find and highlight gadgets that seem natural and easy to construct in other systems, furthering the goal of this model being a useful intermediary for proving hardness results.

Inspired by past work on the complexity of games, as well as the formalization of Constraint Logic [44] in 0-player, 1-player, 2-player, and Team Imperfect Information variations, we define 0-player, 1-player, 2-player, and Team Imperfect Information versions of our motion planning model and consider many of the prior classes of gadgets in those variants, yielding problems with tight complexity bounds for the generated classes.

To this end we fully characterize a few special classes (infinite in size) of gadgets. For every class of gadget we consider, we typically find that it is either "trivial", meaning it is no harder than the corresponding problem on a static graph, or "as hard as possible" given the type of game and whether the number of moves is polynomially bounded. Table 1.1 gives this breakdown. There are some notable exceptions though; for example the undirected door-opening gadget, turns out to be P-complete for the 1-player problem, see Section 2.5. In addition, deterministic single-input input-output gadgets, ones in which there is only one location from which the agent can travel, are all contained in NP $\cap$ coNP in the 0-player model, even though we might expect some to be PSPACE-complete, see Section 4.1.

One major distinction is whether gadgets have several state transitions which are ***polynomially bounded*** or whether they are ***unbounded***, meaning the agent can traverse the

gadgets in such a way that the gadgets can continue changing state forever.

For gadgets that can only be traversed a polynomial number of times (DAG gadgets), we give a full characterization. We classify a significant portion of the more general case of gadgets with polynomially bounded state changes (LDAG gadgets). We also show how the "shortest-path victory condition", where we ask if the agent can reach the goal while traversing gadgets no more than k times, simplifies the characterization of polynomially bounded gadgets.These results are summarized in Table 1.2.

For unbounded gadgets, we consider three main categories. First, we give a full characterization of reversible, deterministic gadgets. Second, we study planar door gadgets and self-closing door gadgets inspired by the "door gadget" of Viglietta [4,61]. Third, we investigate simple input/output gadgets where locations the agent can enter the gadget are distinct from locations in which the agent can exit. These gadgets are inspired by switching graphs and models and puzzles about train routing.

Since many applications of this model will be in an approximately 2D environment, we often consider the case of planar layouts of the gadgets. Frequently we are able to show that the gadgets remain as computationally intractable in a planar setting as in the general setting. One particularly interesting case we explore are door gadgets which were useful tool in many prior proofs. We show PSPACE-completeness for all but one of the many planar cases of door gadgets, leaving an NP-hard to PSPACE gap for the last one. This allows the elimination of the crossover gadget from these past proofs, simplifying the constructions needed. We also show all planar variations of the self-closing door gadget are PSPACE-complete, see Section 2.6, and use this new, simpler gadget to prove several new hardness results about video games, see Section 5.

In addition to problems concerned with ***reachability***, where the decision problem is whether the agent can reach the target location in a given system of gadgets, we also consider several variations. We examine the complexity of the ***reconfiguration problem*** which asks whether the agent can cause the system of gadgets to reach a target state. We show a gadget with non-interacting tunnels whose reconfiguration problem is PSPACE-complete,

Table 1.2: Results for polynomially bounded gadgets

	0-Player	1-Player	2-Player	Team
DAG	**partial (§4.2)**	**full (§2.3.5)**	**full (§3.3.1)**	**full (§3.3.2)**
LDAG	**partial (§4.2)**	**partial (§2.3.7) full for shortest-path (§2.3.6)**		
2-state IO	**full (§4.2)**	**full (§2.5)**		

Table 1.3: Results for polynomially unbounded gadgets

	0-Player	1-Player	2-Player	Team
reversible deterministic		**full (§2.2) planar (§2.2.4)**	**partial (§3.2.1)**	**partial (§3.2.2)**
2-state IO	**full (§4.3)**	**full (§2.5)**		
Doors		**full (§2.6) partial planar (§2.9.3)**		
Single input IO	**partial (§4.1)**	**partial (§2.5)**		

even though the reachability problem for any gadget with non-interacting tunnels is in NL. We also show that, for reversible deterministic gadgets, PSPACE-completeness of the reachability problem implies PSPACE-completeness of the reconfiguration problem. In contrast, we exhibit a gadget for which the reconfiguration problem becomes easier, contained in P, whereas reachability is NP-complete. We also consider the ***shortest-path problem***, which asks whether the agent can reach the target location while crossing less than k gadgets. For the shortest-path problem we show a class of gadgets whose reachability problem is in P which becomes NP-complete.

The results of this thesis are an amalgam of the research in [6–9, 12, 25, 29] and represent work by a large number of people. My collaborators and coauthors whose ideas and words

are represented here include: Joshua Ani, Sualeh Asif, Jeffrey Bosboom, Michael Coulombe, Erik Demaine, Yevhenii Diomidov, Jonathan Gabor, Isaac Grosof, Dylan Hendrickson, Ama Koranteng, Lorenzo Najt, Mikhail Rudoy, Sarah Scheffler, and Adam Suhl. Dylan Hendrickson in particular has been a central figure in almost all of these papers.

1.1 Model

In our model ***gadgets*** consist of ***locations***, ***states***, and ***transitions*** between state and location pairs. ***Agents*** can be at locations and can move across gadgets via transitions changing the gadget's state and the agent's location.

A gadget can be specified by its ***transition graph***,[1] a directed graph whose vertices are state/location pairs, where a directed edge from (s, a) to (s', b) represents that the agent can traverse the gadget from a to b if it is in state s, and that such traversal will change the gadget's state to s'. Gadgets are ***local*** in the sense that traversing a gadget does not change the state of any other gadgets.

A ***state graph*** of a gadget is a graph with a vertex for every state of the gadget with directed edges between vertices where there exists a transition that takes that gadget from one vertex's state to the other's. Many gadgets will share the same state graph. One additionally needs to annotate the edges with ordered pairs of locations to fully specify the gadget. The state graph of a gadget can be obtained by collapsing all of the state-location nodes in a transition graph which share the same location.

The ***traversability*** between an ordered pair of locations (a, b) in a gadget with specified state (often implicitly the current state) refers to whether at least one transition from a to b in that gadget in that state exists. The traversability of a gadget in a specified state is the set of traversability of all pairs of that gadget in that state. A ***traversal*** from location a to location b is any transition from location a to location b. There is a subtle different between transitions and traversals which is mostly relevant when we want to discuss moving

[1] In [25], the transition graph is called the "state space", but we feel that "transition graph" more clearly captures the automaton nature of transitions, which are discrete and directed.

from one location to another location across multiple states of a gadget. These transitions by definition must be different (since they have different starting states) but they may have the same traversals.

A ***system of gadgets*** consists of gadgets, their states, and a ***connection graph*** on the gadgets' locations. [2] If two locations a, b of two gadgets (possibly the same gadget) are connected by a path in the connection graph, then the agent can traverse freely between a and b (outside the gadgets). (Equivalently, we can think of locations a and b as being identified, effectively contracting connected components of the connection graph.) Since an agent can move freely along any path in the connection graph, connection graphs whose connected components are the same are essentially equivalent. These are all the ways that the agent can move: exterior to gadgets using the connection graph, and traversing gadgets according to their current states. A ***configuration*** is a system of gadgets along with a set of agents and locations for those agents.

The primary problem considered in this thesis is ***1-player reachability motion planning***. This problem consists of a system of gadgets, a start location, and a set of goal locations. The question is whether there exists a series of moves which takes an agent from the start location to one of the goal locations. Different decision questions and models are described in Sections 1.1.2, 1.1.3, and 1.1.4.

1.1.1 Diagrammatic Representations

We will frequently depict a gadget as a box or a circle, labeled with a state, and with transitions between locations depicted as arrows or lines (depending on whether they are directed) which are also labeled with the states that they transition the gadget into. Deterministic gadgets will always have one state label per arrow, while non-deterministic ones may have a list of states. Gadgets we deal with often have ***embodied state***, where the traversability in a state (and thus the configuration of arrows) is unique to that state. In these cases we may omit the state label. Further, since many gadgets have both embodied state and are

[2] In [25], locations could only be matched to exactly one other location and a "branching hallway" gadget was introduced to fulfill the need of the connection graph.

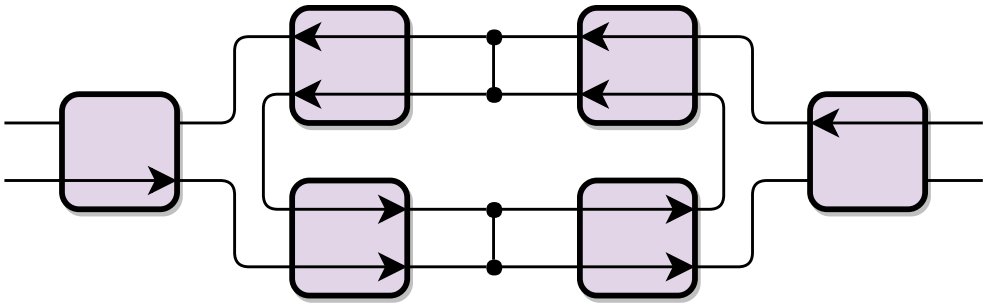


Figure 1-2: A diagram of a system of gadgets involving Locking 2-Toggles

deterministic, we will also frequently omit the labels on the transitions.

To define a gadget, we can give a series of these diagrams, one for each state. This is often easier to interpret than a list of transitions or a transition graph. In addition, the diagrams can sometimes be arranged so the output location after a traversal has a dotted arrow to the state diagram it corresponds to.

For example, Figure 1-1 shows the state diagrams of the Locking 2-Toggle and the Nondeterministic Locking 2-Toggle. The ***Locking 2-Toggle***, shown in Figure 1-1a, is a 3-state 2-tunnel reversible deterministic gadget. In state three, it has two directed traversals. Going over either traversal flips its direction and closes the other tunnel. From states one and two, each has a single transition and that transition brings the gadget back to state three. The ***nondeterministic locking 2-toggle***, shown in Figure 1-1b, is a four-state gadget where each state has two transitions, each across the same tunnel. The top pair of states each allow a single traversal downward, and allow the agent to choose either of the two bottom states for the gadget. Similarly, the bottom pair of states each allow a single traversal upward to one of the top states.

The figure below shows a system of locking 2-toggles. The boxes and transitions are unlabled since the locking 2-toggle has embodied state and thus the state of the gadget in each case is clear from the diagram. Further, we sometimes show the connection graph as a subdivided connection for clarity. The black dots along the middle axis of the diagram denote a connection between those ports.

1.1.2 Planarity

Since many of the applications of the model come from 2D environments, it is natural to restrict our model to better respect the resulting mobility and connectivity constraints from such a 2D environment. Thus we define ***planar motion planning*** where the cyclic order of locations on a gadget is specified, and the system of gadgets must be embedded in the plane without intersections. Specifically, construct the following graph from a system of gadgets: replace each gadget with a wheel graph, which has a cycle of vertices corresponding to the locations on the gadget in the appropriate order, and a central vertex connected to each location. Connect locations on these wheels with edges according to the connection graph. The system of gadgets is ***planar*** if this graph is planar. In planar motion planning, we restrict the problem to planar systems of gadgets. Note that this allows rotations and reflections of gadgets, but no other permutation of their locations. In some contexts, one may want to disallow reflections of gadgets, which corresponds to imposing a handedness constraint on the planar embedding of each wheel. One could allow more general embedding, however, in our experience the proofs that use this framework have an easy time reflecting the gadgets constructed by cannot build all of the combinatorial orderings of locations.

1.1.3 Victory Conditions

We typically consider the problem of whether the agent can reach a target location, or, in a multiplayer setting, whether one team's agent can reach a target location before the other team has an agent reach their target location. These are called ***reachability*** motion planning problems. Some useful alternatives are considered in this thesis. These variants are all only explored for the 1-player model, but natural generalizations for 0-player and multi-player versions exist.

In ***shortest-path*** motion planning, we ask whether there exists a series of at most k moves which bring the agent to the target location. Here is will be important to specify whether k is of polynomial size (thus putting the 1-player problem in NP) or if it can be exponential size (in which case one may only be proving weak-NP or weak-PSPACE-hardness

since the input has numbers of exponential size).

In ***reconfiguration*** motion planning, we are given a system of gadgets, a start location, and a target configuration for that system of gadgets. We ask whether there exists a series of moves after which the system of gadgets is in the target configuration. One could additionally specify a target location for the agent, though this seems unlikely to matter. One could also consider the version where only some of the gadgets have specified target states. This more general notion may make proving hardness easier, although we are able to obtain the stronger result for full reconfiguration in all cases.

1.1.4 Number of Players

Altering the number of players in a motion planning problem can drastically change its complexity. The general categorization follows the same one noted in earlier games, for example Constraint Logic [44]. The single player case is the primary case considered and is described above; here we give more detail on zero and multiplayer models.

In ***1-player reachability motion planning*** we are given a system of gadgets, the start location for the player, and a set of goal locations for that player. The decision question is then: *does there exist a series of traversals which takes the player from their start location to one of the goal locations?* Results in this model are discussed in Chapter 2.

In ***2-player reachability motion planning*** we are given a system of gadgets, a start location for the ***Existential Player*** (Player 1), a start location for the ***Universal Player*** (Player 2), a set of ***goal locations*** for the Existential Player, and a set of goal locations for the Universal Player. Each player takes turns, starting with the Existential Player, moving through the system of gadgets with the same rules of traversability as the 1-player model. Here we only count transitions through gadgets as moves, and allow free movement along the connection graph. The Existential Player wins if they reach one of their goal locations before the Universal Player has reached any of their goal locations. The decision question is whether, under perfect play, the Existential Player can force a win. Results in this model are discussed in Chapter 3.

The ***team reachability motion planning*** problem is similar to the 2-player case. We now have two teams, the Existential Team and the Universal Team, each of whom has a set of players with start locations and a set of goal locations. We also have an ordering for the players each of whom take turns moving in the system of gadgets. The Existential Team wins if some player on the Existential Team reaches one of the Existential goal locations prior to any one the players on the Universal Team reaches any of the Universal goal locations. In the perfect information setting, this is essentially equivalent to a 2-player game where each player controls multiple agents, since, under perfect play, the players will be able to coordinate perfectly.

Thus, we also consider the ***team imperfect information reachability motion planning*** problem. In addition to everything in the team reachability motion planning problem, we also have a visibility function for each player which maps the state of the world into the gadgets. Each player is given access to the states of the gadgets returned by their visibility function on the current state of the world. Normally we will restrict the visibility function to either be a constant function (each player always knows the state of some portion of the gadgets and is never told the state of any other gadgets) or only a function of the player's location. The constant visibility function is what is used in most prior models, for example Team Constraint Logic [44], Team Computation Game, Team Private Peak, or Team Formula Game [51]. Here we will use the visibility model in which the player is able to see the state of any gadget which is connected to their current location. Note this visibility function is highly sensitive to the difference between defining a system of gadgets as a set of connections between locations (the current one used) versus a matching on locations with the addition of the branching hallway gadget (the original definition in [24]). Results in this model are discussed in Chapter 3.

In ***0-player reachability motion planning*** we want to capture a model in which there is no player choice but we still have an agent moving mindlessly forward through a system of gadgets. Thus we restrict the location connectivity graph to be a matching. We call a system of gadgets whose connectivity graph is a matching ***branchless***. After an agent follows a

transition in a gadget, it immediately enters the location connected which is connected in the matching and takes one of the available transitions in that new gadget. Thus the agent is never able to choose where to go between gadgets or to "turn around" after exiting a gadget. If all gadgets are themselves deterministic, then at no point will the agent ever have a choice about what location to go to. This currently leaves behavior undefined when the agent enters a location with no transitions or exits a location that is not connected to another location. One natural resolution is for the agent to stop moving and become "stuck" if this ever occurs, thus preventing the agent from winning if they have not already done so. Another resolution is to have the agent "bounce off" the gadget or unconnected location, causing it to reenter the location it last exited. This is used in a model of reversible computation [36] which resembles our 0-player model. We partially avoid the issue by considering a class of gadgets and constructions which prevent this situation from ever arising. Results in this model are discussed in Chapter 4.

1.1.5 Classes of Gadgets

Here is a description of all of the different types of gadgets considered in this thesis.

A gadget is ***deterministic*** if every traversal can put it in only one state and every location has at most one traversal from it. More precisely, its transition graph has maximum out-degree 1.

A gadget is ***reverse-deterministic*** if its transition graph has maximum in-degree and out-degree of one.

A gadget is ***reversible*** if every transition can be undone after being taken. More precisely, for all transitions t from location l_1 and state s_1 to location l_2 and state s_2 there exists a transition t' from location l_2 and state s_2 to location l_1 and state s_1.

Reversible deterministic gadgets are thus gadgets whose transition graphs are matchings. They are also reverse-deterministic.

DAG gadgets are gadgets whose state graph forms a directed acyclic graph (dag).

LDAG gadgets are gadgets whose state graph which forms a dag with self-loops.

DAG gadgets are a class of naturally bounded gadgets, in that they can only be traversed a bounded number of times. LDAG gadgets represent a large class of gadgets whose 1-player motion planning problem is in NP, as the state transitions of the gadgets are still bounded.

Eventually static gadgets which are a class of LDAG gadgets where the self-loops are only allowed in states which are sinks in the state graph. These are a special case of LDAG gadgets and DAG gadgets are a special case of eventually static gadgets.

A ***k-tunnel*** gadget has $2k$ locations, which are partitioned into k pairs called ***tunnels***, such that every transition is between two locations in the same tunnel. In this thesis we will primarily be considering k-tunnel gadgets and thus this will often not be specified. We will sometimes refer to a k-tunnel gadget as a gadget which is ***on tunnels***.

An ***input/output*** gadget is one whose locations can be partitioned into ***input*** locations (entrances) and ***output*** locations (exits) such that every traversal brings an agent from an input location to an output location, and in every state, there is at least one traversal from each input location. In particular, deterministic input/output gadgets have exactly one traversal from each input location in each state. Note that input/output gadgets cannot be reversible nor DAGs.

An input/output gadget is ***output-disjoint*** if, for each output location, all of the transitions to it are from the same input location. This notion is more general than k-tunnel for input/output gadgets because it allows a many-to-one relation from a single input to multiple outputs.

A ***door gadget*** is a 2-state, 3-tunnel gadget which has a "traverse" tunnel, an "open" tunnel, and a "close" tunnel. The traverse tunnel is only traversable in the "open" state. The close tunnel has traversals in both states and both traversals set the state to "closed". The open tunnel has traversals in both states and those traversals (optionally) set the state to open. There are 8 door gadgets based on whether the tunnels have directed or undirected transitions and whether the door opening is optional.

A ***self-closing door gadget*** is a 2-state, 2-tunnel gadget which has an "open tunnel" and a "self-closing tunnel". The self-closing tunnel only has transitions from the "open" state

to the "closed" state. The open tunnel has transitions which set the state of the gadget to "open". There are 4 self-closing door gadgets based on whether the two tunnels are directed. Sometimes we will refer to door gadgets and self-closing door gadgets collectively as door gadgets.

A ***monotonically opening*** gadget is one in which the traversability of a gadget never decreases as transitions are taken. More formally, for all states s, if there is a transition from location l_1 to location l_2 then for all states reachable from s there is some transition from location l_1 to location l_2.

A ***monotonically closing*** gadget is one in which the traversability of a gadget never increases as transitions are taken. More formally, for all states s, if there is not a transition from location l_1 to location l_2 then for all states reachable from s there is not a transition from location l_1 to location l_2.

An ***unchanging*** gadget is one in which the traversability of a gadget never changes. More formally, for all states s if there is a transition from location l_1 to location l_2 then for all states reachable from s there is some transition from location l_1 to location l_2 and if there is not a transition from location l_1 to location l_2 then for all states reachable from s there is not a transition from location l_1 to location l_2. 1-state gadgets are trivially unchanging as the traversability of a gadget cannot change if it cannot change state.

A gadget with ***embodied state*** is one in which no two states of a gadget have the same traversability. More formally, for all pairs of states s_i and s_j in the gadgets, there exists a pair of locations l_a and l_b such that there is at least one transition from l_a to l_b in exactly one of s_i or s_j and no transition from l_a to l_b in the other state. Thus each state can be uniquely determined simply by looking at the gadget's traversability (as opposed to needing to know the transitions).

1.2 Related Work

We build on four main bodies of work: the Constraint Logic formalism of Hern and Demaine [45], the doors-and-buttons framework of Forišek [35] and Viglietta [60], the Mario/Portal

framework [5,31], and the door-gadget framework [16]. These papers all provided important ideas and computationally hard problems, but were insufficient for our target purpose in some fashion. Constraint Logic was not an agent-based model, the doors-and-buttons and door-gadget frameworks were insufficiently general, and the Mario/Portal framework was not sufficiently formal serving merely as a guideline for how to set up a reduction. In the following section we summarize some of this prior work and also compare it to our model.

1.2.1 Constraint Logic

Constraint Logic is a type of constraint satisfaction on directed weighted graphs. In this model, edges have weights and vertices have a requirement that the sum of the weights of the edges directed towards the vertex exceeds a target value. A common question studied on such graphs is ***reconfiguration***: given a satisfied graph, and the ability to flip the directions of edges one at a time, does there exist a sequence of flips in which the graph remains satisfied and some target edge is flipped? Both bounded and unbounded 0-player, 1-player, and team imperfect information version of constraint logic games have been defined [45]. Details on these problems are given in Section 1.4.1. One of the main useful features was showing that the unbounded, 1-player version of the problem is still PSPACE-complete even when the graph consists of max-degree 3 vertices which all require weight 2 and have either three weight-2 edges or one weight 2 and two weight 1 edges. Further, the problem remains hard when the constraint graphs are planar. Other versions of the problem remain hard with similar restrictions.

This problem was originally introduced in [44] to aid in proving the PSPACE-hardness of sliding block puzzles. Constraint Logic is a reconfiguration problem where the moves are reversible and global, just like sliding block puzzles. Constraint Logic become a useful tool in proving many more hardness results about games and puzzles [28, 45, 47, 58], graph and shape reconfiguration [14,46], and robotic swarm motion planning [10,15]. However, in some of these use cases, the environment is changed by a localized agent acting on its environment. In this case the global nature of moves in constraint logic becomes a barrier. One of the

motivations in defining our model was to create an agent-based version of constraint logic to make proofs of these types of problems easier. This is also one of the reasons that reversible deterministic gadgets were the first class of gadget we examined.

We use unbounded 1-player, 2-player, and Team versions of constraint logic in proving hardness for k-tunnel reversible deterministic gadgets in those models. These can be found in Sections 2.2 and 3.2.1

1.2.2 Doors-and-Buttons Model

The doors-and-buttons model was defined and explored by Forišek [35] and Viglietta [60], and the main remaining questions posed about the model were resolved by Van Der Zanden and Bodlaender [58].

This model involves an agent in a 2D square grid attempting to reach a target location. In the environment there are walls and ***doors*** which block the agent's movement. However, a door only blocks an agent while it is "closed". Around there environment there are also ***buttons*** which the agent can press can "open" or "close" doors to which it is connected. In particular each button has an associated set of doors and for each of those doors either "open" or "close". When pressed, those doors are set to the corresponding open or closed states. Each button exists in a tile in the environment and if the agent is on the same tile, the agent is free to press the button. The model also considers ***pressure plates*** which act exactly like doors except that they are pressed if the agent ever moves into the same square.

One restriction studied in this series of papers is the size of the button's set of doors and the number of button sets each door can appear in. We parameterize this by k-button-j-door to mean each button can effect no more than k doors and each door can have no more than j buttons change its state. We could similarly replace "button" and "door" with "pressure plate" and "trapdoor". Prior work then gives the following dichotomies.

1. The 1-button-d-door for all $d \geq 1$ with crossovers is P-complete [60].

2. The c-button-1-door for all $c \geq 2$ is NP-complete [60].

3. The c-button-d-door for all $c \geq 2$ and $d \geq 2$ is PSPACE-complete [58].[3]

There is a similar characterization of pressure plates from [60].

1. The $\geq$ 1-pressure plates-1-door with crossovers is P-complete.

2. The 1-pressure plates-$\geq$ 1-door is NP-complete.

3. The $\geq$ 2-pressure plates-$\geq$ 2-door is PSPACE-complete.

The papers also provide various other NP-hardness results for keyed-doors or toll bridges that open when some item or quantity of items is collected. These results are directly used to prove hardness for ten video games in those papers, as well as several works that directly use those theorems provided [5, 29, 31, 56]. However, the ideas and adaptations of the techniques can be seen in several papers and are likely even more important than the model and meta-theorems provided.

In [58] there are two extensions of the doors-and-buttons model. One is the notion of a ***trapdoor*** which prevents movement onto a square rather than between squares. The other is a ***switch*** which has an associated set of doors and swaps their current state between open and closed whenever it is pressed. Buried in the proof that motion planning with 2-buttons-2-doors is PSPACE-complete, the problem of motion planning with 1-switch-2-doors, where each switch is only connected to two doors and each door is only connected to one switch, is also shown to be PSPACE-complete. This result has been useful in several PSPACE-completeness proofs including Portal [31], Zelda [12], and several results in this thesis. A more recent paper [38] defines a model almost identical to doors-and-buttons with switches without being aware of the work of [58]. It shows that motion planning with switches and doors remains PSPACE-complete even when restricted to maps of height 3 in a square grid. These sorts of extensions lead to a more general model which we describe in Section 1.3.

This framework has also been used to prove hardness results about Offspring Fling, Back to Bed [17], Zelda: Breath of the Wild [12], and Portal [31].

[3] [60] had resolved this for $c \geq 3$ and $d \geq 2$.

1.2.3 Mario/Portal Framework

The Mario/Portal Framework gives a methodology for constructing NP-hardness proofs for videogames. The original paper [5] lays out how to use enemies that can be defeated from afar (for example, in Mario this is done by sending a Koopa shell through a passage Mario cannot go through) to build a reduction from 3SAT. It is inspired by the technique for hardness of PushPush block puzzles in [21]. It involves constructing one-way gadgets, crossover gadgets, variable gadgets, and clause gadgets. One-way gadgets are usually trivially given by ***long falls***, ones which the agent cannot go back up. For variable gadgets the agent is given a choice of going down one of two paths, enforced by the one-way gadgets, and along those paths are locations where the agent is able to successfully dispatch an enemy from a safe location. The enemies are guarding some other path, and for the clause gadget we place a choice of three paths each guarded by an enemy which can be dispatched from the associated variable gadget. The paper applies this framework to show NP-hardness (via a reduction from 3SAT) of generalized versions of classic SNES games in the Mario, Zelda, Donky Kong, and Metroid series. This framework is later used by others to show NP-hardness results for Portal [31], Mario Kart [13], Wings of VI [49], and Fire Emblem [39].

In [31] it is noted that the key properties are that there is an enemy blocking a traversal, and there is a way to remove that enemy from a separate location. (At this point, this should have been seen as a more general interpretation, or perhaps an application of Forišek's Metatheorem 1 [35]) The paper goes on to describe several common relationships between characters, enemies, and environments which fulfill this criteria, such as having stronger short-range or long-range weapons, or having asymmetric visibility. It further lists several games for which hardness should follow fairly simply.

This framework shows up in Section 2.3. We believe it is much simpler to now view these constructions as consisting of diode, crossover, and door-opening gadgets; where the door opening gadget is serving the role of the literals in the formula. With this view, each of the variable and clause gadgets can be simplified and arguments about the layout and overall structure are no longer needed.

1.2.4 Door-Gadget Framework

The Door-Gadget framework introduced in [61] and simplified and used with great success in [5] is in some sense the prototypical gadget which the motion-planning-through-gadgets framework is modeled after.

The notion of "opening and closing a door" is preserved from the doors-and-buttons framework, but using notions of traversable pathways as is seen in parts of [35] but also treats the notion of "door" as abstract traversability as seen in the Mario/Portal framework. It also gives a framework for proving PSPACE-completeness which was also provided by the doors-and-buttons framework but was lacking in Metatheorem 1 of [35] and the Mario/Portal framework.

This framework was used to prove PSPACE-hardness for Lemmings [61], Donky Kong Country 1-3, Legend of Zelda: A Link to the Past [5], Super Mario Bros. [32], Fire Emblem [39] and The Witness [1]. It was also extended to a 2-player game framework to prove EXPTIME-completeness of Xiangqi, Janggi [65], and Dou Shou Qi [64].

This technique is brought into our framework and studied in more depth in Section 2.6. It has lead to hardness results for additional Mario (Section 5.2), Zelda (Section 5.3), and block-pushing puzzle games (Section 5.1).

1.3 Extended Doors and Buttons Model

In the first three papers laying out the doors-and-buttons model [35, 58, 60] we already see the papers pushing at the boundaries of what was allowed in prior models: distinguishing optional and mandatory use with buttons and pressureplates, having doors live between tiles versus trapdoors taking up entire tiles, and switches swapping door state rather than setting it to a fixed value. This suggests the following simple generalization.

In the ***generalized doors-and-buttons*** model we are given a graph with a start location and goal location. The edges and vertices are labeled with ***activators*** and ***effectors*** which are analogs of buttons and doors. The effectors have two states if they are on a vertex:

"open" and "close". The effectors have four states if they are on an edge: "open", "close", "directed a to b", and "directed b to a" where a and b are the edge's vertices. When the agent crosses an edge with an activator or enters a vertex with an activator, the agent must choose one "action set" from a set of allowed action sets for the activator. An action consists of a target effector and a function from effector states to new states. So common actions might be to set the state of effector e to closed, or to swap the state between "directed a to b", and "directed b to a". Activators are optional if one of the action sets is the empty set. Thus our buttons in the original doors-and-buttons model become vertex activators with two action sets, the empty set and another set with functions that map to one value, either open or closed. Now we still have most of the simplicity and intuition of the doors-and-buttons model, but can express a fuller space around the individually defined aspects.

From this view, we can see that many gadgets in the motion-planning-through-gadgets framework are the same as edge activators and edge effectors with strong restrictions on connectivity. The door gadget is comprised of activators whose action set is either to set an effector closed or set an effector open, and which can connect to no more than one effector. The effectors are restricted to connect to no more than two activators.

With this generalization, there is an interesting question that arises about the computational complexity of switches connected to only one door. In [58] it was shown that having a vertex activator connected to no more than two edge effectors which switched the states of those effectors between open and closed, and had effectors connected to no more than one activator was sufficient for PSPACE-completeness. If one reduces that case to the case of an activator only connecting to one edge effector, it is unknown whether this remains hard but seems likely and we conjecture it to be in P. In particular if it is an optional activator this is known to be in P from [60]. With a vertex activator, having the freedom to step off and then on again will make the activator effectively optional. In contrast, the toggle-lock, described in Section 2.2.6, is essentially an edge activator connected to one edge effector and is known to be PSPACE-complete. In this case the edge activator is very different because in some sense it remembers which side the agent was on and requires completing a cycle in

the graph to flip its state rather than just having an adjacent vertex.

It is useful to note that neither the gadgets model nor the generalized doors-and-buttons model are strictly more general than each other, and there are cases in which it is difficult to express one in terms of the other. The generalized doors-and-buttons model can chain many activators and effectors together, yielding what would look like a gadget with potentially linear number of traversals. In addition, gadgets do not have internal vertices in the way that activators and effectors can sit on vertices. However, the gadgets can have complex internal states, whereas the total state in the generalized doors-and-buttons model is shown in the ability to cross the edges with effectors. Further, gadgets can have transitions which change the state of the gadget also change their own traversability, whereas in the doors-and-buttons model activators and effectors are normally forced to be on different locations (though this need not generally be the case). We believe that some of the power of these two models is from there simplicity.

We now define an even more general model which encompasses both the generalized doors-and-buttons and the gadgets models. We do not believe this is an ideal model to work in, but we will apply restrictions to this model until we reach both the doors-and-buttons and the gadgets model. Along the way, we wo;; offer ideas of why it may be prudent to pick one of those models over the other, or what additional restrictions in those models may be useful to consider.

In the ***fully generalized doors-and-buttons*** model we are given a graph labeled with ***controllers*** and the ***traversability*** of edges and vertices. Vertices can be passable or impassable, and edges can be passable in one, both or neither direction. Controllers are finite state machines which additionally send an output signal and dictate the traversability of the edge/vertex they live on. When an agent enters or leaves a vertex with a controller, or crosses an edge with a controller, that agent has a set of inputs from which they must select one to send to the controller. That input then causes the controller to potentially change state, change the traversability of the edge or vertex it is on, and send inputs to other controllers causing them to also update and perform actions. A gadget can be constructed

by controllers which track the state of the gadget and send signals to each other with their name whenever they are crossed. Doors and buttons are each very simple types of controllers. However, we think there is a lot of utility in the restrictions imposed by the gadgets and the doors-and-buttons models, and will describe some of those here.

One very important simplification is that controllers only send signals in response to states reached from the agent's input. This means we do not need to worry about timing between agent and controller steps. Also, determining if such a system of message passing finite automaton ever stop can be computationally intractable in itself. This forces our environment to be reactive and temporally bounded in some sense.

The generalized doors-and-buttons model goes further and splits up controllers into ***effectors*** which dictate the traversability of the vertex or edge they live on, and ***activators*** which take in the agent's input and send signals to effectors. We further see the restriction that each vertex or edge can have at most one activator or effector on it at a time. We could imagine investigating analogous restrictions in the Gadgets model in which some transitions never change the traversability between their locations and other transitions never cause a state change in the gadget. In particular, the Door gadget from [16] and studied further in [5] obeys this property. The traversal tunnel never changes the gadgets state, and the open and close tunnels never change how they can be traversed.

Further, in doors-and-buttons activators are stateless and effectors have no state besides the traversability of their location. If we ignore switches, which were not in the original doors-and-buttons papers, then the actions of effectors do not need to consider their state. A potentially interesting and analogous restriction in the gadgets model would be for them to have ***embodied state*** where no two states of a gadget can have the same set of traversals. Thus there are no states that "look the same" but are distinguishable. Most of the gadgets we examine do have this property.

The gadget model has a notion of ***gadget locality*** where it only exists on some small, fixed number of edges. The doors-and-buttons model also investigates a similar notion by only allowing the doors to be controlled by a constant number of buttons, and the buttons

to control a constant number of doors. We can apply the same constraint to our general activators and effectors. Further, we could also demand that the entire connected component of activators and effectors remain below a certain size.

The gadget model has a notion of spatial locality when looking at the planar case. Although the doors-and-buttons model was defined on grid graphs, the ability of the doors and buttons to be spatially disparate allowed long distance interactions in the world. For the generalized doors and buttons model we suggest three possible restrictions: full planarity, k-hop locality, and face locality. In ***full planarity*** it would be required that if edges are added to the location graph for all of the connections between controllers, the resulting graph is still planar. This keeps pathways from having to cross the connections between gadgets, but also forces a sparsity on the connection graph among controllers. ***Face locality*** requires the location graph to be planar and requires that controllers which are connected share a face with each other. Finally, ***k-hop locality*** does not care about planairty but instead wants to keep pieces of gadgets near to each other. In this case, all controllers must be within a distance k in the location graph of the controllers to which they are connected with.

1.4 Related Problems for Reductions

In this section we provide full definitions for problems related to Constraint Logic and quantified boolean formulas which are used in reductions throughout this thesis.

1.4.1 Constraint Logic

Constraint Logic [27, 44] is a uniform family of games — one-player, two-player, or team, with both bounded and unbounded variants — with the appropriate complexity in each case (as in Table 1.1). We will only describe the unbounded variants of Constraint Logic, as we use formula games for our bounded reductions. We also do not describe zero-player Constraint Logic, as we do not need it here.

In general, a ***constraint graph*** is an undirected maximum-degree-3 graph, where each

edge has a weight of 1 (called a red edge) or 2 (called a blue edge). A ***legal configuration*** of a constraint graph is an orientation of the edges such that, at every vertex, the total incoming weight is at least 2. A ***legal move*** in a legal configuration of a constraint graph is a reversal of a single edge that results in another legal configuration.

In ***1-player*** Constraint Logic (also called ***Nondeterministic Constraint Logic or NCL***), we are given a legal configuration of a constraint graph and a target edge e, and we want to know whether there is a sequence of legal moves ending with the reversal of target edge e. In this game, two types of vertices suffice for PSPACE-completeness: an OR vertex has exactly three incident blue edges, and an AND vertex has exactly one incident blue edge and exactly two incident red edges. We can also assume that each OR vertex can be assigned two "input" edges, and the overall construction is designed to guarantee that at most one input edge is incoming at any time; thus, we only need a "Protected OR" gadget which does not handle the case of two incoming inputs. Furthermore, the problem remains PSPACE-complete for planar constraint graphs.

In ***2-player Constraint Logic (2CL)***, each edge of a constraint graph is also colored either black or white, and two players named Universal and Existential[4] alternate making valid moves where each player can only reverse an edge of their color. Given a legal configuration of a constraint graph, a target white edge for the Existential player, and a target black edge for Universal, the goal is to determine whether the Existential player has a forced win, i.e., a strategy for reversing their target edge before any Universal player can reverse their target edge. In this game, six types of vertices suffice for EXPTIME-completeness: AND and OR vertices where all edges are white, AND vertices where all edges are black, AND vertices where the blue edge is white and one or both of the red edges are black, and degree-2 vertices where exactly one edge is black.

In ***Team Private Constraint Logic (TPCL)***, there are two players on the Existential team and one player on the Universal team, who play in round-robin fashion. In each move, the player can reverse up to a constant number k of edges of their color. Each player has

[4]These were initially the Black and White players to reflect the edge coloring, but we adopt the names Universal and Existential to reflect what role these players take in the formula alternation.

a target edge to reverse, and can see the orientation of a specified set of edges, including edges of their own color and edges incident to those edges. Given a legal configuration of a constraint graph, the goal is to determine whether the Existential team has a forced win; i.e., whether one of the Existential players can reverse their target edge before Universal can. In this game, all possible black/white colorings of AND and OR vertices suffice for RE-completeness. (Only undecidability has been claimed before, but RE-completeness follows by the same arguments.)

1.4.2 Formula Games

A ***3-CNF formula*** is a boolean formula φ of the form $C_1 \wedge \cdots \wedge C_k$, where each ***clause*** C_i is the disjunction of up to three ***literals***, which are variables or their negations. An ***assignment*** for such a formula specifies a truth value for each variable, and is ***satisfying*** if the formula is true under the assignment.

In ***3SAT***, we are given a 3-CNF formula, and we want to know whether it has a satisfying assignment. 3SAT is NP-complete [40].

A ***partially quantified boolean formula*** is a formula of the form $Q_1 x_1 : \cdots : Q_n x_n : \varphi$, where Q_i is one of the quantifiers $\forall$ or $\exists$, x_i is a (distinct) variable, and φ is a 3-CNF formula. An ***assignment*** for a partially quantified boolean formula specifies a truth value for each variable in φ that is not any x_i, called ***free*** variables. For a partially quantified boolean formula $\psi = Q_1 x_1 : \cdots : Q_n x_n : \varphi$ with $n > 0$, let $\psi' = Q_2 x_2 : \cdots : Q_n x_n : \varphi$. Given an assignment S for ψ, define assignments $S + x_1$ and $S + \neg x_1$ for ψ' which assign the same truth value as S to each free variable of φ and assign "true" and "false" to x_1, respectively. The truth value of ψ under S is defined recursively as follows:

- If $n = 0$ (so $\psi = \varphi$), ψ is true under S if and only if φ is true under S.
- If $n > 0$ and $Q_1 = \forall$, ψ is true under S if and only if ψ' is true under both $S + x_1$ and $S + \neg x_1$.
- If $n > 0$ and $Q_1 = \exists$, ψ is true under S if and only if ψ' is true under at least one of

$S + x_1$ and $S + \neg x_1$.

A ***quantified boolean formula*** is a partially quantified boolean formula with no free variables. A quantified boolean formula has only one assignment (which is empty), so we say it is true if it is true under this unique assignment.

The truth value of a quantified boolean formula $\psi = Q_1 x_1 : \cdots : Q_n x_n : \varphi$ is equivalent to whether the $\exists$ player has a forced win in the following game: two players $\exists$ and $\forall$ choose an assignment for φ by assigning variables in the order they are quantified, with player Q_i choosing the truth value of x_i. $\exists$ wins if the assignment satisfies φ.

In ***QBF***, we are given a (fully) quantified boolean formula, and we want to know whether it is true. QBF is PSPACE-complete, even if we restrict to formulas with alternating quantifiers beginning with $\exists$. This restriction is equivalent to that $\exists$ and $\forall$ take alternating turns, with $\exists$ going first [40].

A ***dependency quantified boolean formula*** is a formula of the form $\forall x_1 : \cdots : \forall x_m : \exists y_1(s_1) : \cdots : \exists y_n(s_n) : \varphi$, where x_i and y_j are (distinct) variables, φ is a 3-CNF formula, and s_j is a subset of $\{x_i \mid i \leq m\}$. We also require that every variable in φ is some x_i or y_j (φ has no free variables). A ***strategy*** for a dependency quantified boolean formula is a collection of functions $f_j : \{\text{true}, \text{false}\}^{s_j} \to \{\text{true}, \text{false}\}$ for $j = 1, \ldots, n$. A strategy ***solves*** a dependency quantified boolean formula if for every map $S : \{x_i \mid i \leq m\} \to \{\text{true}, \text{false}\}$, the assignment given by $x_i \mapsto S(x_i)$ and $y_j \mapsto f_j(S|_{s_j})$ satisfies φ. Intuitively, y_j is only allowed to depend on the variables in s_j. A quantified boolean formula is a special case of a dependency quantified boolean formula, where each $s_j = \{x_i \mid i < k\}$ for some k. A dependency quantified boolean formula is ***true*** if there is a strategy that solves it.

The truth value of a dependency quantified boolean formula $\forall x_1 : \cdots : \forall x_m : \exists y_1(s_1) : \cdots : \exists y_n(s_n) : \varphi$ is equivalent to whether the "$\exists$" team has a forced win in the following game, which puts a team of one player $\forall$ against a team of players $\exists_j$ for $j = 1, \ldots, n$: $\forall$ picks a truth value for each x_i. $\exists_j$ sees the truth value for each element of s_j (and nothing else) and picks a truth value for y_j. The $\exists$ team wins if the resulting assignment satisfies φ.

In the ***DQBF*** problem, we are given a dependency quantified boolean formula, and we

want to know whether it is true. DQBF is NEXPTIME-complete even if we restrict to formulas of the form $\forall\vec{x}_1 : \forall\vec{x}_2 : \exists\vec{y}_1(\vec{x}_1) : \exists\vec{y}_2(\vec{x}_2) : \varphi$, where $\vec{x}_i$ and $\vec{y}_i$ may contain multiple variables, and each variable in $\vec{y}_i$ can depend on all the variables in $\vec{x}_i$. This restriction is equivalent to requiring that the $\exists$ team has two players who each choose multiple variables, and they see disjoint exhaustive subsets of the variables the $\forall$ player picks [52].

Chapter 2

Single Player

The single player model, where a single agent is navigating a semi-static environment in order to reach a goal, is the primary focus of our study. We give a through study of several different classes of gadgets, often considering the planar case and often classifying the complexity of all gadgets in that class. Reversible, deterministic gadgets were inspired by the success of constraint logic, another reversible system, and by the undoability of Push-Pull block puzzles. Door gadgets came from a desire to simplify prior proofs which used the door framework by providing proofs of planar motion planning with those gadgets, as well as exploring related gadget types which may be even easier to use. Input-output gadgets were inspired by switching graphs and train based games and puzzles, although their main consideration is in the 0-player model in Chapter 4. Finally LDAG gadgets are the naturally bounded class of gadgets and thus one of the most general cases we could hope to study for insight into NP-complete 1-player motion planning problems.

In Section 2.1 we give containment in PSPACE and show a class of gadgets which is in NL. These or similar proofs will be used im multiple later sections. In Section 2.2 we consider k-tunnel reversible deterministic gadgets and give a dichotomy classifying them as either PSPACE-complete or in NL. This work comes primarily from [29] and [25] written in collaboration with Erik Demaine, Isaac Grossof, Dylan Hendrickson, and Mikhail Rudoy. We also examine the reconfiguration problem for reversible deterministic gadgets, showing

that hardness for reachability implies hardness for reconfiguration for this class and exhibiting a reversible deterministic gadget for which the reachability problem is in NL but the reconfiguration problem is PSPACE-complete.

In Section 2.3 we examine polynomially bounded gadgets. Sections 2.3.3 to 2.3.5 give both algorithms and hardness results for various classes of LDAGs and concludes with a dichotomy theorem for DAG gadgets. Section 2.3.7 then extends some of those results to give a dichotomy for deterministic eventually static gadgets. Section 2.3.6 examines bounded gadgets under shortest-path alternate victory condition. Section 2.3.8 shows a specific and commonly used gadget, the crossing NAND, is NP-complete in the planar case. Section 2.3.6 shows the shortest-path victory condition collapses LDAGs to have the same characterization as DAG gadgets.

Section 2.4 examines reconfiguration, giving a more general class of gadgets which is in NP, exhibiting both gadget for which reconfiguration is easier than reachability and another gadget for which reachability is easier than reconfiguration. Finally, it shows PSPACE-completeness for gadgets whose traversability only increases or only decreases, showing that having a bounded number of changes in traversability does not suffice for a gadget to be in NP. Results in Sections 2.3 and 2.4 primarily comes from [29] and [8] written in collaboration with Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, and Dylan Hendrickson.

In Section 2.5 we consider 2-state input/output gadgets. These gadgets have locations which act either only as entrances or only as exits. In Section 2.5.2 we show NL-hardness and containment in NP for single-input gadgets. In Section 2.5.3 we show NP-hardness for a class of single-input gadgets and bounded gadgets. For unbounded multi-input gadgets PSPACE-completeness follows from 0-player work in Section 4.3. This work comes from [9], done in collaboration with Joshua Ani, Erik Demaine, and Dylan Hendrickson.

In Section 2.7 we perform a through study of door gadgets. We show all but one of the planar cases of door gadgets to be PSPACE-complete in Section 2.9.3. We also introduce a similar gadget, the self-closing door, in Section 2.8 and show all of its variations are hard in the planar case in Section 2.9.2. This work primarily comes from [7], written in

collaboration with Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, and Dylan Hendrickson.

2.1 General Upper Bounds

We give two upper bounds that will be used repeatedly throughout this chapter. The first is the observation that 1-player motion planning problems are in PSPACE. The second identifies a property of gadgets, roughly that going through one tunnel does not change any other tunnels, which puts the motion planning problem in NL. This, or slight variations on this property, show up in several dichotomy theorems and is the main defining feature we have seen for computationally tractable gadgets. This section comes from [29] and [25] written in collaboration with Erik Demaine, Isaac Grossof, Dylan Hendrickson, and Mikhail Rudoy.

Lemma 1. *1-player motion planning with any set of gadgets is in* PSPACE*.*

Proof. This was shown in [25], but included here for convenience. A configuration of the system of gadgets consists of the state of each gadget and the location of the robot, and has polynomial length. The algorithm that repeatedly nondeterministically picks a legal transition, and updates the configuration based on it, accepting when the robot reaches the goal location, decides the reachability problem in nondeterministic polynomial space. By Savitch's theorem, the problem is in PSPACE. □

Theorem 2. *1-player motion planning with any k-tunnel gadget that does not have interacting tunnels is in* NL*.*

Proof. We first show that, if a system of such gadgets has a solution, then it has a solution which visits each location at most once. Suppose there is a solution, and consider the last time a solution of minimal length visits a previously visited location, assuming there is any such time. Let v be the vertex of this last self-intersection. After leaving v for the last time, every transition the robot makes is through a tunnel that it had not previously traversed. Since the

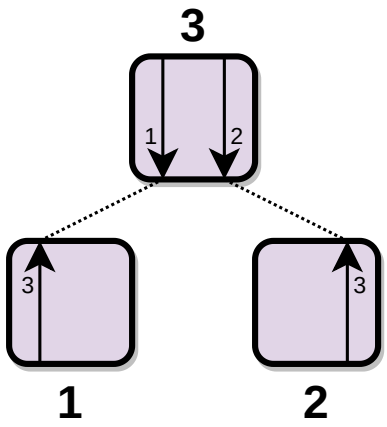
gadget does not have interacting tunnels, these tunnels have the same traversability when the robot goes through them as they do originally. We modify the solution by "shortcutting": remove the portion of the solution between the first visit to v and the last visit to v, so the robot only visits v once, and skips the loop that begins and ends at v. The new path is still a solution: the segment before v is identical to the unmodified solution, and the segment after v consists of tunnels whose traversability is not changed before the robot goes through them. The shortcut path is shorter than the original solution, which was assumed to be minimal. Thus a solution of minimal length has no self-intersections.

We will want to treat the system of gadgets as though it were a directed graph by replacing each tunnel with an edge in the appropriate direction, or a pair edges if it is traversable in either direction. We can locally walk through all the available transitions in a gadget, assess which locations they lead to, and non-deterministically pick one to try, allowing this to be executed in NL. A path from the start location to the end location in this graph is exactly a solution for the system of gadgets with no self-intersections; the traversability of each tunnel used in such a solution does not change before the tunnel is used.

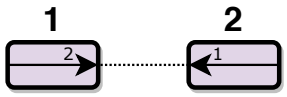
Since reachability in directed graphs is in NL, the motion planning problem is also in NL. Moreover, if the gadget has any state in which a tunnel can be traversed in one direction but not the other, the motion planning problem is NL-complete, and otherwise it is in L. □

2.2 1-Player Reversible Deterministic Gadgets

In this section, we study reversible deterministic k-tunnel gadgets giving a complete categorization as either in NL or PSPACE-complete for reversible, deterministic gadget. Recall a gadget if ***reversible*** if every transition can be undone after being taken, and a ***deterministic*** gadget is one who's transition graph has maximum out-degree of 1. For upper bounds, Theorem 2 showed that 1-player motion planning problems with non-interacting-k-tunnel gadgets is in NL and Theorem 1 shows containment in PSPACE. For the PSPACE-hardness half of the characterization, we introduce a new base gadget, the ***locking 2-toggle (L2T)*** shown in Figure 2-1a. In Section 2.2.2 we show that all interacting-k-tunnel reversible



(a) Locking 2-toggle transition diagram



(b) 1-toggle transition diagram

Figure 2-1: State diagrams for the Locking 2-Toggle and the Non-deterministic Locking 2-Toggle.

deterministic gadgets are able to simulate the locking 2-toggle. Then in Section 2.2.3 we show that 1-player motion planning with locking 2-toggles is PSPACE-complete by simulating Nondeterministic Constraint Logic. Section 2.2.3 shows how to adapt the construction to show these gadgets remain PSPACE-hard even for the planar 1-player motion planning problem. This work comes primarily from [29], [25], and [8] written in collaboration with Joshua Ani, Isaac Grossof, Dylan Hendrickson, Erik Demaine, Yevhenii Diomidov, and Mikhail Rudoy.

2.2.1 Closure Properties

Lemma 3. *Any system of gadgets composed of two reversible gadgets is reversible.*

Proof. Consider any transition through the system formed by composing two reversible gadgets. This transitions is a walk through the gadgets and connections that form a system. Since both gadgets are reversible, it is possible for the robot to enact the exact reverse of this walk after the walk is done. This will exactly reverse the effect of the walk within each gadget. Thus, it is possible to reverse the entire transition.

Since every transition of the system can be reversed, the system is reversible. □

Since all of the gadgets we consider in this thesis are reversible, Lemma 3 means our

systems will all be reversible as well.

Lemma 4. *Any branchless system of gadgets composed of deterministic reversible gadgets is deterministic and reversible.*

Proof. The state space of a reversible, deterministic gadget is an undirected matching of some (state, location) pairs to each other. This a necessary and sufficient characterization of reversible, deterministic gadgets.

When we compose two such gadgets, we create paths through the pair of gadgets. However, no (state, location) pair has more than two edges: One connection to the other gadget, and one edge through its original gadget. Moreover, any (state, location) pair that forms an external location has a most one edge, as it does not connect to the other gadget. As a consequence, the path from any external location through the gadget is either a deterministic path to another external location, or a dead end. There is no branching, as branching would require a location with three edges.

Thus, the resultant object is deterministic. By Lemma 3 it is reversible as well. □

2.2.2 Reducing to Locking 2-Toggles

In this section, we introduce the locking 2-toggle shown in Figure 2-1a, and we show that all interacting-k-tunnel reversible deterministic gadgets can simulate it. The proof first examines what constraints on a gadget are implied by being interacting-k-tunnel, reversible, and deterministic, and goes on to identify that all such gadgets have a pair of special states with some useful common properties. From this pair of states we construct a 1-toggle, and then combine that with our special states to build a locking 2-toggle. One of the major insights is identifying this special pair of states which belongs to all gadgets in the class, and after that the primary challenge is in preventing undesired transitions, which are plentiful when allowing such a wide class of gadgets.

Theorem 5. *Every interacting-k-tunnel reversible deterministic gadget simulates a locking 2-toggle.*

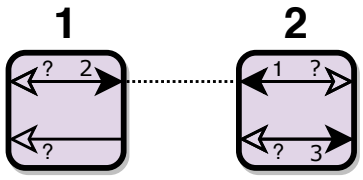


Figure 2-2: An arbitrary interacting-k-tunnel reversible deterministic gadget. Hollow arrows indicate traversals that may or may not be possible. Solid or absent arrows indicate traversals that are or are not possible, respectively.

Proof. We begin by examining an arbitrary interacting-k-tunnel reversible deterministic gadget, as shown in Figure 2-2. Because the gadget has interacting tunnels, we can find a pair of states in which traversing the top line can change the traversability of the bottom line to the right. Since it is also reversible, the inverse transition is also possible, so traversing the top line can change in either direction the left-to-right traversability of the bottom line. Then without loss of generality, the gadget has the form shown in Figure 2-2: in state 1, traversing the top line to the right switches to state 2, and the bottom line is not traversable to the right. In state 2, traversing the top line to the left switches to state 1, and the bottom line is traversable to the right, say to state 3 (which may be the same as state 1). All other traversals may or may not be possible in either state, indicated by the question marks.

Lemma 6. *Every interacting-k-tunnel reversible deterministic gadget simulates a* ***one-directional edge****, that is, a tunnel which (in some state) can be traversed in one direction but not the other.*

Proof. If in some state, some edge in the gadget can be traversed in one direction but not the other, then it is a one-directional edge. Otherwise, the gadget has the form shown in Figure 2-3a. Then the construction in Figure 2-3b is equivalent to a one-directional edge: currently the gadget is in state 1, so the path from the bottom to the top is blocked by the bottom edge, but from the top, you can go across the top edge, switching the gadget to state 2, and then back across the bottom edge. □

Lemma 7. *Every interacting-k-tunnel reversible deterministic gadget simulates a 1-toggle (Figure 2-1b).*

(a) State graph, refining Figure 2-2.

(b) Simulating a one-directional edge.

Figure 2-3: An arbitrary interacting-k-tunnel reversible deterministic gadget which has no one-directional edge.

(a) Form of state graph.

(b) Simulating a 1-toggle.

Figure 2-4: A one-directional edge gadget.

Proof. By the previous lemma, we can build a one-directional edge, which has the structure shown in Figure 2-4a: in state 1, we can traverse the edge to the right and switch to state 2, but not to the left. In state 2, we can undo this transition, and possibly also traverse the edge to the right. The construction in Figure 2-4b is then a 1-toggle. In the current state, it can be traversed to the right but not to the left because of the gadget on the left. After making this traversal, it becomes the rotation of the current state, and it cannot be traversed to the right again because of the gadget on the right. □

To build a locking 2-toggle, we put the arbitrary gadget (in state 2), an antiparallel pair of 1-toggles, and the rotation of the arbitrary gadget (also in state 2) in series, as in Figure 2-5. Currently, the top edge is traversable to the left and the bottom edge is traversable to the right, but not in the other direction. After traversing the top edge to the left, the 1-toggles prevents us from traversing either edge to the left, and the leftmost gadget (in state 1) prevents us from traversing the bottom edge to the right, so the only legal traversal is going back across the top edge to the right. Similarly after traversing the bottom edge, the only legal traversal is across the bottom edge in the opposite direction. Thus this construction is equivalent to a (antiparallel) locking 2-toggle.

Traversing the simulated locking 2-toggle takes either 4 or 6 transitions of the raw gadget,

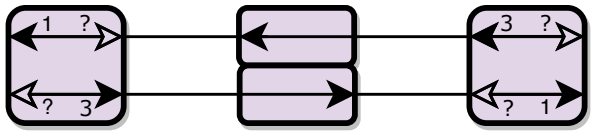
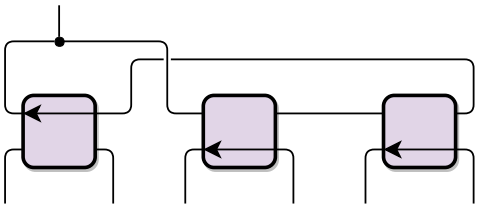
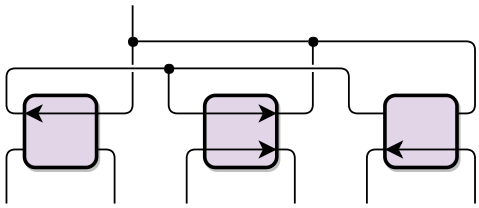


Figure 2-5: An arbitrary interacting-k-tunnel reversible deterministic gadget and a 1-toggle simulate a locking 2-toggle.



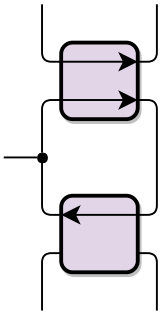
(a) An AND vertex gadget. The leftmost edge has weight two and is pointing in (up). The other edges have weight one and are pointing away (down).



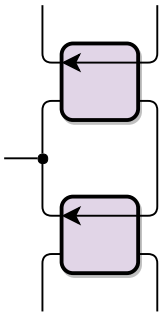
(b) An OR vertex gadget. All edges are weight 2. The leftmost edge is pointing in (up), the middle edge is free, and the rightmost edge is pointing away (down).

Figure 2-6: Vertex gadgets in the NCL reduction.

depending on whether it contains a one-directional edge (from Lemma 6). For simplicity, we can include additional gadgets (e.g. another pair of 1-toggles) to ensure it always takes exactly 6 transitions; this will be relevant to timing considerations in multiplayer games. □



(a) An edge gadget pointed up, in the unlocked state. The gadget is accessed by the loose end on the left.



(b) The same edge gadget in the locked state.

Figure 2-7: Edge gadget in the NCL reduction.

2.2.3 PSPACE-hardness

In this section, we show that 1-player motion planning with the locking 2-toggle is PSPACE-complete by a reduction from Nondeterministic Constraint Logic (NCL). See Section 1.4.1 for a definition of NCL. We represent edges by pairs of locking 2-toggles. The construction requires ***edge gadgets*** which are directed and can be flipped, as well as AND and OR ***vertex gadgets*** which apply constraints on how many edges must be directed towards them at any given point in time.

Theorem 8. *1-player motion planning with the locking 2-toggle is PSPACE-complete.*

Proof. Motion planning with the gadget is in PSPACE by Lemma 1. We use a reduction from Nondeterministic Constraint Logic (NCL) to show PSPACE-hardness.

The ***edge gadget***, shown in Figure 2-7, contains two locking 2-toggles, each of which is also attached to a vertex gadget. It is oriented towards one of the vertices, can be either ***locked*** or ***unlocked***. Specifically, the edge gadget is unlocked (Figure 2-7a) if either locking 2-toggle is in the middle state (with both lines traversable), and locked (Figure 2-7b) otherwise. It is oriented towards the vertex attached to the locking 2-toggle whose edge not accessible from the edge gadget is traversable. The robot can access the free line on the left. If the edge gadget is unlocked, the robot can traverse a loop through one edge of each locking 2-toggle to change the orientation of the edge gadget. The edge gadget switches between being locked and unlocked when the robot moves through a vertex gadget to traverse one of the edges not accessible from the edge gadget.

The **vertex gadgets** are shown in Figure 2-6. The robot can access the free line on the top, and traverse loops to lock and unlock edge gadgets, enforcing the constraints of vertices. Specifically, if all three edges are pointing towards an AND vertex, the robot can traverse a loop to lock both weight-1 edges and unlock the weight-2 edge, or vice versa. If multiple edges are pointing towards an OR vertex, the robot can traverse a loop to unlock the currently locked edge and lock another edge. Observe that for both vertex gadgets, the sum of the weights of locked edges does not change.

Given an NCL graph, we construct a system of locking 2-toggles. Each edge in the graph corresponds to an edge gadget (Figure 2-7). Each locking 2-toggle in the edge gadget corresponds to a vertex incident to the edge. When three edges meet at a vertex, we put a vertex gadget on the locking 2-toggles corresponding to that vertex. We use an AND vertex gadget (Figure 2-6a) or an OR vertex gadget (Figure 2-6b) depending on the type of vertex. The vertical "entrance" line on each vertex gadget and horizontal "entrance" line on each edge gadget is connected to the starting location. Each edge is oriented as in the NCL graph. For each vertex, we pick a set of edges initially pointing at the vertex with total weight 2. The edge gadgets corresponding to the chosen edges are locked, and other edge gadgets are unlocked. The goal location is placed inside the edge gadget corresponding to the target edge so that it is reachable if and only if the target edge is unlocked.

If the original NCL graph is solvable, the robot can perform the same sequence of edge flips, visiting vertex gadgets to lock and unlock edges as necessary, and reach the goal location. If the robot can reach the goal location, the same sequence of edge flips solves the NCL graph. So the problem is solvable if and only if the NCL graph was. □

This reduction is also possible without edge gadgets, and leads to a system with only one L2T for each constraint logic edge. We use edge gadgets because the reduction is easier to understand, and adaptations of this construction in Sections 2.2.4, 3.2.1, and 3.2.2 will need them.

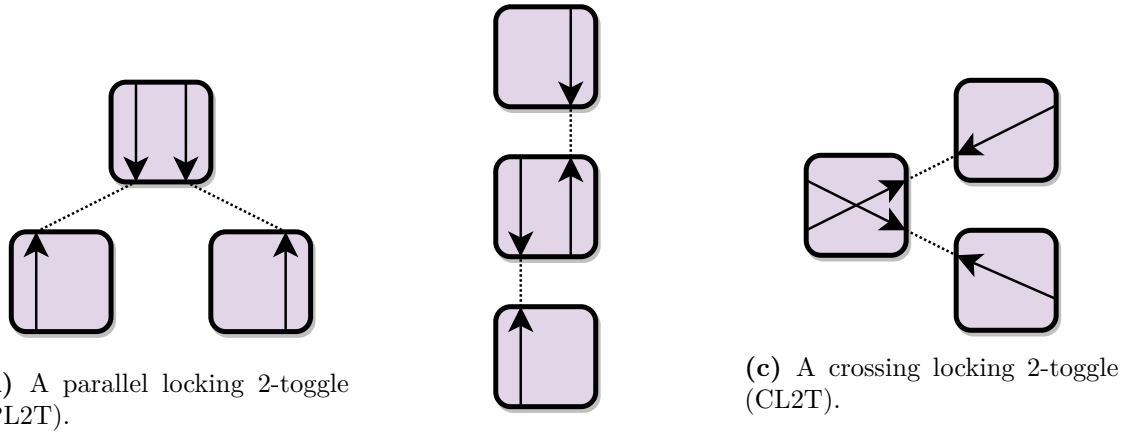
Corollary 9. *1-player motion planning with any interacting-k-tunnel reversible deterministic gadget is* PSPACE*-complete.*

Proof. Hardness follows from Theorems 5, and 8. For any such gadget, we have a reduction from systems of locking 2-toggles to systems of that gadget by replacing each locking 2-toggle with a simulation of one built from the arbitrary gadget. Motion planning with the gadget is in PSPACE by Lemma 1. □

2.2.4 Planarity

In this section, we show that interacting-k-tunnel reversible deterministic gadgets are PSPACE-complete even for the planar 1-player motion planning problem. We once again work with the locking 2-toggle, showing that each of its planar versions can simulate each other. From there we use the crossing locking 2-toggle to build an A / BA crossover, which is less powerful than a full crossover but will suffice to make our reduction in Section 2.2.3 planar. An interesting question is whether the locking 2-toggle is powerful enough to build a full crossover, which can be done with any of the 2 state gadgets. Although not needed here, it would allow the multiplayer game results later in this thesis to carry over to the planar case.

Recall for the planar problem we allow rotations and reflections of gadgets. This leaves three distinct embeddings of the locking 2-toggle into a plane: parallel, antiparallel, and crossing, shown in Figure 2-8, and which we abbreviate PL2T, APL2T, and CL2T. (Up to only rotation, there are four, the other being the antiparallel locking 2-toggle with the other handedness). We will allow reflections of gadgets, so these are the three kinds of locking 2-toggles we will consider.



(a) A parallel locking 2-toggle (PL2T).

(b) An antiparallel locking 2-toggle (APL2T).

(c) A crossing locking 2-toggle (CL2T).

Figure 2-8: Types of locking 2-toggles for planar problems.

Lemma 10 ([25])**.** *Parallel, antiparallel, and crossing locking 2-toggles all simulate each other in planar graphs.*

Proof. Figure 2-9 shows APL2T simulating CL2T, Figure 2-10 shows CL2T simulating PL2T, and Figure 2-11 shows PL2T simulating APL2T. Note that we use both APL2Ts of both handednesses, so we need to be able to reflect gadgets. □

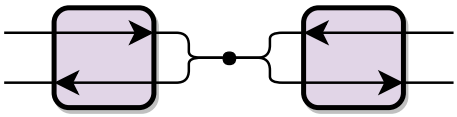


Figure 2-9: APL2T simulating CL2T. (Based on [25, Figure 4].)

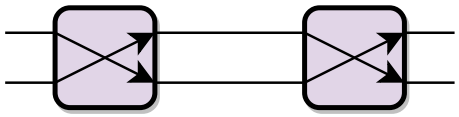


Figure 2-10: CL2T simulating PL2T. (Based on [25, Figure 5].)

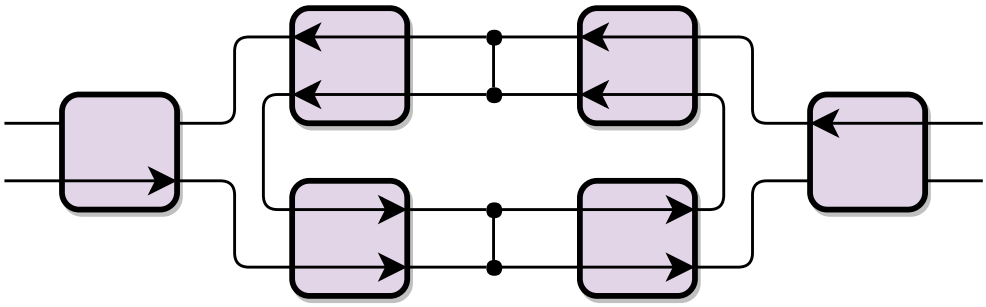


Figure 2-11: PL2T simulating APL2T. (Based on [25, Figure 13].)

Theorem 11. *Every interacting-k-tunnel reversible deterministic gadget simulates each type of locking 2-toggle in planar graphs.*

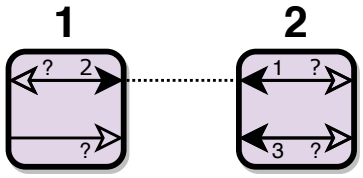


Figure 2-12: The antiparallel case of an arbitrary interacting-k-tunnel reversible deterministic gadget.

Proof. We follow the proof of Theorem 5. As before, we assume that traversing a line to switch from state 1 to state 2 makes a traversal on another line legal. This new traversal can be parallel to, antiparallel to, or cross the first traversal; we consider each case. If the

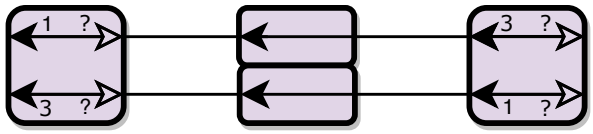


Figure 2-13: An arbitrary antiparallel interacting-k-tunnel reversible deterministic gadget and a 1-toggle simulate a PL2T.

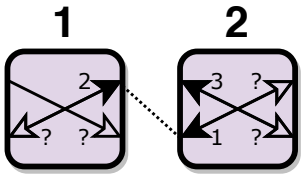


Figure 2-14: The crossing case of an arbitrary interacting-k-tunnel reversible deterministic gadget.

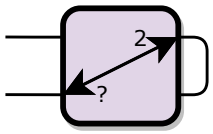


Figure 2-15: A crossing interacting-k-tunnel reversible deterministic gadget simulates a one-way edge.

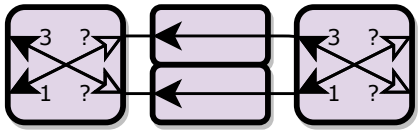


Figure 2-16: An arbitrary crossing interacting-k-tunnel reversible deterministic gadget and a one-toggle simulate a PL2T.

new traversal is parallel, the construction in the proof of Theorem 5 works to simulate an APL2T in a planar graph.

If it is antiparallel, the gadget has the form shown in Figure 2-12. Either the gadget has a one-directional edge, or it has the form in Figure 2-3a, and simulates a one-directional edge by the construction in Figure 2-3b. Thus it simulates a 1-toggle by the construction in Figure 2-4b. Then the construction in Figure 2-13 simulates a PL2T: currently either edge can be traversed to the left, if the top edge is traversed, the left gadget blocks the bottom edge, and if the bottom edge is traversed, the right gadget blocks the top edge.

Finally, if the new traversal crosses the first traversal, the gadget has the form shown in Figure 2-14. Either it has a one-directional edge, or the construction in Figure 2-15

simulates a one-directional edge, similarly to Lemma 6. So the gadget simulates a 1-toggle by the construction in Figure 2-4b. Then the construction in Figure 2-16 simulates a PL2T, similarly to the previous case.

Once the gadget simulates some locking 2-toggle, we can use Lemma 10 to simulate all three types. □

Theorem 12. *1-player planar motion planning with any interacting-k-tunnel reversible deterministic gadget is* PSPACE*-complete.*

Proof. We begin by constructing some weak crossover gadgets. The crossover locking 2-toggle is itself a very weak crossover. We use it to construct an ***A/BA crossover***, shown in Figure 2-17a. Calling the traversal from top to bottom A and that from left to right B, we can perform either of the sequences A and BA. Since everything is reversible and deterministic, we can also undo those sequences. The A/BA crossover is sufficient for the rest of the proof; we abbreviate it as shown in Figure 2-17b.

We modify the proof of Theorem 8, giving a reduction from planar NCL to planar system of gadgets with locking 2-toggles. By Theorem 11, this is sufficient to show PSPACE-hardness. Our gadgets use PL2Ts, CL2Ts, and A/BA crossovers; they do not use APL2Ts.

The edge gadget is shown in Figure 2-18, and vertex gadgets are shown in Figure 2-19. Given a planar NCL graph, we construct a system of gadgets as follows.

Pick a rooted spanning tree of the dual of the NCL graph, directed away from the root; the robot will use this tree to navigate the graph. The system of gadgets will contain a vertex for each face f of the NCL graph, which is a vertex of the spanning tree.

For each edge of the graph, we place an edge gadget. When an edge is in the spanning tree, we orient it so that the A/BA crossover points, from entrance to exit, in the same direction as the edge points in the spanning tree (left to right in Figure 2-18, and away from the root). If an edge is in the spanning tree and has target f, we connect its exit to f. For each edge e, we connect its entrance to the vertex f corresponding to the face containing its entrance, i.e. the face adjacent to e to which we can connect its entrance without crossings. If e is in the spanning tree, this connects the entrance of e to the source f of e.

Now we place a vertex gadget of the appropriate type for each vertex of the NCL graph, so that the gadget shares a PL2T with each incident edge gadget. AND vertex gadgets must be oriented so the weight-2 edge has the appropriate PL2T (the bottom one in Figure 2-19a). The entrance of each vertex gadget is connected to the vertex f corresponding to the face containing the entrance.

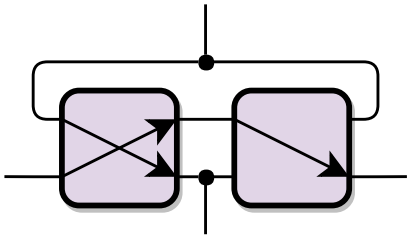
We set each edge gadget to the orientation of its corresponding edge. For each vertex, we select edges directed towards it with total weight 2, and set the selected edges to locked and other edges to unlocked. The goal location is placed inside the target edge so that reaching it requires flipping the target edge. The starting location is the vertex corresponding to the root of the spanning tree.

Play on this system of gadgets proceeds as follows: the robot travels down the spanning tree, crossing edges until it reaches some face. It goes into an edge or vertex attached to that face, and manipulates it. Then the robot travels back up the spanning tree and down a different branch, manipulating another edge or vertex, and so on. The edge and vertex gadgets enforce the NCL constraints. If the target edge can be flipped, the robot can reach the goal location. Thus the system of gadget is solvable if and only if the NCL graph was. The system of gadget is planar by its construction, using the planarity of the NCL graph.

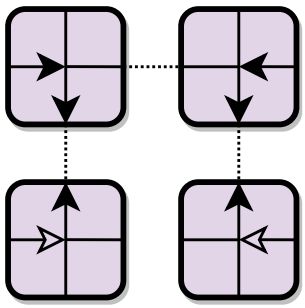
This completes the proof of PSPACE-hardness. Containment in PSPACE is by Lemma 1, so the problem is PSPACE-complete. □

2.2.5 Restricted Starting States and the Nondeterministic Locking 2-toggle

The doors-and-buttons model has been considered when the initial states of the doors are restricted, for example to all start open. One can similarly ask whether the local motion planning problem remains hard if the instance only contains gadgets in some subset of the gadget's states. We have not explored this question in detail because none of our applications have needed this additional structure; however, while investigating a gadget closely related to the locking 2-toggle, we conveniently strengthened our prior results to show that



(a) Simulating an A/BA crossover using CL2Ts.



(b) A state diagram and notation for the A/BA crossover.

Figure 2-17: An A/BA crossover gadget: the robot can traverse top to bottom (A), or traverse left to right (B) and then top to bottom. Thinking of the gadget as a crossing pair of 1-toggles, the vertical 1-toggle is always traversable, and the horizontal 1-toggle is traversable when the vertical one is pointing down.

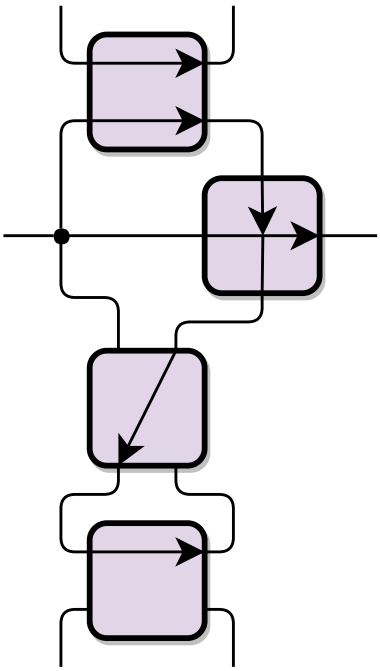
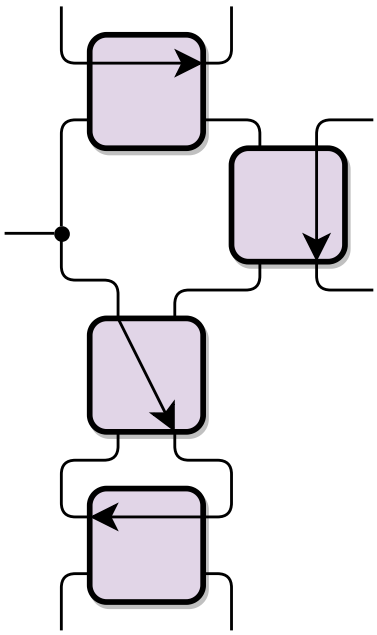
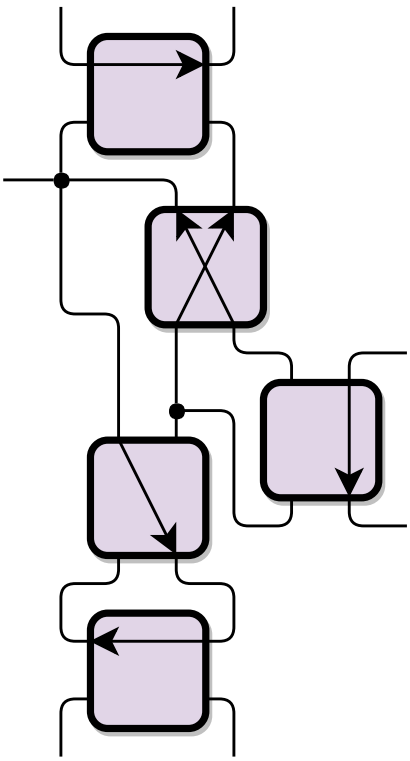


Figure 2-18: An edge gadget for planar graphs, currently unlocked and directed up. This is analogous to Figure 2-7, with two changes. First, the bottom PL2T is "twisted" to have the same handedness as the top PL2T for connecting to vertex gadgets; the CL2T is sufficient for the crossing caused by this. Second, the A/BA crossover allows the robot to cross the edge from left to right, regardless of the state of the edge. We call the line on the left the ***entrance*** and the line on the right, on the other side of the A/BA crossover, the ***exit***.



(a) An AND vertex for planar graphs. Currently the weight-2 edge, connected at the bottom PL2T, is directed towards the vertex and locked, and both weight-1 edges are directed away. If the weight-1 edges become directed towards the vertex, the robot can visit the vertex gadget and traverse a loop through all three PL2Ts, locking the weight-1 edges and unlocking the weight-2 edge. The CL2T is a sufficient crossover.



(b) An OR vertex for planar graphs. Currently the edge containing the bottom PL2T is directed towards the vertex, and the other edges are directed away. If multiple edges are ever directed towards the vertex, the robot can visit the vertex gadget, unlock the locked edge, and lock another edge.

Figure 2-19: NCL vertex gadgets for planar graphs, analogous to the gadgets in Figure 2-6. In each gadget, each of the three PL2Ts is also part of an edge gadget. The robot enters at the line on the left, called the ***entrance***, traverses loops that enforce the NCL constraints, and then leaves at the entrance.

1-player motion planning with the locking 2-toggle remains PSPACE when the gadgets are restricted to start in leaf states. In addition, we prove that 1-player motion planning with the nondeterministic locking 2-toggle is PSPACE-complete.

Work in this section comes from [6] written in collaboration with Joshua Ani, Sualeh Asif, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, Scheffler, Sarah and Adam Suhl.

The ***nondeterministic locking 2-toggle***, shown in Figure 2-21, is a four-state gadget where each state has two transitions, each across the same tunnel. The top pair of states each

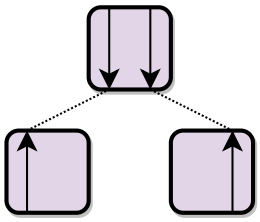


Figure 2-20: State space of the locking 2-toggle.

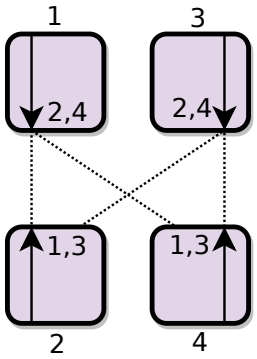


Figure 2-21: State space of the nondeterministic locking 2-toggle.

allow a single traversal downward, and allow the agent to choose either of the two bottom states for the gadget. Similarly, the bottom pair of states each allow a single traversal upward to one of the top states. We can imagine this as being similar to the locking 2-toggle if the tunnel to be taken next is guessed ahead of time: the bottom state of the locking 2-toggle is split into two states which together allow the same traversals, but only if the agent picks the correct one ahead of time.

We use the construction shown in Figure 2-22 to show both that locking 2-toggles starting in leaf states can simulate a locking 2-toggle starting in a nonleaf state, and nondeterministic locking 2-toggles can simulate a locking 2-toggle. This construction consists of two nondeterministic locking 2-toggles and a 1-toggle. A 1-toggle can be trivially simulated by taking a single tunnel of a locking 2-toggle or nondeterministic locking 2-toggle.

Theorem 13. *1-player planar motion planning with the nondeterministic locking 2-toggle is* PSPACE*-complete.*

Proof. In the construction shown in Figure 2-22, the agent can enter through either of the top lines; suppose they enter on the left. Other than backtracking, the agent's only path is across the bottom 1-toggle, then up the leftmost tunnel, having chosen the state of the nondeterministic locking 2-toggle which makes that tunnel traversable.

Now the only place the agent can usefully enter the construction is the leftmost line. The agent can only go down the leftmost tunnel, up the 1-toggle, and out the top right entrance,

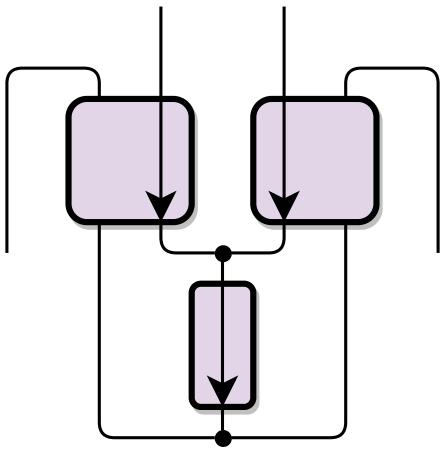


Figure 2-22: Constructing a locking 2-toggle from a nondeterministic locking 2-toggle. It is currently in the unlocked state. The nondeterministic locking 2-toggles are in leaf states (top states in Figure 2-21).

again making the appropriate nondeterministic choice when traversing the left gadget.

Symmetrically, if (from the unlocked state) the agent enters the top right, they must exit the bottom right, and the next traversal must go from the bottom right to the top right and return the construction to the unlocked state. Thus this construction simulates a locking 2-toggle. □

If we instead build the above construction with locking 2-toggles in leaf states, then all three of the locking 2-toggles used are in leaf states (the 1-toggle is one tunnel of a locking 2-toggle). A very similar argument as the nondeterministic locking 2-toggle construction shows this gadget also simulates a locking 2-toggle. Thus, given a 1-player motion planning problem with locking 2-toggles, we can replace all of the locking 2-toggles in nonleaf states with this gadget to obtain an instance where all starting gadgets are in leaf states.

Corollary 14. *1-player motion planning with the locking 2-toggle where all of the locking 2-toggles start in leaf states is* PSPACE*-complete.*

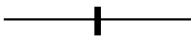
2.2.6 Self-simulation of 2-State Reversible Deterministic Gadgets

In this section we show that all of the 2-state reversible deterministic gadgets can simulate each other with a constant number of gadgets. Although we already know these gadgets

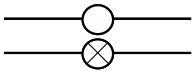
are PSPACE-complete from the prior section, we think the ability to replicate behavior in a simple manner is interesting. In addition, many of the named gadgets in this class seem useful and worth explicitly stating. This section comes from [25] written in collaboration with Erik D. Demaine, Isaac Grossof, and Mikhail Rudoy.

Catagorizing 2-state 2-tunnel reversible deterministic gadgets

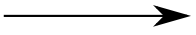
To categorize the possible deterministic reversible 2-state 2-tunnel gadget types, we first categorize the possible tunnel types in such a gadget. A tunnel is ***trivial*** if its traversability does not change with the gadget's state or if traversing it does not change the gadget's state. A trivial tunnel can always be split into a separate 1-state 1-tunnel gadget, so we can ignore them. What remain are three possible ***nontrivial*** tunnel types:



Tripwire A tunnel that can always be traversed in either direction, but traversing it switches the gadget's state.

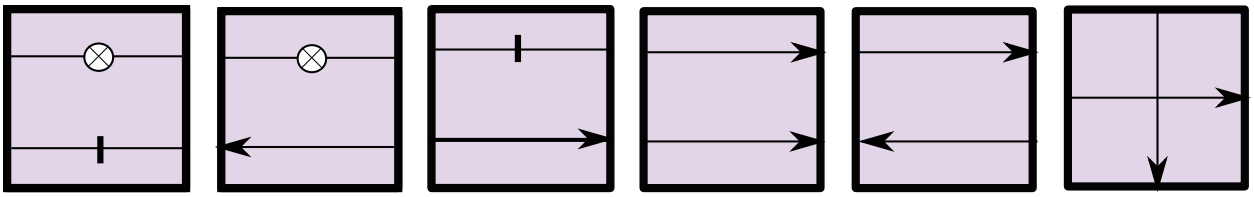


Lock In the ***unlocked*** state (shown above), the tunnel can be traversed in either direction; in the ***locked*** state (shown below), the tunnel cannot be traversed in either direction.



Toggle A tunnel that can always be traversed in a single direction, where the direction differs in the two states of the gadget. The state is switched when the gadget is traversed.

There are six ways to combine these tunnel types into pairs. Two combinations, Lock–Lock and Tripwire–Tripwire, are trivial combinations equivalent to one-state gadgets in which each tunnel is either always traversable in both directions or never traversable. Thus we restrict our attention to the four other combinations, listed below. Because we are interested in planar systems, we consider the multiple planar gadgets for each nontrivial combination. As a result, there are nine different nontrivial two-tunnel two-state gadgets, abbreviated and listed below. The bulk of this Section focuses on the six gadgets shown in Figure 2-23, which



(a) NWL **(b)** NTL **(c)** NWT **(d)** P2T **(e)** AP2T **(f)** C2T

Figure 2-23: Six of the nine deterministic reversible 2-state gadgets on two tunnels. We leave out the CWL, CTL, and CWT gadgets as they are not heavily used in this Section.

omits most crossing variants.

1. **Tripwire–Lock**: Traversing the tripwire makes the other tunnel flip between being passable and impassable, causing it to "lock" or "unlock". There are crossing and non-crossing varieties, abbreviated **CWL** (crossing wire lock) and **NWL** (non-crossing wire lock).

2. **Toggle–Lock**: Traversing the toggle flips the lock tunnel between being passable and impassable. Crossing the lock tunnel, by definition, does not change the state of the gadget. Notice that one direction of the toggle corresponds to an open lock and the other direction to the closed lock. There are crossing and non-crossing varieties, abbreviated **CTL** (crossing toggle lock) and **NTL** (non-crossing toggle lock).

3. **Tripwire–Toggle**: Here traversing either the tripwire or the toggle flips the direction of the toggle. There are crossing and non-crossing varieties, abbreviated **CWT** (crossing wire toggle) and **NWT** (non-crossing wire toggle).

4. **Toggle–Toggle**: Also known as a **2-toggle** [24]. Traversing either toggle flips the direction of both of them. This is the only case where there are two directed tunnels, leading to three possibilities: crossing, parallel, and anti-parallel. They are abbreviated **C2T** (crossing 2-toggle), **P2T** (parallel 2-toggle), and **AP2T** (anti-parallel 2-toggle).

Self-simulation constructions

To show all 2-state 2-tunnel reversible deterministic gadgets simulate each other, roughly we will show that the 2-toggle simulates every other k-tunnel 2-state reversible deterministic gadget, and then show that each of those in turn simulate the 2-toggle.

Theorem 15. *The 2-toggles, toggle-locks, tripwire-locks and tripwire-toggles, in all orientations, can each simulate each other.*

Proof. We will show:

- AP2Ts can simulate P2Ts, C2Ts, NTLs, NWTs, and NWLs and crossovers
- P2Ts, C2Ts, NTLs, NWLs and NWTs can each simulate AP2Ts,
- CTLs can simulate NTLs, CWLs can simulate NWLs, and CWTs can simulate NWTs.

Thus, every gadget can simulate AP2Ts, and AP2Ts can simulate every non-crossing gadget, as well as crossovers. By combining non-crossing gadgets with crossovers, AP2Ts can simulate every gadget.

This gives a simulation of every gadget by every other gadget, via AP2Ts as an intermediate step. □

Lemma 16. *Antiparallel 2-toggles (AP2Ts) simulate a crossing 2-toggle (C2T).*

Proof. The construction is given in Figure 2-24. In the state of the construction shown in the figure, there are two possible transitions: the robot can move from the upper left to the bottom right of the construction, or from the upper right to the bottom left. Either of those transitions toggles both AP2Ts, leaving the construction mirrored top to bottom. Thus, the construction has two states. The possible traversals in one state (as shown above) are from the top left to the bottom right and from the top right to the bottom left, while the possible traversals in the other state are (by symmetry) from the bottom left to the top right and from the bottom right to the top left. Following any of these traversals swaps the state of the construction. Notice that this is exactly the behavior of a C2T.

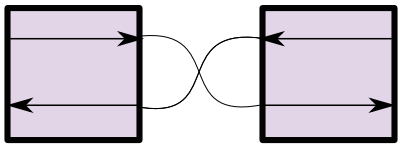


Figure 2-24: Anti-parallel 2-toggles simulate a crossing 2-toggle

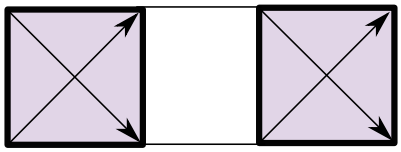


Figure 2-25: Crossing 2-toggles simulate a parallel 2-toggle

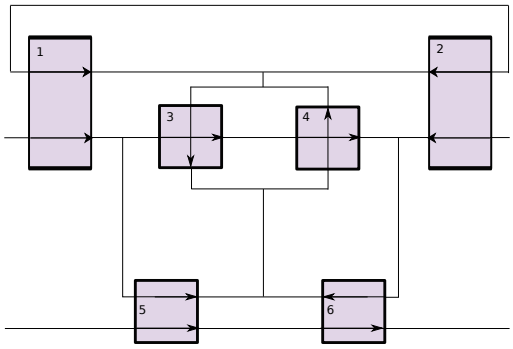


Figure 2-26: 2-toggles simulate 1-toggle-lock.

If the robot enters the construction shown from the upper left, upon reaching the center the robot can only proceed to the bottom right, or come back the way it came. Therefore, the upper left to bottom right transition is the only possible transition from that location. By symmetry, the same is true from top left to bottom right. Thus, the one traversal described for each location in each state is the only one possible. □

Lemma 17. *Crossing 2-toggles (C2Ts) simulate a parallel 2-toggle (P2T).*

Proof. The construction is given in Figure 2-25. In the state of the construction shown in the figure, there are two possible transitions: the robot can move from the top left to the top right of the construction, or from the bottom left to the bottom right. Either of these transitions toggles both C2Ts, leaving the construction mirrored left to right. The allowed traversals in one state (as shown above) are from the top left to the top right and from the bottom left to the bottom right, while the allowed traversals in the other state are (by symmetry) from the top right to the top left and from the bottom right to the bottom left. Following any of these traversals swaps the state of the construction. Notice that this is exactly the behavior of a P2T.

Since the system is composed entirely of C2Ts (without even branching hallways), which are both reversible and deterministic, the result is also both reversible and deterministic, by Lemma 4. Thus, the one transition described for each location in each state is the only transition possible. □

Lemma 18. *2-toggles (AP2Ts, P2Ts and C2Ts) simulate a noncrossing toggle lock (NTL).*

Proof. The construction is shown in Figure 2-26.

In this lemma, we will refer to toggles 1 and 2 in the figure as the "outer toggles", toggles 3 and 4 as the "middle toggles", and toggles 5 and 6 as the "bottom toggles". We will call the pathway through the lower tunnels of the bottom toggles the "bottom tunnel" of the overall gadget, and the rest of the gadget the "middle tunnel" of the overall gadget.

An NTL has two externally observable states: locked, and unlocked. The locked state corresponds to the upper tunnels of the bottom toggles oriented out, and the unlocked state corresponds to the bottom toggles oriented in. The unlocked state is shown in Figure 2-26.

In this gadget, there are two internal states corresponding to each external state: with the horizontal tunnels of the middle toggles both oriented left, and with both oriented right. The only accessible states of this gadget are the states with the outer toggles oriented in, the middle toggles oriented both left or both right, and upper pathways of the bottom toggles oriented both in or both out. We will show that the gadget allows exactly the traversals of the NTL from these configurations, and cannot be left in any other configuration.

The bottom tunnel traversals are straightforward — the bottom tunnel acts as a toggle, and a traversal flips both bottom toggles, and hence the externally observable state.

Also clearly, the robot cannot move between the bottom tunnel and the middle tunnel.

Now, we wish to establish that in the unlocked state, the robot can always traverse the middle tunnel in either direction. In the state shown, the middle tunnel may be traversed from external location to external location as follows:

- The robot can get across, left to right, by traversing the following toggles in the following order: enter through toggle 1's lower tunnel, down to toggle 5, up to toggle 4's vertical tunnel, through toggle 1's upper tunnel, around the top to toggle 2's top tunnel, back down through toggle 4, back out through toggle 5, across through toggle 3's horizontal tunnel, then through toggle 4's horizontal tunnel, then out through toggle 2's lower tunnel.

- The robot can get across, right to left, by traversing the following toggles in the following order: enter through toggle 2's lower tunnel, down to toggle 6, up to toggle 4's vertical tunnel, through toggle 2's top tunnel, around to toggle 1's top tunnel, down through toggle 3's vertical tunnel, back out through toggle 6, across through toggle 4's horizontal tunnel, then through toggle 3's horizontal tunnel, then out through toggle 1's lower tunnel.

- If the middle toggles are in the opposite orientation, the system is simply mirrored, left to right, and the traversals are still possible.

Next, we wish to establish that the robot cannot cross the middle tunnel in the locked state. After entering from either middle tunnel location, the only traversable toggles are the middle toggles. After traversing those, the robot can go no further. The bottom toggles can not be traversed, so the entire middle region is inaccessible. As a consequence, the opposite outer toggle's upper pathway can not be accessed. Therefore the robot can only leave via its original location.

We also must establish that if the gadget starts in one of the configurations mentioned, the robot must leave it in the proper state, and can not leave it in a configuration that was not mentioned. This is straightforward for the bottom tunnel, so we will focus on the middle two locations.

We will show that the accessible configurations of the gadget are exactly as described. To do so, we will make use of the concept of a cut in a gadget.

Lemma 19. *Let A be a connected region of a planar embedding of a gadget system which does not contain any locations. Then the boundary of A, which we will call a cut, is traversed an even number of times during any traversal of the construction.*

Proof. Whenever the boundary of A is crossed, the robot goes from inside A to outside or vice versa. Since the robot starts a traversal outside A and ends it outside A, it must cross the boundary an even number of times. □

The upper pathways of the outer toggles form a cut, and the lower pathways of the outer toggles form a cut. Thus, the upper pathways of the outer toggles are crossed an even number of times, and the lower pathways are passed an even number of times, so the outer toggles must be passed an even number of times in total. Thus, the toggles must either be both oriented in or both out when leaving. However, when leaving the gadget, the outer toggle which the robot exited through must end up oriented in, so both outer toggles must end up oriented in.

The vertical pathways of the middle toggles form a cut. The horizontal pathways form a cut. Thus, upon leaving, the middle toggles must have been traversed an even number of times in total, and hence must end up both left or both right.

The upper pathways of the bottom toggles must be passed an even number of times. So the upper pathways of those toggles must either be both in or both out when leaving the gadget system.

Thus, the gadget system must be left in a state where the outer toggles are oriented in, the middle toggles are oriented either both left or both right, and the upper pathways of the bottom toggles are oriented either both in or both out. Therefore, these are exactly the accessible configurations, as desired.

Finally, we show that the robot leaves the gadget in the same state it was entered in, if it is entered on the middle tunnel. If the robot passes through one of the upper tunnels of the bottom toggles, when it leaves the region bounded by the bottom toggles' upper tunnels, it must leave one of the bottom toggle's upper tunnels oriented in. By the parity constraint, both bottom toggles' upper tunnels will be oriented in, thus leaving the gadget in the unlocked state. If the central tunnels are entered in the unlocked state, they will be left in the unlocked state. In the locked state, the upper tunnels of the bottom toggles cannot be passed, and so the gadget will be left in the locked state.

Thus, the construction correctly simulates a NTL. □

We introduce some new three tunnel objects. There are several distinct planar topologies of the tunnels in a three tunnel object. We will focus on the two topologies which can

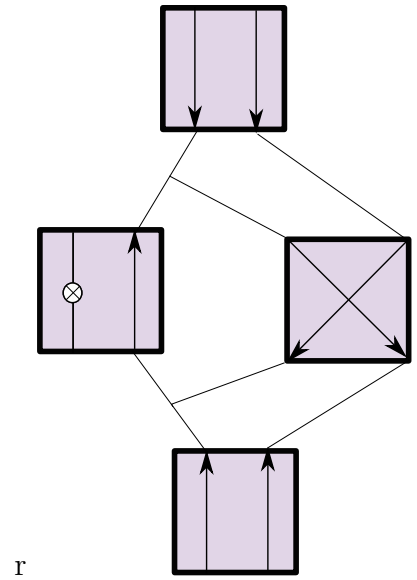


Figure 2-27: Round antiparallel 2-toggle-lock construction

be drawn with no internal crossing tunnels: three tunnels around the perimeter, and three tunnels in parallel. We will call the former a "round" topology, and the latter a "stacked" topology. Note that in the stacked topology, the order of the tunnels is relevant. In either topology, if there are multiple toggles, the relative orientation must still be specified.

Lemma 20. *2-toggles and noncrossing toggle locks simulate a round antiparallel 2-toggle-lock (RAP2TL) and a round parallel 2-toggle-lock (RP2TL).*

Proof. The construction shown in Figure 2-27 simulates the behavior of a round antiparallel 2-toggle-lock. It has two externally accessible states: as shown, and with the middle two gadgets flipped. These correspond to the 2-toggle of the RAP2TL being pointed counter-clockwise and clockwise respectively.

We will demonstrate that this gadget is equivalent to a RAP2TL by examining all possible traversals. From the two locations that are on the lock tunnel of the NTL, the only possible traversals are to each other, if the lock tunnel is unlocked. This forms the lock tunnel of the RAP2TL.

Traversals from the top left location: The robot must go down and to the right, due to the orientation of the toggle of the NTL. Then, the robot can go through the C2T, at which

point it is blocked by the orientation of the bottom P2T. Thus, no traversal is possible from this location in this state.

Traversals from the top right location: The robot can go through the C2T, then through the NTL. At this point, the robot cannot go through the C2T again, because the C2T has been toggled. Therefore, its only option is to go through the upper P2T and leave at the top left location. This traversal toggles both of the middle two gadgets, and toggles the upper P2T twice. Thus, the external state of the gadget is flipped. This is the equivalent of traversing the upper toggle of the RAP2TL that we are simulating.

Traversals from the bottom left location: The robot must go up and to the left, due to the orientation of the C2T. Then, the robot can go through the NTL. Due to the orientation of the upper P2T, the robot must now go through the C2T. Now, the robot can leave at the bottom right location. This traversal toggles both of the middle two gadgets, and toggles the lower P2T twice. Thus, the external state of the gadget is flipped. This is the equivalent of traversing the lower toggle of the RAP2TL that we are simulating.

Traversals from the bottom right location: The robot is blocked by the orientation of the C2T. Thus, no traversal is possible from this location in this state.

The opposite state is equivalent to a top-bottom mirror reversal, except for a change in the state of the lock, which does not affect which traversals are possible. Thus, in every state, this system of gadgets is equivalent to a round antiparallel two-toggle-lock (RAP2TL).

Consider the gadget which is the same as the one in Figure 2-27, except that the bottom P2T is replaced with a C2T with its toggles allowing traversals from the bottom locations into the gadget. Clearly, the effect of this change is to swap the roles of the bottom two locations. As a result, this new construction is a round parallel two-toggle-lock, a RP2TL. □

Lemma 21. *RP2TLs and 2Ts simulate a stacked antiparallel 2-toggle-lock (SAP2TL).*

Proof. A SAP2TL is a three tunnel gadget where the three tunnels cross the gadget in parallel, with the two antiparallel toggle tunnels next to each other.

Starting with a RP2TL and two C2Ts, we can simulate a SAP2TL as shown in Figure 2-28. The lock tunnel is straightforward. The two other traversals are from the top left to the

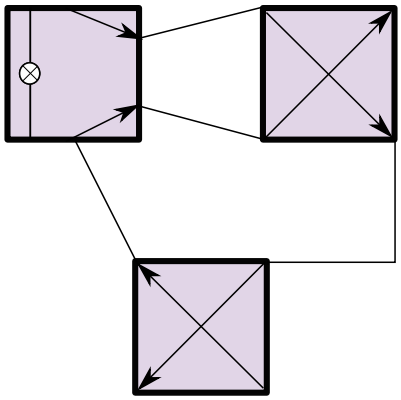


Figure 2-28: A round parallel 2-toggle lock is used to construct a stacked antiparallel 2-toggle lock

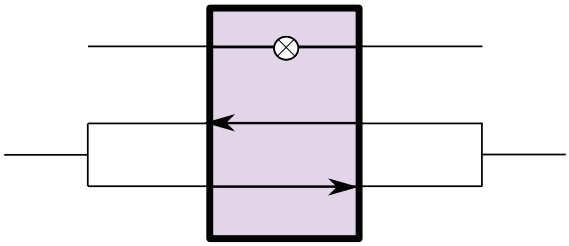


Figure 2-29: A noncrossing tripwire lock constructed from an anti-parallel 2-toggle and lock with the lock on the side

bottom left, and from the bottom right to the top right. Both of these traversals pass through every gadget. In the other state, all three gadgets are flipped, and the same traversals are possible in the opposite direction.

Since every state-affecting traversal traverses all gadgets, the states of the three gadgets always switch together, and the behavior is that of an SAP2TL. Equivalently, by Lemma 4, the system of gadgets is deterministic and reversible, so the three traversals mentioned are the only ones possible, and the construction simulates a SAP2TL. □

Lemma 22. *AP2TLS simulates a NWL.*

Proof. By connecting the locations of the SAP2TL as shown in Figure 2-29, we can simulate a NWL.

Each traversal of either connected toggle tunnel flips the state. The connections between these two tunnels ensure that travel in either direction is always possible. As a result, the combination of these connected pathways acts as a tripwire, always allowing the robot to pass in either direction and opening or closing the lock with each traversal. □

On our way to simulating a crossover, we will simulate another three tunnel gadget, a stacked tripwire-lock-tripwire (SWLW). Note that the lock tunnel is specifically the center tunnel.

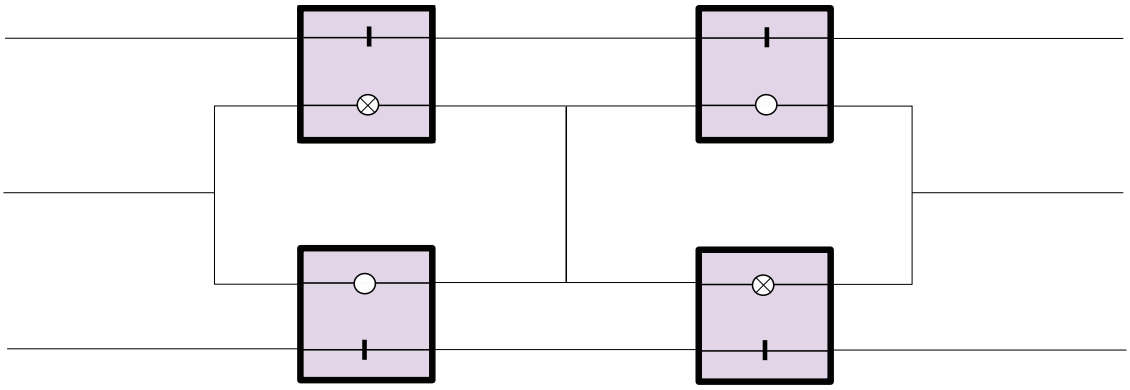


Figure 2-30: A stacked tripwire-lock-tripwire constructed from non-crossing tripwire locks.

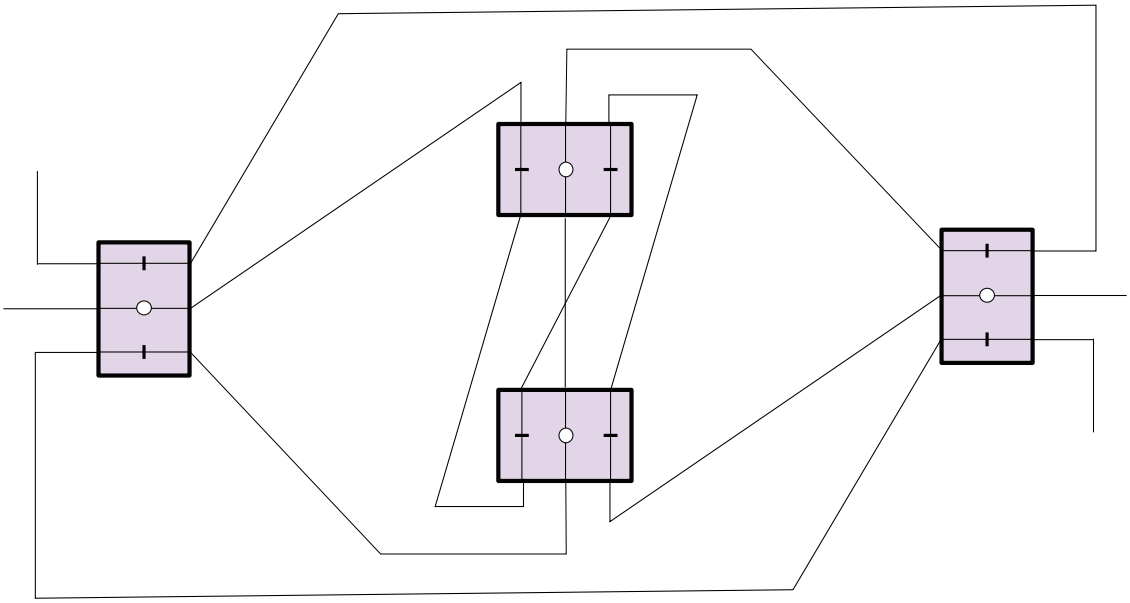


Figure 2-31: A crossover constructed from stacked tripwire-lock-tripwires

Lemma 23. *NWLs simulate a stacked tripwire-lock-tripwire (SWLW).*

Proof. The construction is shown in Figure 2-30. There are four accessible states of this gadget, which are any of the states where there is one locked and one unlocked NWL among the two top NWLs, and one of each among the two bottom NWLs.

The states can only be changed by traversing the tripwire tunnels, and doing so flips both NWLs on the side traversed, maintaining the invariant.

If both left NWLs are locked, or both right NWLs are locked, the center tunnel is not passable. In the other two accessible states, the center tunnel is passable. The two pairs

correspond to the two external states, with the lock locked and unlocked respectively. In any state, traversing either tripwire moves the gadget to a state with the opposite passability of the lock tunnel. Thus, this construction simulates a SWLW. □

Lemma 24. *SWLWs simulate a crossover.*

Proof. The gadget shown in Figure 5-9 implements a crossover. The robot may always cross from left to right, right to left, top to bottom and bottom to top, but in no other directions. There is a single accessible state, the one with all four SWLWs in the unlocked state.

When the robot enters from any of the four external locations it has only a single option up until the point where it reaches the four-way intersection at the center. Upon reaching this point, the robot has traversed the tripwire tunnels of two of the SWLWs, locking them. In particular, the SWLWs whose lock tunnels are on the two orthogonal pathways are locked. For instance, if the robot entered from the top, the left and right pathway's SWLWs would be locked at this point. As a result, the only way for the robot to continue is to go straight, passing through the other tripwires of the same two SWLWs, and emerging from the other side. The robot has completed a crossover traversal, with no other options.

Because the robot passed through the tripwires of two SWLWs twice, and only the lock tunnels of the other two SWLWs, the object is left in its original state, making the state shown in Figure 5-9 the only accessible state. This construction correctly simulates a crossover. □

Lemma 25. *AP2Ts simulate an NWT.*

Proof. We will construct a NWT as shown in Figure 2-32. This requires NWLs, crossovers, and 1-toggles. We already have existing constructions of NWLs and crossovers with AP2Ts. We can also build a 1-toggle with an AP2T simply by ignoring one of the two tunnels. Thus, all that is left is to show that the construction successfully simulates a NWT.

There are four accessible states: As shown in Figure 2-32, with all of the NWLs flipped, with the toggle flipped, and with everything flipped. The first and last correspond to the external state where the toggle is pointed right, while the other two correspond to the external state where the toggle is pointed right. The horizontal tunnel corresponds to the

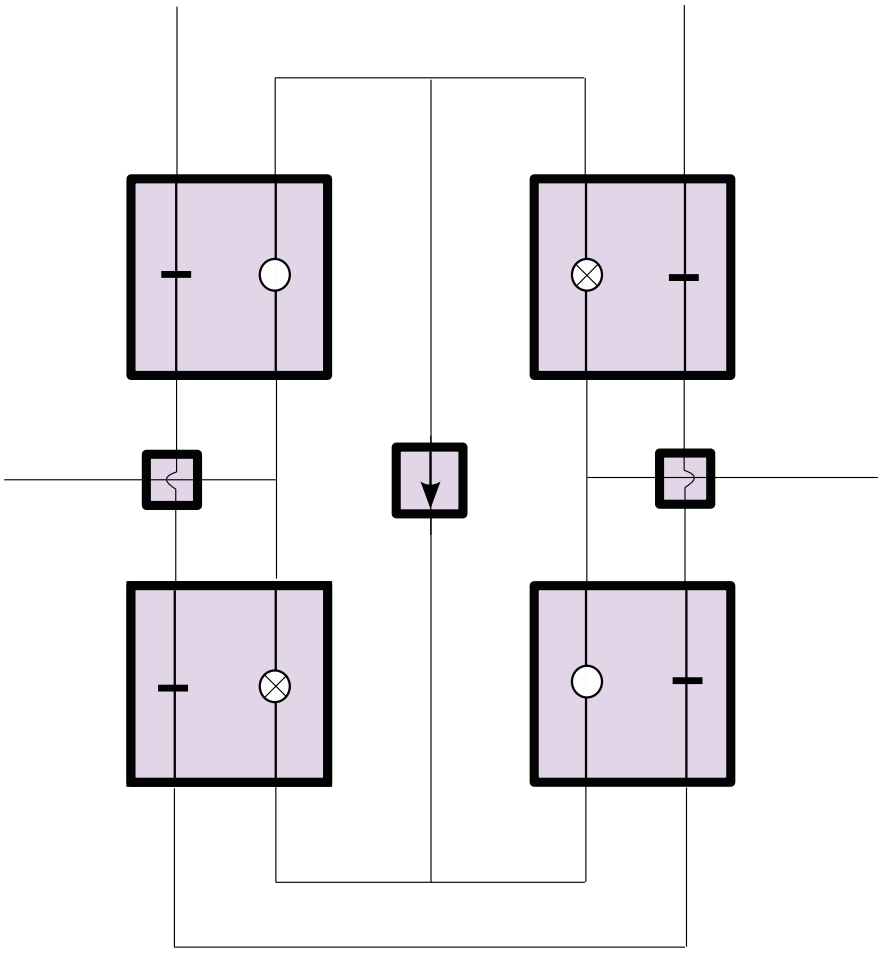


Figure 2-32: A noncrossing wire toggle constructed from a toggle, four noncrossing tripwire locks, and two crossovers.

toggle, while the U-shaped tunnel corresponds to the tripwire in the composed gadget. In the state shown in the figure, the toggle is oriented to the right from the external perspective.

Clearly, traversing the U-shaped tunnel will flip all of the tripwires of the NWL, resulting in a state which corresponds to the opposite external state, as desired.

In the state shown in the figure, the horizontal tunnel may be traversed from left to right along a unique pathway due to the placement of the locks, flipping the toggle along the way. The orientation of the toggle blocks the right to left traversal. Thus, in this state, the upper tunnel may be traversed in one direction resulting in an allowed state which corresponds to the opposite external state, as desired.

Placing the toggle in the opposite state is equivalent to a rotation by π of the upper

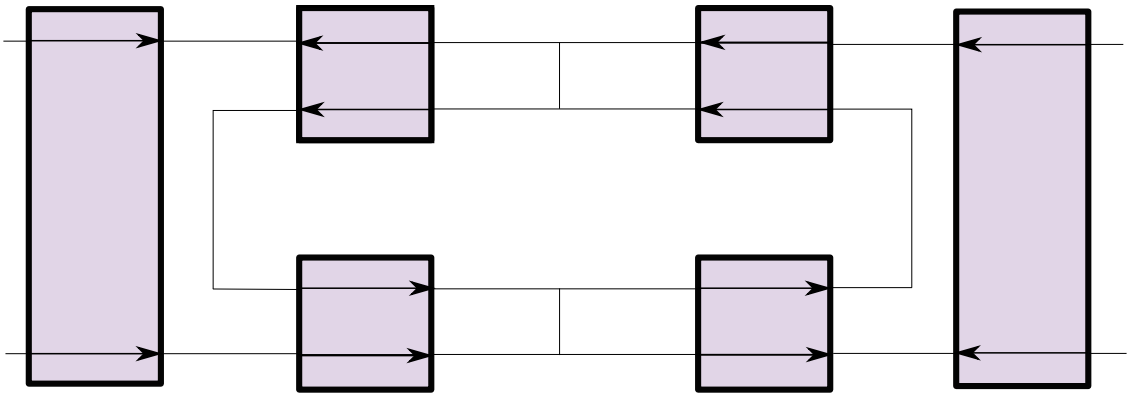


Figure 2-33: Parallel 2-toggles simulate anti-parallel 2-toggles

tunnel, showing this state also correctly simulates an NWT.

Flipping the states of all of the NWLs is equivalent to a vertical reflection of the upper tunnel, showing this state also correctly simulates an NWT.

Thus, we have built an NWT. □

Lemma 26. *P2Ts simulate an AP2T.*

Proof. Figure 2-33 gives a construction of an antiparallel-2-toggle out of parallel-2-toggles.

There are two accessible states: As shown, and with the four inner P2Ts flipped. The former corresponds to the AP2T having a tunnel connecting the left two locations with its toggle oriented upward, and a tunnel connecting the right locations with its toggle oriented downward, while the latter corresponds to the two toggles flipped.

First, let us examine the bottom right location in the state shown in the figure. After passing the rightmost P2T, the robot is blocked. No transitions or state changes are possible. This matches the desired behavior, because the right toggle in the AP2T being simulated is oriented down.

Next, let us examine the top right location in the state shown in Figure 2-33. After passing the rightmost P2T, then the upper right P2T, the robot may now either proceed along the top tunnel, or down to the central loop. In the former case, the robot may pass through the upper left P2T, but then is blocked. In the later case, the robot may either proceed around the loop to the left or to the right. If the robot goes to the right, it can pass

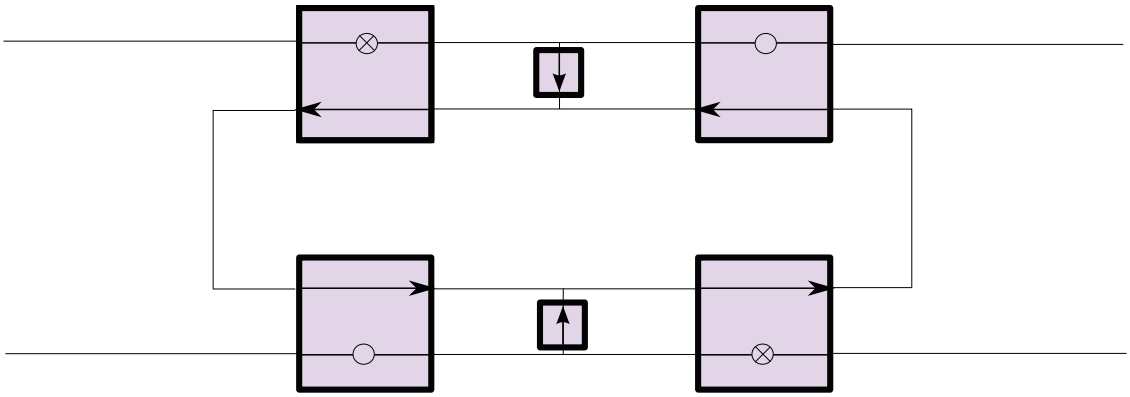


Figure 2-34: Noncrossing-toggle-lock simulates anti-parallel-2-toggle

through the lower tunnel of the upper right P2T, but then is stuck. If the robot goes to the left, it can pass through the lower tunnel of the upper left P2T, then the upper tunnel of the lower left P2T.

At this point, the robot may either continue around the loop, or exit the loop downward. If the robot continues around the loop, it can pass through the upper tunnel of the lower right P2T, but then is stuck. If it exits the loop, it can either go left or right on the bottom tunnel. If it goes left, it can pass through the lower tunnel of the lower left P2T, but then is stuck. If it goes right, it can pass through the lower tunnel of the lower right P2T, then the lower tunnel of the rightmost P2T, and exit the gadget.

Overall, we observe that the robot can make exactly one transition, from top right to bottom right. The right toggle is traversed twice, and the inner toggles are all traversed once, leaving the gadget in the other accessible state. No other transition or state change is possible, from that entrance.

Since the gadget is rotationally symmetric about its center, the possible transitions from the right mirror the possible transitions from the left. Since the other state is simply the state shown in the figure mirrored top-to-bottom, the transitions described mirror the transitions in the other state as well.

Thus, the construction simulates an AP2T. □

Lemma 27. *NTLs simulate an AP2T.*

Proof. The construction is shown in Figure 2-34. The two accessible states are the state shown in the figure and the state with all of the NTLs flipped, but the one-toggles still oriented inward. These correspond to an AP2T with the top tunnel directed left and bottom tunnel directed right, and the left-right mirror image.

If the robot enters from the top right, after passing the lock of the top right NTL, it must pass the upper one-toggle and proceed into the central loop. Since the lower toggle is directed upward, the robot must eventually leave the central loop via the upper toggle. The robot may now proceed around the loop. The loop may only be traversed counterclockwise, and it may only be traversed once. The robot may of course backtrack at any point, but when it leaves via the upper toggle, it must have either traversed the loop zero or one times. In the former case, the robot must leave via the top right location, leaving the system in its original state. In the latter case, the robot must leave via the top left location, as all of the locks have flipped. Thus, the top tunnel may be traversed via a right to left traversal, flipping the state, and that is the only traversal in that direction.

If the robot enters from the top left, it is immediately blocked by the lock, and no traversal is possible. Thus, the top tunnel works as desired.

Since the gadget possesses rotational symmetry around its center, the bottom tunnel is exactly the same, allowing only a left to right traversal, flipping the state.

The opposite state is the same as the original state except for a left-right right mirror reversal, so it also functions exactly as desired from the AP2T.

Thus, we have constructed an AP2T. □

Lemma 28. *NWTs simulate an AP2T.*

Proof. A noncrossing wire toggle can simulate an anti-parallel 2-toggle with the simple construction shown in Figure 2-35. The direction of each tunnel is dictated by the toggle on the tunnel, and the wire ensures both toggles are synchronized. Thus when either tunnel is traversed, both NWTs flip and the direction each tunnel can be traversed flips. □

Lemma 29. *NWLs simulate an AP2T.*

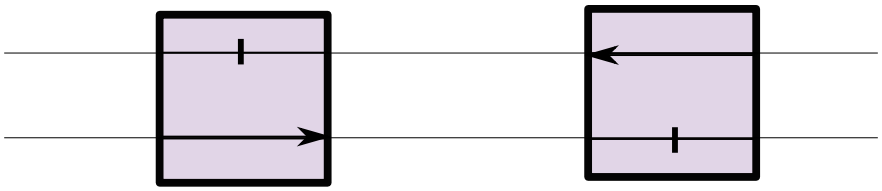


Figure 2-35: Noncrossing-wire-toggle simulates anti-parallel-2-toggle

Proof. The construction of an anti-parallel 2-toggle from non-crossing tripwire locks can be seen in Figure 2-36. Note that a 1-toggle can be constructed from an NWL by simply connecting one location of the wire to one location of the lock. A closed lock will prevent travel in one direction, but crossing the tripwire in the other direction will open the lock and allow the robot to proceed. An open lock will allow travel in the other direction. In the direction starting from the tripwire, the tripwire will close the lock in front of the robot preventing traversal. In either traversal, the tripwire is crossed, flipping the state.

There are two main parts to this gadget, the top and bottom tunnels, and the inner loop. As with the NTL construction from Lemma 27, the 1-toggles ensure that the loop must be exited from the same place it was entered, which ensures all gadgets on the loop are traversed the same number of times. Since all wires are on this loop, in a given traversal of this gadget system, all of the NWLs will change state the same number of times, keeping them in sync. The upper and lower paths each contain a locked and unlocked tunnel. The locked portion prevents entry and interaction with the gadget. From the unlocked side, the robot is able to enter the gadget and flip its state an arbitrary number of times. If the state is flipped an even number of times, the robot's only path out is the way it came. If an odd number of flips have occurred, the robot can now exit through the opposite side of its path, leaving the gadget in the opposite state.

Therefore, the gadget may traversed right to left along the top tunnel, flipping the state, and left to right along the bottom tunnel, flipping the state. We have built an AP2T. □

Lemma 30. *CWTs simulate an NWT, CWLs simulate an NWL, CTLs simulate an NTL.*

In general, one can very easily simulate a non-crossing version of a 2-tunnel gadget from the crossing version. Figure 2-37 shows a parallel-2-toggle being constructed from a

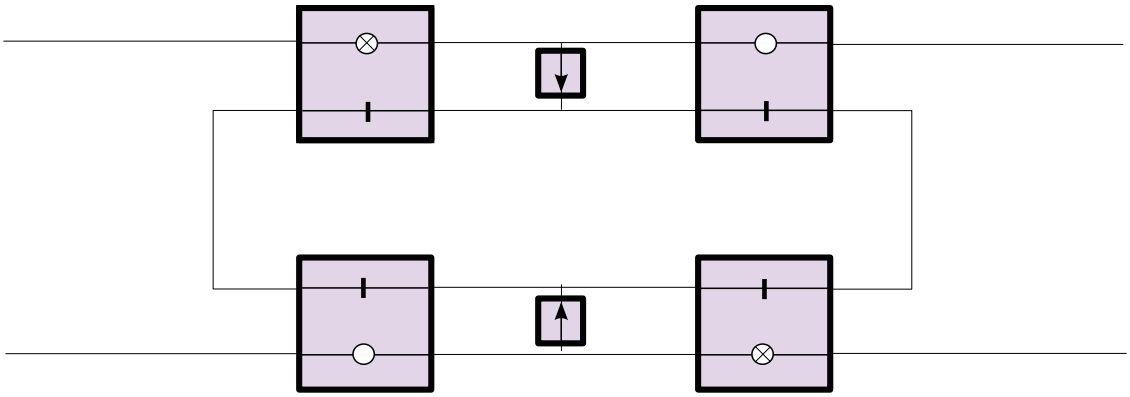


Figure 2-36: Noncrossing-wire-lock simulates anti-parallel-2-toggle

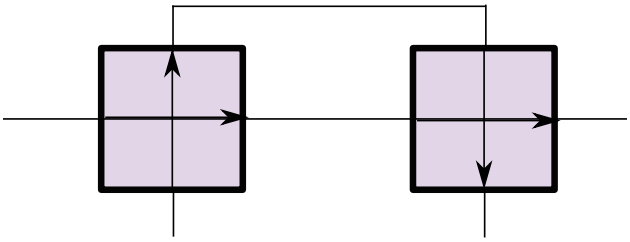


Figure 2-37: Crossing 2-toggles simulate parallel 2-toggle

crossing-2-toggle. The same construction works for uncrossing the other gadgets we have analyzed, namely tripwire-toggles, tripwire-locks and toggle-locks. Going from non-crossing to crossing versions is significantly more complicated (except in the case of anti-parallel-2-toggle to crossing-2-toggle) but we are rescued from the need of such constructions by being able to simulate a general crossover in Lemma 24.

2.2.7 Reconfiguring Reversible Gadgets

In this section we first show that for any reversible gadget the reachability problem being PSPACE-complete implies the reconfiguration problem is also PSPACE-complete. We then exhibit a reversible, deterministic gadget with non-interacting tunnels for which the reconfiguration problem is PSPACE-complete showing an example where reconfiguration is a harder problem than reachability. This work primarily comes from [8] written in collaboration with Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, and Dylan Hendrickson.

Theorem 31. *1-player reconfiguration motion planning is* PSPACE*-complete for any set*

of reversible gadgets for which 1-player reachability motion planning is PSPACE*-complete.*

Proof. We use the same technique for showing reconfiguration NCL is PSPACE-complete [45]. First we take a hard instance of the 1-player reachability motion planning. Now at the target location we add a loop with a single gadget which permits a traversal which changes its state. For the reconfiguration problem, we set the target states of all but the newly added gadget to be the same as the initial states, and we set the target state of the added gadget to be one reachable by making a traversal in the loop. Since the gadgets in this system are all reversible, the agent can always take the inverse transitions that have been made so far to return the start location and restore the states of all the gadgets to their initial states. Thus if the agent can reach the added gadget, the agent will be able to traverse the gadget and then undo the rest of the path except for that final traversal. Since the agent must interact with the added gadget to achieve the desired reconfigured state, the agent must be able to reach the gadget. Thus the agent is able to solve the reconfiguration problem if and only if the agent could solve the reachability problem. □

There are cases where the reconfiguration problem can be harder. Below we describe a 2-tunnel reversible deterministic gadget with non-interacting tunnels for which the reconfiguration problem is PSPACE-complete.

The non-interacting box gadget is a reversible, deterministic, 12-state, 2-tunnel gadget shown in Figure 2-38. We will refer to states the right-top and right-middle states as the right leaf states and left-bottom and middle-bottom states as the bottom leaf states. We call the right leaf states and their two adjacent states the right square and similarly the bottom leaf states and their two adjacent states the bottom square. Notice that from some states a tunnel either only once in the same direction or potentially twice. Although going through one tunnel never changes the traversability of the other tunnel, it may change whether that tunnel can be traversed twice in a row in the same direction. To show PSPACE-completeness we will first show that a cooperative, multi-agent reconfiguration problem is hard by reduction from 1-player reconfiguration motion planning with a locking 2-toggle. We then show how we can augment that construction to allow a single agent to simulate the

actions of all of the others in our multi-agent construction.

Multi-agent 1-Toggle. Recall a 1-toggle is a 2-state, 1-tunnel, reversible, deterministic gadget that allows a directed traversal in one direction in one state and the other direction in the other state. A regular 1-toggle can be easily constructed from the non-interacting box gadget by taking a single tunnel in a leaf state. Instead we will build a gadget that does not allow individual agents through at all, but if it has an agent on either side of it, a third agent can use the gadget as though it were a 1-toggle. Our construction will only work as intended if there are three or fewer agents adjacent to the gadget at any point in time; however, these gadgets will only be used in a way that this condition remains satisfied.

To build our multi-agent 1-toggle we simply connect the tunnels together, as shown in Figure 2-39 and consider the bottom-middle state to be the canonical configuration for the leftward pointing state of the 1-toggle, and the middle-right state to be the canonical configuration for the rightward pointing state. More configurations will need to be considered shortly, but we first describe the intended usage. In the bottom-middle state two agents can move up into the middle connection of the gadget, then the remaining agent can moves left joining them in the middle connection. The gadget is now in one of the upper left states. If one agent exits down, the other two can then exit to the right putting the gadget in the desired middle-right state with two agents on the right side. The transition in the other direction is symmetric.

Next we argue that the only ways agents can move through this gadget are equivalent to the intended usage. First, consider the case where there is no more than one agent on either side of the gadget. From the bottom middle state no right or down traversals can be made. Further, if no more than one up traversal is made then no more than one down traversal can be made, and the same is true for right and left. Thus the agents can put the gadget into four pairs of agent location and gadget state pairs, including only the bottom square states. Rather than just the middle-bottom state with one agent on each side, we actually consider these four agent and state pairs to be the rightward facing state of our multi-agent 1-toggle. Importantly these are the only reachable agent and state pairs and none of them have more

than one agent on any side of the gadget.

Now consider the case where the gadget is rightward facing and there is a second agent on the right side. None of the previously reachable states will allow more than one left transition, and thus the second agent is unable to interact with the gadget.

Finally we are back to the case of a rightward pointing gadget with an extra agent on the bottom. In this state there can be no more downward traversals than upward traversals and there can be at most one more rightward traversal than leftward traversals. Thus we cannot move extra agents to the left side of the gadget and we can move at most one extra agent to the right side of the gadget. Further, after making two right traversals, there be at least two agents on the rightward side of the gadget and the gadget must be in a right leaf state. Thus, once there are two agents on the rightward side, we are in one of the right square states above with an extra agent. This is exactly the situation where the multi-agent one-toggle has changed state and allowed an agent to traverse it.

Multi-agent Locking 2-Toggle. The multi-agent locking 2-toggle will be comprised of one non-interacting box gadget, four multi-agent 1-toggles, and six helper agents. It will allow an additional agent to interact with it as though it were a locking 2-toggle. Two helper agents will be located in the horizontal and vertical connections next to the non-interacting box gadget, and the other four agents will be external, each adjacent to one of the multi-agent 1-toggles. Note, these external four agents will be shared between gadgets rather than duplicated.

The canonical unlocked state is shown in Figure 2-40 with the non-interacting box gadget in the upper-left state, the 1-toggles pointing right and down, and the internal agents in the left and top sides respectively. If a second agent comes in from the top, it is able to cross the first 1-toggle, both agents can move down through the non-interacting box gadget, and then the two agents can allow one of them to cross the second 1-toggle. We consider this resulting state to be the canonical up locked state. The non-interacting box gadget is in the lower left state, the 1-toggles are pointed right and up, and the internal helper agents are on the left and bottom. The right locked state and transition to it are symmetric.

From the locked bottom state, if an agent arrives on the left, it could move through the 1-toggle, but only one agent would be able to pass the non-interacting box gadget. Thus it would not be able to pass the second 1-toggle preventing a traversal of our gadget as desired. Notice in the locked states the internal helper agents will only be able to shift the state of the gadget over once, ensuring that they cannot move the gadget outside of the bottom or the right square states respective to being locked up or left. This is the property that ensures only one agent can cross the non-interacting box gadget along the incorrect pathway while it is locked. The other traversals are prevented by the directionallity of the multi-agent 1-toggles.

Now we must inspect the construction as a whole because our gadgets depend on never having more than two agents on any side and no more than three adjacent agents total. In this construction all pathways have two multi-agent 1-toggles between every non-interacting box gadget. Since individual agents can freely cross the non-interacting box gadget, let us imagine replacing them with simple connections. Now every multi-agent 1-toggle (except next to the start location) has exactly 1 agent on either side of it. There is only one additional agent in the entire construction, so from the properties of the multi-agent 1-toggles we know that we will never end up with more than one extra agent adjacent to any of the gadgets fulfilling our needed condition.

The states of the locking 2-toggles in the single agent 2-toggle reconfiguration can now be represented by the canonical agent location and gadget pairs in this multi-agent reduction. The movement of the agent is represented by the movement of the location containing two agents. Since the doubled agent can only move through the system of gadgets in the same way as the single agent in the original instance, and we have shown that the canonical pairs are reachable if the original instance is true, but are disjoint from reachable states which do not correspond to a valid traversal in the original instance we obtain PSPACE-hardness for the cooperative multi-agent reconfiguration problem with the non-interacting box gadget.

Simulating Extra Agents. Now we wish to simulate the multi-agent reduction with a single agent. Recall that we can directly build a (single agent) 1-toggle out of the non-

interacting box gadget. Also recall that our reduction ensured that no connection contained more than two agents at any given point in time. For each connection in the multi-agent instance, we connect two 1-toggles to that connection, each representing a potential agent. The other side of all the one toggles are connected together so that this central location has access to all of our simulated edges. If the multi-agent instance has an agent at some location, in the single-agent instance we direct a number of 1-toggles towards that connection for each agent there. All other 1-toggles are directed towards the central connection. Finally we start the agent in the central connection.

From the central connection, the agent is able to cross a 1-toggle "instantiating" the agent it represents in the multi-agent problem. The agent is then in the same location and able to interact with original instance. If the agent then traverses a 1-toggle from some connection to the central connection, the flip in direction of that 1-toggle will allow the agent to return effectively "remembering" the location of that agent in the multi-agent instance. Since the multi-agent problem never has more than two agents along the same connection, we need not worry about running out of 1-toggles to record the agent locations. If we pick one of the toggles to always flip when representing the presence of a single agent, we can directly map states of the gadgets onto pairs of gadget states and agent locations in the multi-agent instance, completing the reduction.

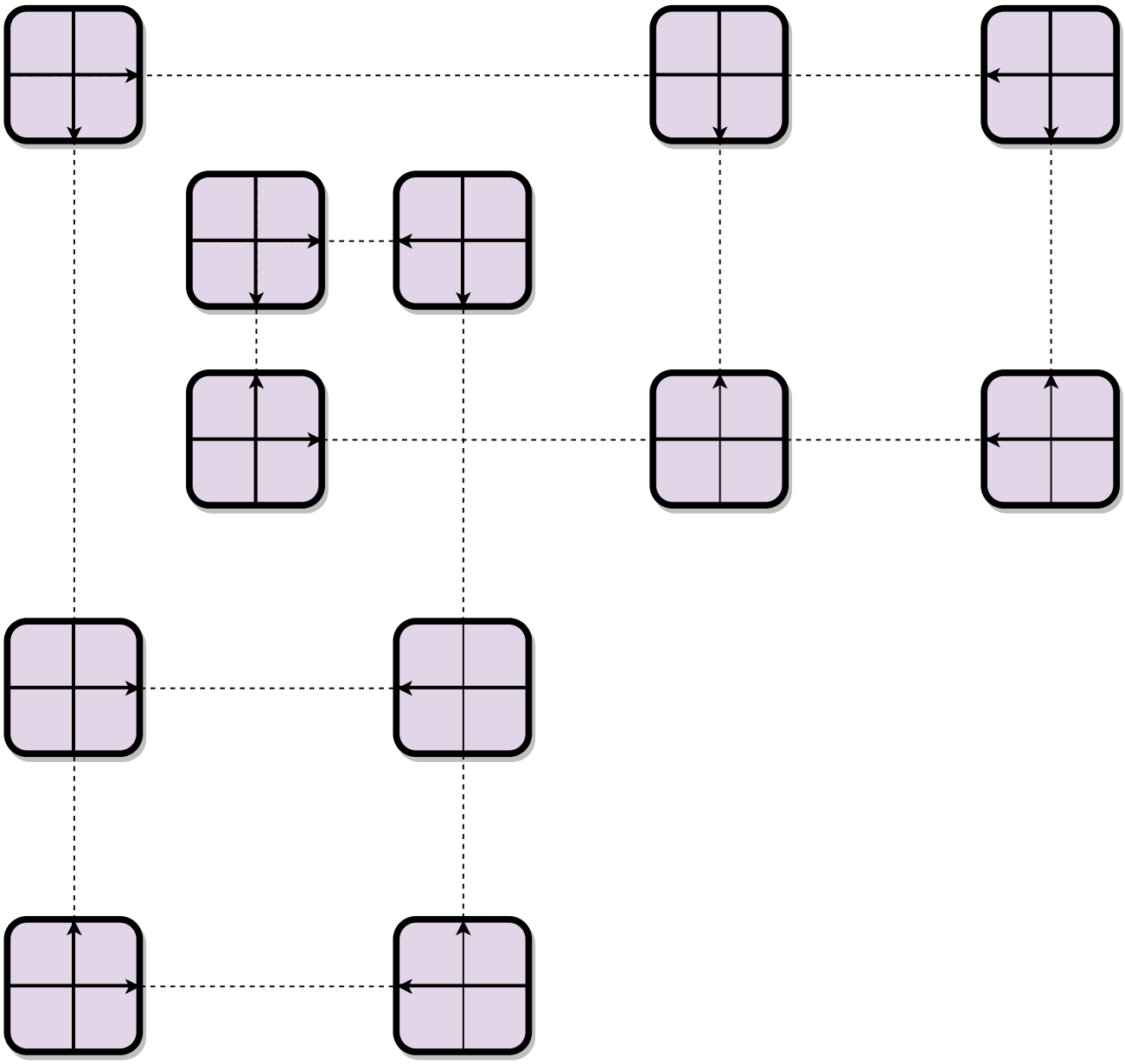


Figure 2-38: The state diagram of the non-interacting box gadget

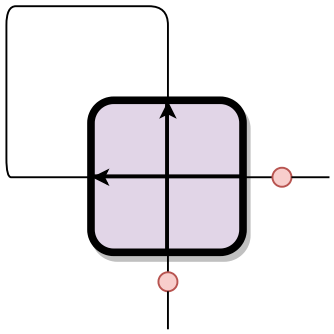


Figure 2-39: The multi-agent 1-toggle. The two helper agents are denoted by red dots.

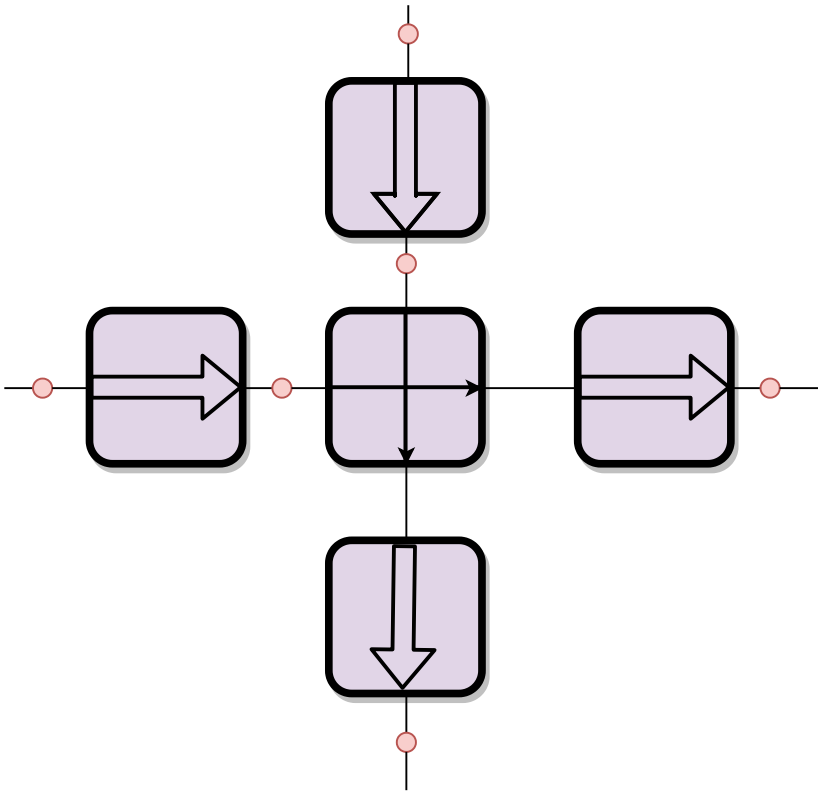


Figure 2-40: The multi-agent locking 2-toggle in the unlocked state.

2.3 1-Player Bounded Motion Planning

In this section, we consider a broad class of gadgets which are naturally in NP. These LDAG gadgets are the k-tunnel gadgets whose state graph forms a DAG with self-loops. Thus it is the class of k-tunnel gadgets which has a polynomially bounded number of state changes. For a subset of those gadgets, the DAG gadgets, which have a polynomially bounded number of transitions, we give a dichotomy classifying them as NP-complete or in NL in Section 2.3.5. We are also able to classify another subset, deterministic eventually statics gadgets, into NP-complete, P-complete, or in NL, given in Section 2.3.7. These proofs, which are more general than needed for our dichotomy theorems, also give a significant picture of the computational complexity of LDAG gadgets; however, a full characterization of LDAG gadgets remains to be completed.

The NP-hardness results can be seen as similar to Viglietta's Metatheorem 1 about location traversal (being implemented by the interacting tunnels in gadgets) and single-use paths [60]. They also bear resemblance to Metatheorem 4 about pressure plates which only affect one door [60]. However, our proof reduces from 3SAT rather than Hamiltonian Path, uses a different underlying model which makes different features salient, and gives generalizations in a different direction. Structurally the proof follows the proof structure used to show Mario as well as many other games are NP-hard [4].

We go on to consider other victory conditions. In Section 2.3.6 we give a dichotomy for LDAG gadgets under the shortest-path victory condition. In Section 2.4 we examine the reconfiguration victory condition, showing there are gadgets which are both harder and easier under reconfiguration and using it as intuition for showing a broader class of gadgets is in NP and another seemingly bounded class of gadgets actually has cases that are PSPACE-complete.

2.3.1 Upper Bounds

We start with a general upper bound showing LDAG gadgets are always in NP. In a system of gadgets, the LDAG gadgets can only undergo a polynomial number of state changes. This

is the core reason that motion planning involving these gadgets are in NP.

Lemma 32. *1-player motion planning with any set of LDAG gadgets is in* NP*.*

Proof. We will use as a witness a list of all state changing traversals taken in order by the agent on a solution path. This may be an empty set, but trivially exists as long as a solution path exists. Since the number of state changes is polynomially bounded, this witness is also polynomially bounded. To verify this witness we will construct a polynomial length path from the start to the goal which respects those state changing traversals.

Take the system of gadgets and construct a graph with locations as vertices and edges which are transitions in gadgets that do not change the gadget's state and edges of the connection graph. Search this graph for a path between the start location and the entrance of the first transition in the witness. Next, update the system of gadgets with transitions along that path and the first transition of the witness. For each pair of state changing transitions, construct the graph of non-state changing transitions as before and check that a path exists. After verifying that there are moves that allow the agent to get from the start to each state-changing transition and then to the goal, we have constructed a polynomial length solution to the problem instance. This additionally proves that if a solution to a 1-player motion planning problem with LDAGs exists, there exists a polynomial length solution. □

Since DAG gadgets are a subset of LDAG gadgets, containment in NP trivially follows for DAG gadgets. In Section 2.4.1 we will use a similar proof to show an even more general class of gadgets is in NP.

Corollary 33. *1-player motion planning with any set of DAG gadgets is in* NP*.*

2.3.2 Characterization Overview

Recall from Theorem 2 that all gadgets without interacting tunnels are in NL. Thus one might hope to show that all interacting-k-tunnel LDAG gadgets are NP-complete, similar to the classification for reversible deterministic gadgets. This hope is unfortunately not true for LDAG gadgets. We will define two behaviors a gadget may have: "distant opening" and

"forced distant closing." We show that having neither of these properties puts the gadget in NL. This behavior is similar to non-interacting, with an exception that an agent will generally not close a tunnel if there is an otherwise equivalent option not to do so. Reversible, deterministic gadgets do not have nondeterminism and always pair a distant opening with a distant closing and thus do not need this distinction. We then show forced distant closing suffices for NP-hardness. Distant opening by itself only suffices for P-hardness, but distant opening with a non-undoable tunnel does give NP-hardness. Since all DAG gadgets have a non-undoable tunnel, this will suffice to categorize DAG gadgets. We then dig a little deeper and show another condition for when door opening gadgets yield NP-hardness and will use this to give a dichotomy for deterministic eventually static gadgets.

2.3.3 Distant opening and non-undoable tunnel is NP-hard

A ***distant opening*** in a DAG gadget is a transition in some state across a tunnel which opens a different tunnel. A tunnel is ***opened*** if a transition has taken it from a state where the tunnel did not have traversability in some direction to a state where it is now traversable.

A ***non-undoable tunnel*** is a tunnel with locations A and B and a state s such that in state s there exists a transition from location A to location B and for all transition in state s from location A to location B the resulting state s' does not have any transition from location B to location A. Simply put, going across means you can not immediately go back over the tunnel in the opposite direction.

We now show NP-hardness for gadgets which contain these elements. This is comparable to a generalization of comparable to Forišek's Meta-theorem 3 [35] which covers a similar notion of door opening and diodes.

Lemma 34. *1-player motion planning with any k-tunnel gadget with a distant opening and a non-undoable tunnel is NP-hard.*

Proof. We show this problem is hard by a standard reduction from 3SAT. However, we will need to break it into three cases depending on the behavior of our non-undoable tunnel and each case will require a slightly different construction for the variable gadget.

Our construction makes use of the tunnel which is traversed in the distant opening and one of the tunnels it opens. Each literal in a 3-CNF formula will be represented by those two tunnels in a single gadget, in the state of the distance opening. Each variable x_i is represented by a connection to two different paths, one which goes through the opening transitions for the x_i literals, and one for the $\neg x_i$ literals. The non-undoable edges will need to be placed at the start and ends of these variable pathways to prevent the agent from opening both sets of doors.

Each clause contains connections between the openable tunnels for each of its literals. All variable gadgets are laid out in series followed by the clause gadgets, with the goal location at the end of the clause gadgets. Each clause gadget can only be traversed if at least one of its corresponding variable gadgets has been traversed, allowing at least one passage to be open. The agent can reach the goal location exactly when it has a path through the variable gadgets which makes each clause gadget traversable, which corresponds to a satisfying assignment of the 3-CNF formula.

Now that the overall construction is laid out, we return to the question of how to protect the variable gadgets. Recall our non-undoable tunnel has locations A and B and a state s such that (1) in state s there exists a transition from location A to location B; and (2) for all transitions in state s from location A to location B, the resulting state s' does not have any transition from location B to location A. We will call a gadget ***forward*** if the first location reach while going through a variable branch is A and ***backward*** if it is B.

Case 1: There is no traversal B to A in state s. In this case we put a copy of the non-undoable edge at the start and end of each variable branch as shown in Figure 2-41. After crossing the first non-undoable edge, the agent will not be able to go backward across that edge and will thus need to continue forward through the branch. The agent cannot go backward through either branch because the crossed non-undoable edge has no B to A transition and the uncrossed non-undoable edge on the other side has no B to A transition.

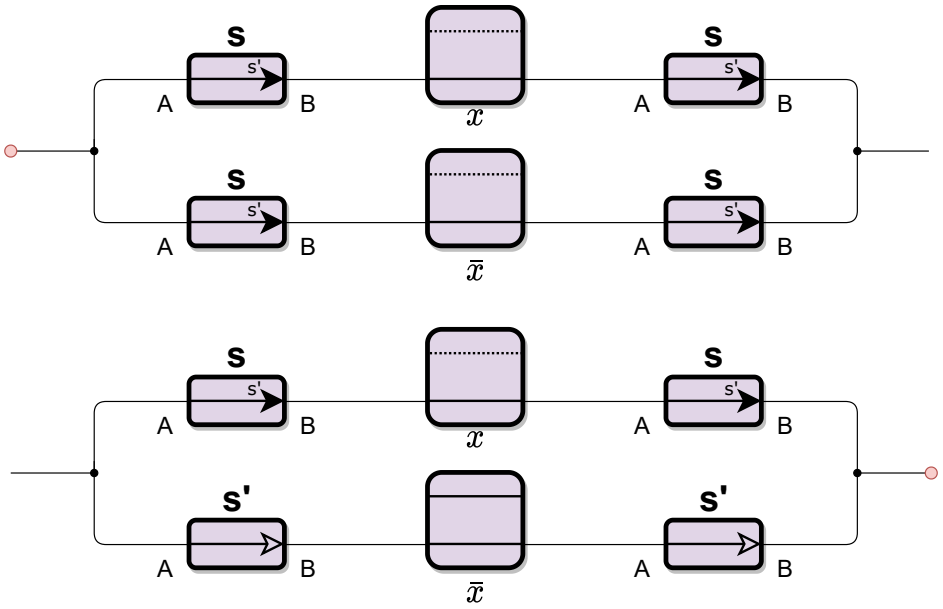


Figure 2-41: The variable gadget for distant door opening gadgets, Case 1. The agent is shown as a red dot; the top diagram shows the initial state; the lower diagram shows the state after the agent has traversed the lower branch of the variable gadget.

Case 2: There is a traversal B to A in state s but all s' do not have an A to B traversal. This case implies that after making the A to B traversal there is neither an A to B nor a B to A traversal, closing the tunnel. However, we do not know what behavior we will have if the gadget is first traversed from B to A as this will put us in some state which may not be one of our s'. For this case we put a pair of non-undoable tunnels at the start and end of each variable branch, one with location A first along the path and the other with location B first along the path as shown in Figure 2-42. When the agent goes through one branch the forward tunnels will close, preventing further use. Although the backward tunnels may be in some unknown state, the agent is unable to return along that branch. If the agent tries to enter the other branch, the backward non-undoable tunnel will undergo an A to B traversal, closing behind the agent; then the agent will be unable to progress past the variable gadget since both branches will have a gadget that has closed.

Case 3: There is a traversal B to A in state s and some s' (resulting from the B to A traversal from state s) has an A to B traversal. In this case we put two gadgets in a row at

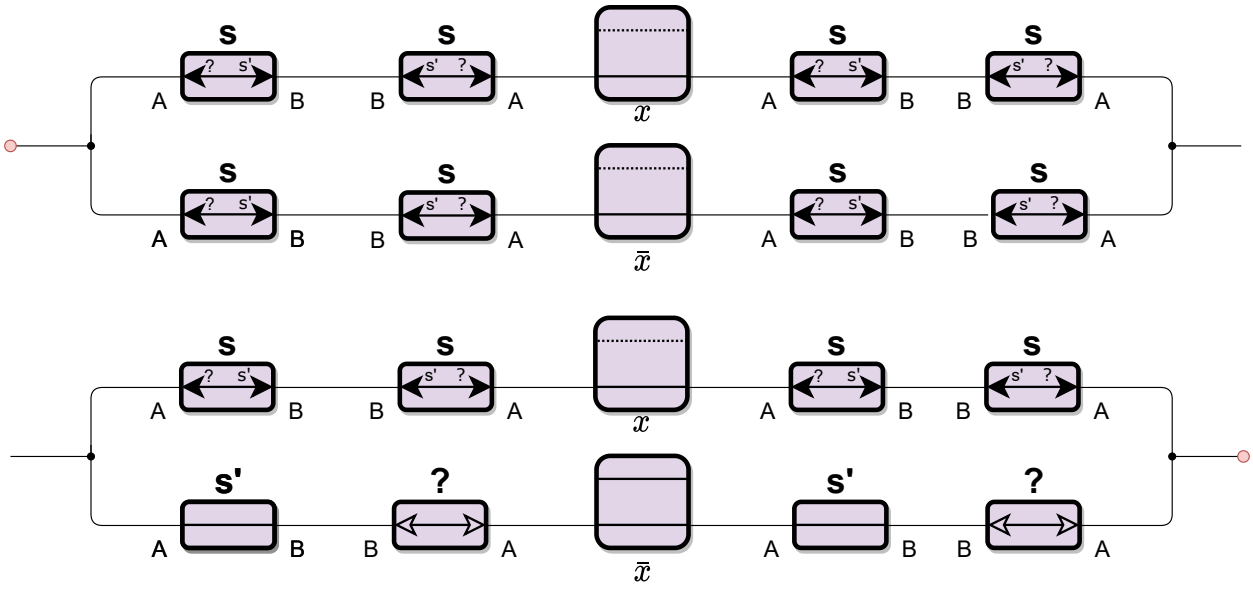


Figure 2-42: The variable gadget for distant door opening gadgets, Case 2. The agent is shown as a red dot; the top diagram shows the initial state; the lower diagram shows the state after the agent has traversed the lower branch of the variable gadget.

the start and end of each branch with a forward gadget in state s and a forward gadget in state s' as shown in Figure 2-43. After going through a branch we have the forward gadgets in state s' preventing backtracking. If the agent attempts to go backward through the other branch, we know the gadget already in state s' does not have a B to A traversal. □

We now note that all DAG gadgets have a non-undoable tunnel, noteably a single-use tunnel. This means a DAG gadget containing a distant opening is NP-hard.

Lemma 35. *All DAG gadgets contain a single-use transition unless they are a transitionless gadget.*

Proof. Call states which are sinks in the state graph of the gadget ***terminal states***. We now wish to find a state s which only has transitions to terminal states. This can be done by removing all terminal states from the state graph and locating a sink (which exists so long as there is at least one transition between states in the gadget). Terminal states in a DAG gadget have no transitions by definition. Thus s has at least one transition which is possible and then goes to a state which has no transitions, generating a single-use transition

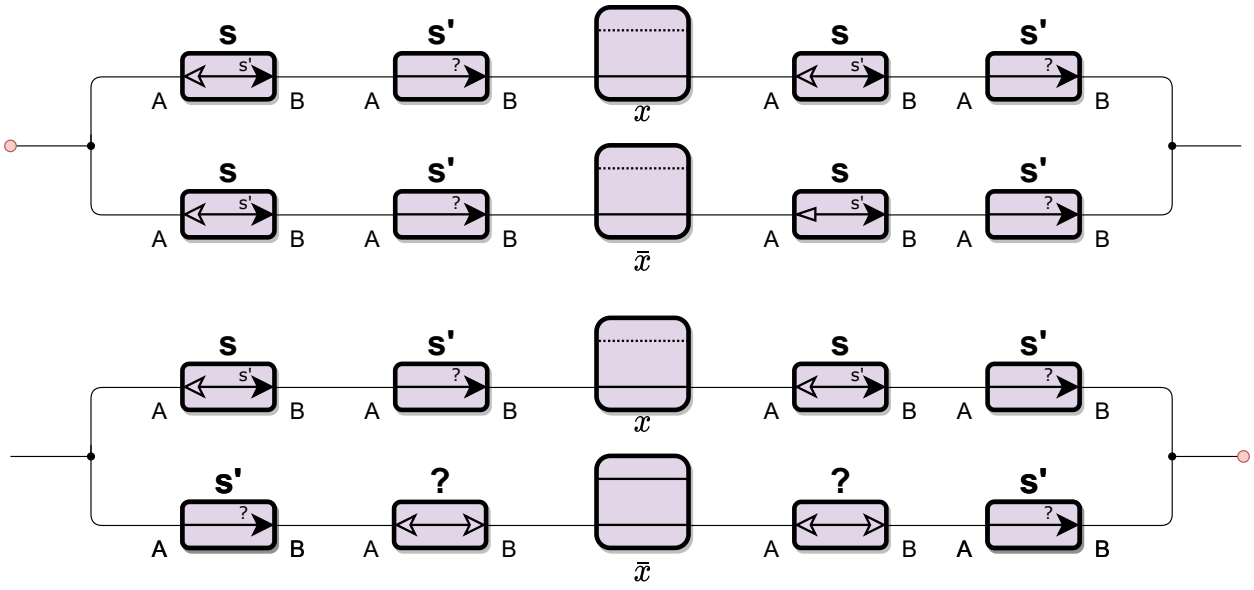


Figure 2-43: The variable gadget for distant door opening gadgets, Case 3. The agent is shown as a red dot; the top diagram shows the initial state; the lower diagram shows the state after the agent has traversed the lower branch of the variable gadget.

as desired. □

Corollary 36. *1-player motion planning with any k-tunnel DAG gadget which contain a distant opening is NP-hard.*

2.3.4 Forced distant closing is NP-hard

When a transition across a tunnel closes another tunnel, the situation is more complicated, since the agent may be able to cross the same tunnel through a different transition, choosing not to close the other tunnel. Thus, for NP-hardness we will need the agent to be "forced" to close the tunnel; however, defining this correctly is somewhat delicate. A tunnel is ***closed*** if a transition has taken it from a state where the tunnel was traversable in some direction to a state where it is no longer traversable in that direction. We will now consider only ***distant monotonically closing*** DAG gadgets, which are DAG gadgets with no distant openings, since we know distant openings suffice for hardness from the prior section and because the ability to re-open the tunnel that was closed would cause significant complication.

For NP-completeness one might suggest there exists a traversal such that all of its transitions close some other traversal. However, this property fails to give hardness in a simple two-tunnel case where one transition closes one direction of the other tunnel and the other transition closes the other direction of the tunnel. This leads us to a more complex definition. An ***orientation*** of a set of tunnels in a state contains, for each tunnel in the set, a single traversal of the tunnel in the state. A ***forced distant closing*** in a state of a gadget is a traversal across a tunnel in that state and an orientation of some other tunnels in the state such that, for each transition corresponding to the traversal the transition closes some traversal in the orientation. The ***size*** of a forced distant closing is the number of traversals in the orientation. Note that for deterministic gadgets, any distant closing is a forced distant closing of size one.

Lemma 37. *1-player motion planning with any distantly monotonically closing k-tunnel LDAG gadget with a forced distant closing is NP-hard.*

Proof. Consider all states which have forced distant closings, and let s be such a state that is minimal in the state-transition graph, so that after making a (non-self-loop) transition from state s there are no forced distant closings. Consider a forced distant closing in s with smallest size; say this forced distant closing traverses tunnel t and has size i. We chain the i tunnels in the orientation for the forced distant closing, in the directions specified by the orientation, to make what is effectively a single long tunnel r. We will use the tunnels t and r in a reduction from 3SAT, and they have two important properties:

- If the agent traverses t, it cannot later traverse r: since we are using a forced distant closing, after traversing t at least one (oriented) tunnel in r is not traversable. Since there are no distant openings, this tunnel cannot become traversable again.

- The agent can traverse r from state s: in state s, each tunnel in r is open. The agent begins by traversing the first tunnel in r. This cannot be a forced distant closing for the remaining $i - 1$ tunnels, since we assume the smallest forced distant closing has size i. So the agent can choose a transition which leaves the remaining tunnels in r

open. After this first traversal, there are no more forced distant closings, so the robot can always choose a transition which leaves the remaining tunnels in r open.

We can now describe the reduction, which is similar to the reduction in the proof of Lemma 34. Each literal in a 3-CNF formula is represented by a gadget in state s, with the tunnels in r chained together. Each variable x_i is represented by a connection to two different paths, one which goes through t for the x_i literals, and one for the $\neg x_i$ literals. Since traversing both sides will only decrease the traversability of the system of gadgets, the agent will never want to set both sides to false. We could also use single-use gadgets which can be constructed by attaching the path which closes to the path that is closed.

When the agent goes through the x_i (respectively $\neg x_i$) path of a variable, it closes r in the gadget for each literal x_i (respectively $\neg x_i$), which corresponds to assigning x_i to false (true). This behavior is the revers from the reduction in Lemma 34.

Each clause contains connections between the r tunnels for each of its literals. All variable gadgets are laid out in series followed by the clause gadgets, with the goal location at the end of the clause gadgets. Each clause gadget can only be traversed if at least one of its corresponding variable gadgets has *not* been traversed, leaving at least one set of tunnels open. The agent can reach the goal location exactly when it has a path through the variable gadgets which leaves each clause gadget traversable, which corresponds to a satisfying assignment of the 3-CNF formula. □

2.3.5 Putting together distant opening and closing and finishing the DAG dichotomy

We now combine the results of the prior two sections into a stronger lemma.

Lemma 38. *1-player motion planning with any k-tunnel LDAG gadget with a forced distant closing or a non-undoable tunnel and a forced distant opening is NP-hard.*

Proof. If we have a forced distant closing and no distant opening, then the gadget is NP-hard by Lemma 37. Otherwise we have a distant opening. If we additionally have a forced distant

closing, we can use it to construct a single-use (and thus a non-undoable) tunnel. This is then NP-hard by Lemma 34. If we do not have a forced distant closing, then we have both a non-undoable tunnel and a distant opening which is once again the case in Lemma 34. □

We now improve upon our lemma for non-interacting tunnels being in NL by showing a lack of distant opening or *forced* distant closing is sufficient to put the gadget in NL.

Lemma 39. *1-player motion planning with any k-tunnel gadget with no distant openings or forced distant closing is in* NL.

Proof. The proof follows that of Theorem 2, though we must be more careful to account for optional distant closings. As in Theorem 2, if a system of gadgets has a solution, then a solution of minimal length does not intersect itself. This only requires that the gadget has no distant openings, since then making transitions can never increase traversability, and the shortcutting argument applies.

We locally convert the system of gadgets into a directed graph, and show a path in the graph from the start location to the goal location corresponds to a solution to the system of gadgets which does not intersect itself. Given a (not self-intersecting) path in the graph, we follow the corresponding path through the system of gadgets. When we make a traversal, we must pick a transition to avoid closing tunnels we will need later. This is always possible because there are no forced distant closings; we can always choose a transition which does not close any traversal in the orientation consisting of the traversals the path will later take. Although this seems to require looking ahead and constructing a path, we can non-deterministically check all traversals; some of which will result in closing a needed pathway, but some of which will avoid the unwanted distant closing. By doing this, we ensure that every traversal we need is available when we get to it, so the system of gadgets is solvable.

Suppose there is a solution to the system of gadgets that does not intersect itself. Since it uses each tunnel at most once, and the gadget has no distant openings, the traversability of each tunnel does not change before the solution uses it. Thus the solution is also a path in the directed graph.

So the system of gadgets has a solution iff there is a path from the start location to the end location in the directed graph. Since we can locally convert the system of gadgets to the graph in logarithmic space and solve reachability in NL, the motion planning problem is in NL. □

Combining Lemmas 35, 33, 38, and 39, we have a dichotomy for DAG gadgets:

Theorem 40. *1-player motion planning with a k-tunnel DAG gadget is NP-complete if the gadget has a distant opening or forced distant closing, and otherwise is in* NL.

We will investigate the question of whether this condition can be tested in polynomial time in Section 2.3.9. But next we continue to explore distant opening gadgets first showing they suffice for hardness under the shortest-path victory condition in Section 2.3.6 and then reach a dichotomy for deterministic eventually static gadgets in Section 2.3.7.

2.3.6 LDAG Dichotomy for Shortest Paths

Here we consider the shortest-path alternate victory condition which asks whether the agent is able to reach the goal location in k moves. Under this victory condition, we are able to give a dichotomy for LDAG gadgets and find that it is the same characterization as DAG gadgets for reachability. In essence, the strict time limit is playing the role of a non-undoable edge in the reduction.

Lemma 41. *1-player shortest-path motion planning with any k-tunnel gadget with no distant openings or forced distant closing is in P.*

Proof. Recall the proof for Lemma 39. It argues that there is no need to ever revisit a tunnel for this class of gadget and thus the problem is equivalent to reachability in a static graph. The properties of these gadgets do not change under the shortest-path victory condition. Thus we can search for the shortest-path in the static graph with vertices represented connected components of gadget locations and edges formed by the traversability of the gadgets in the system of gadgets. Since edges taken directly correspond to transitions taken in the

system of gadgets, a shortest-path here will also be a shortest-path through the system of gadgets. □

For forced distant closing, NP-hardness remains by simply making the target number of moves much larger than the number of moves needed to solve the bounded system of gadgets. Forced distant opening is a more interesting case, and we will show it suffices for NP-completenss by a reduction from Hamiltonian Path. This is comparable to Forišek's Meta-theorem 2 [35].

Lemma 42. *1-player shortest-path motion planning with any k-tunnel gadget with distant openings is* NP*-hard.*

Proof. We reduce from Hamiltonian Cycle. We use the directed or undirected version as needed by the traversability of the gadget. Say the Hamiltonian Cycle has n vertices and m edges. The door opening property will be used to check that locations have been visited. From the start location we construct a path of length n which goes through the openable traversals of n different gadgets. We will now use the opening traversals of those gadgets as our vertices. For each edge, add a gadget and use a traversable tunnel. We connect both locations of our openable vertex to one side of each tunnel representing an edge. We set the target time to be $3n$. The agent must go through all of the door opening tunnels to be able to reach the goal. This will require at least $2n$ traversals from going through the edge gadgets and the opening tunnels, and another n traversals to go through the openable tunnels. If a Hamiltonian cycle exists, the agent can use this solution to guide a pathway to open all the tunnels in $2n$ moves and win within the time limit. If no such cycle exists then visiting all of the opening tunnels will take more than $2n$ moves making it impossible to reach the goal in the allotted time. □

Combining Lemmas 41, 37, and 42 we have our dichotomy. Interestingly the classes of gadgets which are hard for LDAGs under shortest-path become the same as those for DAG gadgets.

Theorem 43. *1-player shortest-path motion planning with a k-tunnel LDAG gadget is NP-complete if the gadget has a distant opening or forced distant closing, and otherwise is in P.*

2.3.7 Deterministic Eventually Static Dichotomy

An ***eventually static*** gadget is a gadget whose state graph forms a DAG plus self-loops at the sinks. A special case of LDAG gadgets, these continue to change state until they reach some final form, but unlike DAG gadgets that final form may have traversable tunnels. Between Theorem 38 and Lemma 39 we are left with the case of having a distant opening but no forced distant closing or non-undoable edge.

If we restrict to the deterministic case, then any distant closing must be a forced distant closing as the traversal that permits the tunnel closing must have only one transition and thus taking that transition closes the tunnel. Since any given transition can increase the traversability of other tunnels but never decrease it, traversing tunnels appears to be a good idea. Since we have no non-undoable tunnels, wherever the agent makes a traversal they can always immediately go back across that tunnel. Thus, if an agent is ever able to make a single traversal in a gadget, the agent can make arbitrarily many traversals in that gadget. Since we are looking at eventually static gadgets, after some number of traversals the gadget will eventually end up in a terminal state.

Now the crux for eventually static gadgets will be what their terminal states look like. If a gadget has at least two different terminal states which can each be reached by a series of traversals from some common state, and those two terminal states have different traversability, we say the gadget has ***distinguishable terminal states***. Otherwise the gadget has ***indistinguishable terminal states***.

Lemma 44. *1-player motion planning with any k-tunnel deterministic eventually static gadget with no forced distant closings or non-undoable tunnel and indistinguishable terminal states is in P.*

Proof. First, we will make several observations about eventually static gadgets with no forced

distant closings or non-undoable tunnels. Since no tunnel is non-undoable, if the agent is ever able to make a traversal in a gadget, that agent can continue making traversals along that tunnel. By continuing to do so, the agent is able to bring the gadget into a terminal state. Since the gadget is deterministic and has no forced distant closings, the gadget has no distant closings. If an eventually static has no non-undoable tunnels, then all tunnels in terminal states are undirected. Thus, bringing a gadget from any state to a terminal state will never decrease its traversability and can be done if the agent is ever able to make any traversal in that gadget. Finally, since all terminal states are indistinguishable, the traversability of a gadget in any of its terminal states which were reachable by the agent will be the same.

To solve this problem, the agent first performs a breath-first search looking for the goal location and gadgets in non-terminal states. If the agent finds the goal location, we are done. If an agent finds a traversal of a gadget in a non-terminal state, we have the agent move back and forth along that tunnel until the gadget is in a terminal state. Based on the prior observations, this state change will only help the agent traverse the system of gadgets. We repeat this process until either the goal is found or the agent is unable to reach any state-changing traversals. □

Lemma 45. *1-player motion planning with any k-tunnel LDAG gadget with a distant opening is P-hard.*

Proof. This follows almost directly from Viglietta's Metatheorem 5(a) [60]. We create a loop including the traversal which opens to simulate a button and use the traversal which opens as a door. □

Lemma 46. *1-player motion planning with any k-tunnel deterministic eventually static gadget with no forced distant closings or non-undoable tunnel and distinguishable terminal states is* NP*-hard.*

Proof. First, observe that all tunnels in a terminal state of a eventually static without non-undoable tunnels must be undirected. In addition, if a tunnel is traversable in any direction

from some state s, then all terminal states reachable from s must have that tunnel traversable in both directions. Thus, if the eventually static has distinguished states, there must be a tunnel, call it t, which is closed in one terminal state and open in another terminal state. Now we find states from which there is a series of traversals to both of the distinguishable terminal states. Of these, we want to choose one which does not have a series of transitions to another such state. This state has some transition, call it r, which will force the gadget into a subset of states in which r is not traversable. We now use this transition from s exactly as we would the forced closing path in the proof of NP-hardness for gadgets with distant forced closings, Lemma 37.

Variables are comprised of two pathways, one for setting the variable false and the other for setting the negation of the variable false. These pathways are comprised of a series of gadgets in state s with the r transition oriented forward. Thus the agent must take the r transitions for at least one branch of each variable, ensuring that the corresponding gadgets do not have their tunnels t traversable. Clauses are constructed from the t tunnels of the associated variable gadgets set in parallel. If a gadget was not traversed, it is considered to be set true, and has the possibility of being opened.

Now the last issue is to make sure that t can be opened. Since s admits a series of traversals to both distinguishable states and does not admit a series of traversals to any other state with this property, then there must exist some transition, call it r' from s after which taking enough of any other transition in the gadget will make t traversable. If the tunnel containing r also contains r' in the opposite direction, then we are done. The agent can go forward down one branch of a variable gadget, setting it false, then backtrack down the other, traversing the tunnel until t is open. If r' resides on another tunnel, then we simply put that tunnel immediately before t in our construction of the clause. Since t cannot be traversable in either direction in state s, we know r' cannot reside in t. This covers all of the cases and completes our proof. □

Getting rid of determinism or generalizing to LDAGs make critical parts of the above proof fall apart. However, generally if one can cause the agent to be forced to not open a

traversal or to choose between two different traversals to open, then one can construct a situation akin to a distant closing and prove NP-hardness.

Combining Theorems and Lemmas 38, 39, 44, 45, and 46 we have our dichotomy.

Theorem 47. *1-player motion planning with a k-tunnel deterministic eventually static gadget is* NP*-complete if it has:*

1. *forced distant closing;*
2. *distant opening and non-undoable tunnel; or*
3. *distant opening and distinguishable terminal states.*

Otherwise it is P-complete if it has a distant opening.
Otherwise it is in NL*.*

2.3.8 Planar Hardness for Crossing NAND

We now show a very simple type of LDAG gadget with a distant door closing is still hard in the planar case. A ***NAND*** gadget is a directed 2-tunnel gadget where traversing either tunnel closes both tunnels (preventing all future traversals). A ***crossing door closing*** gadget is a directed 2-tunnel gadget where traversing either tunnel closes the other tunnel. We prove NP-completeness for the NAND gadget below, and the exact same constructions and arguments also work for the crossing door closing gadget.

There are three planar types of NAND gadgets, named by analogy with 2-toggles [25]: one ***crossing*** type (where the two tunnels cross); and two noncrossing types, ***parallel*** (where the directions are the same) and ***antiparallel*** (where the directions are opposite). The notion of NAND gadgets was introduced in [20], which proved NP-hardness using a combination of parallel and antiparallel NAND gadgets, "one-way" gadgets, "fork" gadgets, and "XOR" gadgets. We prove that NAND gadgets alone suffice:

Lemma 48. *1-player planar motion planning is* NP*-hard with either antiparallel NAND gadgets, or crossing NAND gadgets.*

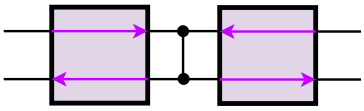


Figure 2-44: Simulation of crossing NAND gadget by antiparallel NAND gadgets.

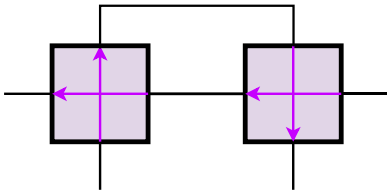


Figure 2-45: Simulation of antiparallel NAND gadget by crossing NAND gadgets.

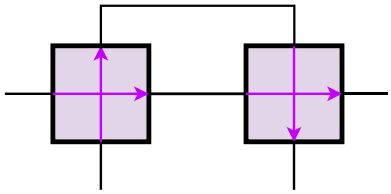


Figure 2-46: Simulation of parallel NAND gadget by crossing NAND gadgets.

Proof. Figures 2-44 and 2-45 show that antiparallel NAND gadgets can simulate crossing NAND gadgets and vice versa. Figure 2-46 shows how crossing NAND gadgets can simulate parallel NAND gadgets. Therefore we can assume the availability of all three planar types of NAND gadgets.

We follow the NP-hardness reduction from Planar 3-Coloring to Push-1-X in [20]. This reduction requires four types of gadgets. Their "NAND gadget" is our parallel and antiparallel (noncrossing) NAND gadgets, which we have. Their "XOR-crossing gadget" is a crossing 2-tunnel gadget that breaks down (in a particular way) if both tunnels get traversed. The reduction guarantees that at most one tunnel in an XOR-crossing gadget will be traversed (because they correspond to different color assignments), so we can replace this gadget with a crossing NAND gadget (which even prevents both tunnels from being traversed). Their "fork gadget" is a one-entrance two-exit gadget such that either traversal closes the other traversal; we can simulate this gadget with a parallel NAND gadget by connecting together the two entrances. Their "one-way gadget" is a gadget that prevents traversal in one direction, but provides no constraint after being traversed in the other direction. Because this gadget is required only to block certain traversals, and each gadget gets visited only once (in particular because the reduction is to Push-1-X where the robot is not permitted to revisit a square), we can replace this gadget with a NAND gadget where one tunnel is not connected to anything. Therefore we have established NP-hardness using only NAND gadgets. □

Since gadgets with a forced distant closing and no distant opening can simulate a single-use gadget, it is simple to construct a NAND gadget from it. If both the construction of

the forced distant closing is planar and connecting it to form the single-use gadget is planar than that gadget will also be hard in the planar case.

Lemma 49. *1-player planar motion planning is* NP*-hard with either antiparallel door closing gadgets or crossing door closing gadgets.*

2.3.9 Deciding DAG Gadget Hardness

It is natural to wonder whether this condition for hardness can be checked in polynomial time. That is, is there a polynomial-time algorithm which determines whether 1-player motion planning with a given DAG gadget is NP-complete? For all of our other dichotomies, the question of whether a gadget of the appropriate type satisfies the condition for hardness is clearly in P; in fact, in L. But a forced distant closing involves an orientation of the tunnels in the gadget, so there may be exponentially many potential forced distant closings to check. We will show that whenever it is necessary to search through each potential forced distant closing, the number of states of the gadget is exponential in the number of tunnels, so the search takes time polynomial in the number of states.

First, it is easy to determine whether a DAG gadget has a distant opening in polynomial time, since we can iterate through the transitions and see whether each one opens another tunnel. So we consider gadgets with no distant openings, and wish to determine whether they have a forced distant closing.

Lemma 50. *Suppose a monotonic k-tunnel DAG gadget has a state s with k open tunnels, and there are no forced distant closings from states reachable from s. Then the gadget has at least 2^k states reachable from s.*

Proof. For each subset of the open tunnels in s, we will find a state that has exactly those tunnels open. Since there are 2^k such subsets, this implies there are at least 2^k states. Assume without loss of generality that each tunnel is traversable from left to right in state s.

Given a subset X of the open tunnels, we perform transitions starting from s as follows. For each tunnel not in X, traverse the tunnel repeatedly until it is closed in both directions;

this must happen eventually because the gadget is a DAG. At each traversal, choose a transition which does not close any other tunnel from left to right. If there were no such choice of transition, that traversal with all other tunnels oriented from left to right would be a forced distant closing, which does not exist by assumption.

After making these transitions, we have closed each tunnel not in X without closing any tunnels in X. Since the gadget is monotonic, we have not reopened any tunnel. So the final state has exactly the tunnels in X open. □

Theorem 51. *Deciding whether a 1-player motion planning with a k-tunnel DAG gadget is NP-complete can be done in polynomial time.*

Proof. The following algorithm checks in polynomial time whether 1-player motion planning with a given a DAG gadget is NP-complete.

- For each transition, see whether it is a distant opening. If it is, accept.
- Iterate through the states of the gadget in reverse order; i.e. check each state reachable from s before checking s. For each state, and for each traversal from that state:
  - Suppose the state has k open tunnels other than the tunnel of the traversal. If every transition corresponding to the traversal leaves fewer than k of these tunnels open, accept.
  - Enumerate the 2^k orientations of these k open tunnels, and check for each orientation whether it is a forced distant closing with the traversal. If it is, accept.
- Reject.

If the gadget has a distant opening, the algorithm notices it in the first step. Otherwise, we check for each state and traversal whether it has a forced distant closing. If every transition for a traversal reduces the number of other open tunnels, than any orientation of the other tunnels gives a forced distant closing. Otherwise, we check for each orientation whether it gives a forced distant closing. So the algorithm accepts exactly when the gadget

has a distant opening or a forced distant closing, which is when 1-player motion planning with the gadget is NP-complete by Theorem 43.

The only step of the algorithm which does not obviously take polynomial time is running through all 2^k orientations of tunnels. Suppose the algorithm reaches this step for some state and traversal. Then there are no forced distant closings after making a transition from this state, since we would have accept already if there were. Also, there is some transition corresponding to the traversal which leaves all k other open tunnels open. By Lemma 50, there are at least 2^k states reachable after making this transition. In particular, the gadget has more than 2^k states, so enumerating the 2^k orientations takes time polynomial in the number of states. Thus the algorithm runs in polynomial time. □

2.4 Reconfiguration and "bounded" gadgets

Reconfiguration can also teach us about reachability problems. In this section we show that LDAG gadgets are actually a special case of a larger class of gadgets whose 1-player reachability problem is in NP. However, this class is defined in part by other gadgets whose reconfiguration problem is also in NP. We also show that two natural classes of gadgets that seem like they might make a problem bounded, monotonically opening and monotonically closing gadgets, actually contain examples which are PSPACE-complete. This is surprising since the change in the traversability of the gadgets is polynomially bounded. The intuition for this construction is closely linked to another result in this section showing there are gadgets which *never* change traversability and yet their 1-player reconfiguration problem is PSPACE-complete.

2.4.1 Generalized DAG Structure

One might wonder whether LDAG gadgets can be further generalized, leading to a larger class of gadgets naturally in NP. In this section, we consider a much wider generalization which we call DAG-like gadgets and find an interesting relationship with reconfiguration

problems.

We call a gadget F-DAG-like if its state graph can be decomposed into disjoint subgraphs for which those subgraphs are from some family of gadgets F and all transitions between these subgraphs form a DAG. We call these transitions between the subgraphs ***DAG-like transitions***. In this case LDAG gadgets are F-DAG-like with some family of single state gadgets.

With this notion, one may wonder what gadgets can be used in an F-DAG-like gadget and have the resulting gadget still be in NP. We initially believed this would be true for gadgets with non-interacting tunnels, however, below we give an example of such a gadget where the reachability question is PSPACE-complete. We then show that if F is a family of gadgets for which the reconfiguration problem is in NP, then the reconfiguration and reachability problems for F and for F-DAG-like gadgets are also in NP. We will call these NPReDAG gadgets. Since the reconfiguration problem for single state gadgets is trivial, this gives a more general description of gadgets for which the motion planning problem is in NP.

Theorem 52. *1-player reconfiguration motion planning with NPReDAG gadgets is in NP.*

Proof. We give the following certificate for 1-player motion planning with an NPReDAG gadget. We list all of the DAG-like transitions taken in the solution and the states of all of the gadgets before and after the transition. Further, for each pair of adjacent DAG-like transitions we imagine the reconfiguration problem on the system of gadgets which is only comprised by the reconfigurable super-node gadgets and takes this system from the state after the last DAG-like transition to the state before the next DAG-like transition. This problem is solvable in NP by definition, so we provide each of these certificates. The verifier can now check in polynomial time that the final state is the target state, that the polynomial many DAG-like transitions are valid transitions and take the given pre-transition state to the post-transition state, and that the (polynomially many) portions of the path between the DAG-like transitions have some valid path performing that reconfiguration. □

Theorem 53. *1-player reachability motion planning is in NP for gadgets where 1-player reconfiguration motion planning is in NP.*

Proof. We now essentially want to "guess" the final configuration that the system will be in when the agent solves the reconfiguration problem and then solve the reconfiguration problem. However, this strategy also needs to verify that the agent actually reaches the target location. To do this first we take the 1-player reachability instance and add at the target location a loop with a gadget that has access to a transition with a state change. If there is none, then both the reconfiguration and reachability problems are trivially in NL. To be able to change the state of the added gadget, an agent must have reached the location of the loop. Thus we will take as a certificate a final configuration of the system of gadgets which has the added gadget in a different state, as well as the certificate for the reconfiguration problem from the initial state to this new target state. □

Corollary 54. *1-player reachability motion planning with NPReDAG gadgets is in NP.*

2.4.2 Verified Gadgets and Shadow Gadgets

In this section we will discuss a technique for generating hard gadgets. The main idea is constructing a gadget which behaves well when used like a hard gadget, but might also have other transitions which are allowed but put the gadget into some undesirable state.

First, we will pick some base gadget which we want to modify. Next we will add additional ***shadow states*** to the gadget and additional transitions with the restriction that all newly added transitions must take the gadget to a shadow state. We call such a construction a ***shadow gadget*** of the base gadget. This has the nice property that if the agent takes any transition that would not be allowed in the base gadget, then the gadget will always stay in a shadow state after that point.

Theorem 55. *1-player reconfiguration motion planning with a shadow gadget is at least as hard as 1-player reconfiguration motion planning with the base gadget.*

Proof. We simply take the hard instance for the problem with the base gadget and replace it with a shadow gadget. If a solution exists in the initial instance, it will still be a solution with the shadow gadgets. If the agent ever tries to take a transition in a shadow gadget that

would not have been allowed in the original instance, that gadget will now be in a shadow state. Since no target state is a shadow state and all transitions from shadow states lead to shadow states such a path cannot be a solution. □

Corollary 56. *1-player reconfiguration motion planning is* PSPACE*-complete for some unchanging gadgets.*

Proof. Recall an unchanging gadget is one in which the gadget's traversability never changes. Now, take a gadget for which 1-player reconfiguration motion planning is PSPACE-complete, such as the 2-toggle. Now construct a shadow gadget with one shadow state that has transitions between all of the locations. Add transitions starting from every state and location and going to every other location and the shadow state. The resulting gadget always has available traversals from every location to every other location and thus never changes traversability. However, the reconfiguration problem is hard by Theorem 55. □

A ***verified gadget*** is a shadow gadget with some additional structure. From a shadow gadget we add two or more locations, the ***verifying locations*** to the gadget. We additionally may add ***verified states*** which can only be reached by transitions from the added locations while the gadget is in normal states. We now add transitions among the verifying locations such that these locations can be connected in a series so there is always a traversal from the first to the last location if the gadget is in a normal state, and there is no such traversal if the gadget is in a shadow state. We call this added traversal the ***verification traversal***.

Theorem 57. *1-player reachability motion planning with a verified gadget is at least as hard as 1-player reachability motion planning with the base gadget.*

Proof. We simply take the hard instance for the problem with the base gadget and replace it with a verified gadget and then make a path from the original target location through the verification traversals of all of the gadgets to a new target location. If a solution exists in the initial instance, then performing that solution will bring the agent to the start of the verification traversals and all of those traversals will be possible since the gadgets are all in

normal states. If the agent ever tries to take a transition in a verifiable gadget that would not have been allowed in the original instance, that gadget will now be in a shadow state and at least one of the necessary verifiable traversals will now be possible. □

Monotonically Opening and Closing Gadgets. A ***monotonically opening*** gadget is one in which the traversability of the gadget never decreases. That is to say for all states t reachable from a a given state s, and for all pairs of locations a and b, if there is a transition from a to b in s then there is a transition from a to b in t. A ***monotonically closing*** gadget is one in which the traversability of the gadget never increases. That is to say for all states t reachable from a a given state s, and for all pairs of locations a and b, if there is not a transition from a to b in s then there is not a transition from a to b in t.

We now use verified gadgets to show that there are both monotonically opening and monotonically closing gadgets for which reachability is PSPACE-complete. This is surprising because the number of changes of traversability in such a system of gadgets is bounded, so one might suspect such a class to fall in NP.

Corollary 58. *1-player reachability motion planning is* PSPACE*-complete even for monotonically closing gadgets.*

Proof. Take a gadget for which 1-player reconfiguration motion planning is PSPACE-complete, such as the 2-toggle. Now construct a shadow gadget with one shadow state that has transitions between all of the locations. Add transitions starting from every state and location and going to every other location and the shadow state. The resulting gadget always has available traversals from every location to every other location and thus never changes traversability. Next, we convert it into a verified gadget by adding a pair of locations A and B where there is a traversal between them if the gadget is in a normal state and no traversal if it is in a shadow state. This gadget now only removes traversals, but by Theorem 57 its reachability problem is PSPACE-complete. □

Corollary 59. *1-player reachability motion planning is* PSPACE*-complete even for monotonically opening gadgets.*

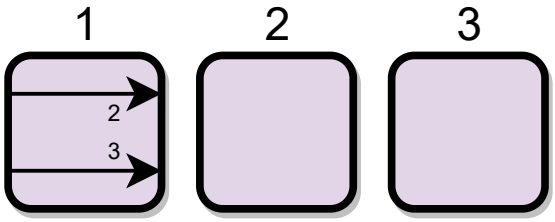


Figure 2-47: The Labeled Two-Tunnel Single-Use gadget.

Proof. Take a gadget for which 1-player reconfiguration motion planning is PSPACE-complete, such as the 2-toggle. Now construct a shadow gadget with one shadow state that has transitions between all of the locations. Add transitions starting from every state and location and going to every other location and the shadow state. The resulting gadget always has available traversals from every location to every other location and thus never changes traversability. Next, we convert it into a verified gadget by adding two pairs of locations A, B and C, D. There is a transition between C and D only if the gadget is in the verified state. There is additionally a transition between A and B from all normal states to the verified state, and also transitions between them from shadow states to shadow states. This gadget now only adds the traversal between C and D and never removes traversals, but by Theorem 57 its reachability problem is PSPACE-complete. □

2.4.3 Reconfiguration Can Be Easier

In this section we introduce the Labeled Two-Tunnel Single-Use gadget for which the reachability question is harder than the reconfiguration problem. Figure 2-47 shows the Labeled Two-Tunnel Single-Use gadget. It is a DAG gadget where going through either tunnel closes both of them; however, the states are distinguished based on which tunnel was traversed. This is a DAG gadget with a forced distant door closing, so it is NP-complete by Theorem 22 in [29]. We now give a polynomial time algorithm for the reconfiguration problem.

Theorem 60. *1-player reconfiguration motion planning with the Labeled Two-Tunnel Single-Use gadget is in P.*

Proof. We will call states 2 and 3 ***terminal states***. Now let us consider what the initial and

final configurations of the gadgets can look like. If the initial state is terminal, the gadget cannot be traversed. Similarly, if the initial and final configuration are both state 1, then the gadget cannot have been traversed since there is no way to return the gadget to state 1 after traversal. Thus the only case we need to consider is starting in state 1 and ending in a terminal state. In this case, the labeling of the target configuration tells us which of the two tunnels must have been traversed to reach that state. We can thus construct the graph which uses only those tunnels and ask whether there is a path which traverses them all exactly once. Since this is exactly checking for the existence of an Eulerian path in a graph, we can solve it in polynomial time.. □

It remains an interesting open question to exhibit a gadget for which reachability is PSPACE-complete but reconfiguration is in P. It would also be interesting to have an example of a gadget which has a different traversability in every state so that the easiness of such a reconfiguration problem would not be using a degeneracy which is indistinguishable in the reachability problem.

2.5 One-Player Input Output

In this section, we consider one-player motion planning with input/output gadgets. This is a generalization of zero-player motion planning, where we no longer require each connected component of the connection graph to have only one input location. We also now allow nondeterministic gadgets. This section is taken primarily from [9], done in collaboration with Joshua Ani, Erik Demaine, and Dylan Hendrickson.

A simple nondeterministic input/output gadget is the ***directed branching hallway***, which has one input location, two output locations, and one state; the player may choose which output location to take. One-player motion planning (with input/output gadgets) can be equivalently defined by introducing the branching hallway to zero-player motion planning, instead of removing the constraint that the system is branchless.

We can characterize the complexity of one-player motion planning with an output-disjoint

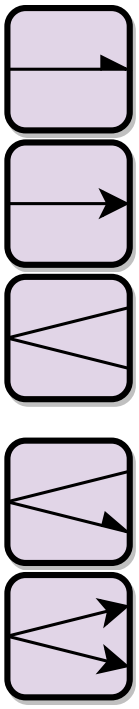
deterministic 2-state input/output gadget: if the gadget is unchanging, one-player motion planning is just reachability in a directed graph which is NL-complete. If the gadget is bounded, one-player motion planning is in NP, and we will prove it is NP-complete as a corollary to Theorem 67 which investigates single-input gadgets. If the gadget is unbounded, one-player motion planning is PSPACE-complete because it is a generalization of zero-player motion planning. See Section 4.3 for that proof. Note, it is not generally the case that 1-player motion planning has a trivial reduction from 0-player motion planning, because even in a branchless system of deterministic gadgets the agent might decide to reverse direction and enter the location it just exited. However, input/output gadgets never have transitions starting in their output nodes and thus any 0-player construction using them will never give the agent a different option of what to do.

Section 2.5.1 gives a classification of 2-state input/output output-disjoint gadgets; however this section mostly focuses on single-input input/output gadgets.

One-player reachability switching games, studied in [34], are equivalent to one-player motion planning with deterministic single-input input/output gadgets. It is shown in [34] that this problem is NP-complete when the gadgets are described as part of the instance.

In this section, we improve this result in two ways. First, we show in Theorem 62 that the problem remains in NP even when we allow nondeterministic single-input input/output gadgets, which can not all obviously be simulated by deterministic gadgets. Our proof is similar to the proof of containment in NP in [34].

Second, we show in Section 2.5.3 that the problem remains NP-hard with a specific gadget instead of instance-specified gadgets. In particular, we show that 1-player motion planning with the toggle switch or the set switch is NP-complete. Our reduction is simpler than the one in [34], and the technique can easily be used to prove NP-hardness for many other single-input input/output gadgets.



Set-Up Line A tunnel that can always be traversed in one direction and sets the state of the gadget to a specific state.

Toggle Line A tunnel that can always be traversed in one direction and toggles the state with each crossing.

Switch A three-location gadget with one input which transitions to one of two outputs depending on the state, without changing the state.

Set-Up Switch A switch which also sets the state of the gadget to a specific state.

Toggle Switch A switch which also toggles the state of the gadget with each crossing.

2.5.1 Classifying Output-Disjoint Deterministic 2-State Input/Output Gadgets

In this Section, we are primarily interested in output-disjoint deterministic 2-state input/output gadgets. In this section, we omit the adjectives and refer to them simply as "gadgets", and give a categorization of these gadgets, into "trivial," "bounded," and "unbounded" gadgets. For each category, we will show that every gadget in the category can simulate at least one of a finite set of gadgets. The behavior of an input location to a gadget is described by how it changes the state and which output location it sends the agent to in each state. If the input location does not change the state and always uses the same output location, it can be ignored (the path can be "shortcut" to skip that transition). Otherwise, the input location corresponds to one of the following five nontrivial subunits, and the gadget is a disjoint union of some of these subunits (which interact by sharing state):

The ARRIVAL problem [33] is equivalent to zero-player motion planning with the toggle switch: we replace each vertex in their switch graph with a toggle switch, or vice versa. More generally, zero-player motion planning with an arbitrary set of deterministic single-

input input/output gadgets (with gadgets specified as part of the instance) is equivalent to explicit zero-player reachability switching games, as defined in [34].

We call the states of any such two state gadget ***up*** and ***down***, and assume that each switch transitions to the top output in the up state and the bottom output in the down state; because we are not concerned with planarity, this assumption is fully general by possible reflection of each subunit. There are two versions of the set line and set switch: one to set the gadget to each state. For example, any gadget with a set-up line and set-down switch is meaningfully different from a set-up line and set-up switch. We draw the set-down line and switch as the reflections of the set-up version above. To represent the current state of a gadget, we make one of the lines in each switch dashed, so that the next transition would be made along a solid line.

We categorize gadgets into three families:

1. ***Trivial*** gadgets have either no state change or no state-dependent behavior; they are composed entirely of either switches or toggle and set lines. They are equivalent to collections of simple tunnels, and zero-player motion planning with them is in L by straightforwardly simulating the robot.

2. ***Bounded*** gadgets have state-dependent behavior (i.e., some kind of switch) and one-way state change, either only to the up state or only to the down state. They naturally give rise to bounded games, because each gadget can change its state at most once.

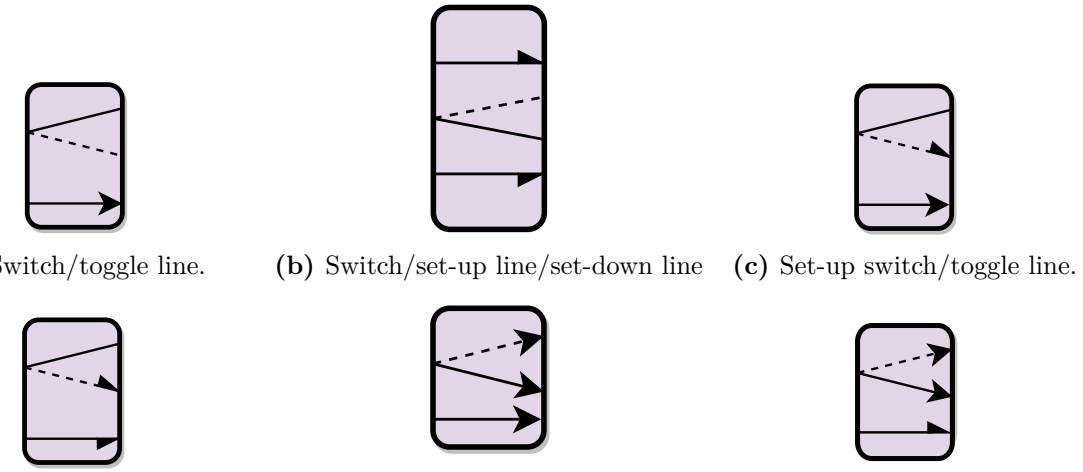
3. ***Unbounded*** gadgets have state-dependent behavior and can change state in both directions. They naturally give rise to unbounded games.

We will find that the complexity of a gadget also depends on whether it is ***single-input***, meaning it has only one input location, or multiple nontrivial inputs. The only nontrivial single-input gadgets are the set switch and toggle switch, which are bounded and unbounded, respectively.

To characterize all non-trivial, multi-input gadgets we show that they all simulate at least one of the eight gadgets listed in Lemma 61 and shown in Figures 2-48 (bounded) and

(a) Switch/set-up line. **(b)** Set-up switch/set-up line.

Figure 2-48: A "basis" for the bounded multi-input gadgets



(a) Switch/toggle line. **(b)** Switch/set-up line/set-down line **(c)** Set-up switch/toggle line.

(d) Set-up switch/set-down line. **(e)** Toggle switch/toggle line. **(f)** Toggle switch/set-up line.

Figure 2-49: A "basis" for the unbounded multi-input gadgets

2-49 (unbounded), and thus it will suffice to show hardness for these eight cases.

Lemma 61. *Let G be an output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs.*

- *If G is bounded, then it simulates either a switch/set-up line or a set-up switch/set-up line.*
- *If G is unbounded, then it simulates one of the following gadgets:*
  1. *switch/toggle line,*
  2. *switch/set-up line/set-down line,*
  3. *set-up switch/toggle line,*
  4. *set-up switch/set-down line,*

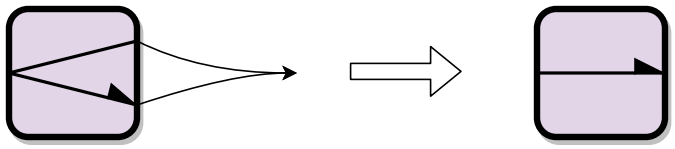


Figure 2-50: Joining the outputs of a set-up switch yields a set-up line.

5. *toggle switch/toggle line, or*

6. *toggle switch/set-up line.*

Proof. We first merge the two outputs of (***compress***) every switch, set switch, and toggle switch, except for one. This replaces set switches with set lines, toggle switches with toggle lines, and ordinary switches with trivial lines. For an example, see Figure 2-50. If the gadget has any ordinary switches, we use one of them as the switch that does not get compressed. The resulting gadget has the same boundedness as the original gadget, has a single switch of some type, and still has multiple nontrivial inputs: if it had only one nontrivial input, then the other inputs must have all been ordinary switches which got compressed, so the remaining uncompressed input is also an ordinary switch, and thus the original gadget contained only ordinary switches and was trivial.

For multi-input bounded gadgets, we now have either a switch or a set switch (any sort of toggle would make the gadget unbounded), and at least one set line. Each set switch and line must set the gadget to the same state (which we can assume is the up state), and we can ignore all but one set line. In particular, without loss of generality the resulting gadget contains exactly a set-up line and either a switch (2-48a) or a set-up switch (2-48b).

For multi-input unbounded gadgets, there are multiple cases to consider based on the type of the single switch which was not compressed. First, if the switch is an ordinary switch, there must be lines that can set the state in both directions, which must include either a toggle line (2-49a) or two set lines in different directions (2-49b). If the switch is a set switch, there must be a line that can set the state in the opposite direction, which can be either a toggle line (2-49c) or a set line opposite the set switch (2-49d). Finally, if the switch is a toggle switch, there must be some nontrivial line: either a toggle line (2-49e) or a set line (2-49f). We have made arbitrary choices for the directions of set lines and set switches; these

are without loss of generality because we can reflect the gadget (or rename the "up" and "down" states). □

2.5.2 Containment in NP

We first show that one-player motion planning with any single-input input/output gadget is in NP, generalizing a result from [34]. Our proof is similar, but requires more care to account for nondeterministic gadgets.

Theorem 62. *One-player motion planning with any single-input input/output gadget is in NP.*

Proof. A single-input input/output gadget G is described by a directed graph with states as vertices and transitions as edges, where each edge is labeled with an output location. An edge labeled ℓ from s to s' indicates that when the robot enters the unique input location in state s, it can exit at ℓ and change the state to s'. This can equivalently be thought of as a NFA on the alphabet of locations.

We will adapt the certificates used in [34], controlled switching flows, to work for nondeterministic gadgets. The number of times each output location (or edge in the equivalent reachability switching game) is used is no longer enough information, since it may in general be hard to determine whether a nondeterministic gadget has a legal sequence of transitions which uses each location a specified number of times.[1] Instead, we will have the certificate include the number of times each ***traversal*** in each gadget is used, which will be enough information to be checked quickly. We modify the definition of controlled switching flows as follows.

Definition 63. *A **controlled switching flow** in a system of G is a function f from the set of transitions in copies of G to the natural numbers (including zero) which is "locally consistent" in the following sense:*

[1] In fact, this is NP-hard by a reduction from the existence of a Hamiltonian path in a directed graph: given a graph with n vertices, construct a gadget with n states and n output locations whose transition graph is the input graph, and ask for a sequence of transitions which uses each output location exactly once.

- *For a connected component H of the connection graph, let H_i and H_o be the sets of traversals from input locations and to output locations in H, respectively. That is, H_i contains all transitions in gadgets whose input location is in H, and H_o contains the transitions which leave the robot in H. Then*

$$\sum_{t \in H_i} f(t) - \sum_{t \in H_o} f(t) = \begin{cases} 1 & H \text{ contains the start location} \\ -1 & H \text{ contains the goal location} \\ 0 & \text{otherwise.} \end{cases}$$

- *For each gadget, there is a legal sequence of transitions from its starting state s which uses each transition t in the gadget exactly $f(t)$ times.*

That is, thinking of $f(t)$ as the number of times the robot uses the transition t, the robot enters and exits each connected component the same number of times, except that it exits the start location and enters the goal location once, and the robot uses the transitions of each gadget a consistent number of times.

To prove containment in NP, our certificate that it is possible to reach the goal location is a controlled switching flow. We need the following three lemmas:

Lemma 64. *If there is a controlled switching flow, then it is possible to reach the goal location.*

Lemma 65. *If it is possible to reach the goal location, then there is a polynomial-length controlled switching flow, i.e., one where $f(t)$ is at most exponential in the size of the system.*

Lemma 66. *There is a polynomial-time algorithm which determines whether a function f is a controlled switching flow.*

Together these imply that controlled switching flows can actually be used as certificates, and thus the one-player problem is in NP.

Proof of Lemma 64. Let f be a controlled switching flow. For each copy g of G, pick a sequence of transitions of length $\ell_g = \sum_{t \in g} f(t)$ in that copy which uses each transition t exactly $f(t)$ times; this exists by the definition of a controlled switching flow. We play the one-player motion planning game in the system. Our strategy is based on the chosen sequences: whenever we arrive at a gadget, take the next transition in the sequence. If we find ourselves in a connected component with the input locations of multiple gadgets, we can enter any gadget g which we have previously used fewer than ℓ_g times. We stop when we reach the connected component of the goal location, or when we have no moves obeying this strategy, meaning every gadget g whose input location is currently reachable has already been used ℓ_g times.

We claim this strategy must reach the goal location. If it does not, we must eventually get stuck with no moves (specifically, within $\sum_t f(t)$ steps), and we will show this can not happen because f is a controlled switching flow. For the sake of contradiction, let H be the connected component of the connection graph we are stuck in. To be stuck, we must have previously exited H at least $\sum_{t \in H_i}$ times. So we must have entered H at least $\sum_{t \in H_i} f(t)+1$ times (or one fewer, if the start location is in H). However, we have entered H at most $\sum_{t \in H_o} f(t)$ times, so $\sum_{t \in H_o} f(t) \geq \sum_{t \in H_i} f(t) + 1$, which violates the assumption that f is a controlled switching flow. □

Proof of Lemma 65. For some path which reaches the goal location, let $f(t)$ be the number of times the path uses the traversal t. Then f is clearly a controlled switching flow. The number of traversals in the shortest solution path is at most the number of configurations of the system of gadgets, which is at most nk^n if G has k states and there are n copies of G. Thus using the shortest solution path, we have a controlled switching flow f where $f(t) \leq nk^n$ and thus f has polynomial length. □

Proof of Lemma 66. Think of G as a directed graph with locations as vertices and transitions as edges. The first condition on controlled switching flows says that there is a path through this graph starting at s which uses each edge t a specified number $f(t)$ of times. This is equivalent to an Euler path in the (possible exponentially large) graph with $f(t)$ copies of

the edge t. To verify that such a path exists, we only need to check that the total in- and out-degrees match at each vertex (except possibly off by one at s and one other vertex) and that the set of used transitions, i.e., those t where $f(t) > 0$, is connected. This can all be checked in polynomial time.

The second condition can also be easily checked in polynomial time by computing the relevant sums. □

□

2.5.3 NP-hardness

In this section, we prove NP-hardness of one-player motion planning with each nontrivial single-input 2-state deterministic gadget (the set switch and toggle switch). The proofs used can be easily adapted to prove NP-hardness of the corresponding problem for many input/output gadgets, but we leave open the problem of providing a characterization.

Recall the set switch, shown in Figure 2-51, is a bounded, deterministic, 2-state, 3-location, single-input, input/output gadget. In state 1 the agent goes to one output and flips the state, where-after the agent will continue to to to the other location. The toggle switch, shown in Figure 2-52, is an bounded, deterministic, 2-state, 3-location, single-input, input/output gadget. Each state has a transition to each of the outputs, and those transitions also flip the state.

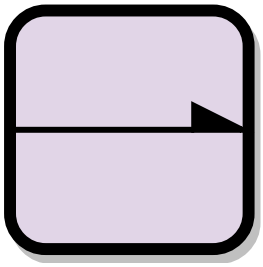


Figure 2-51: The set switch gadget.

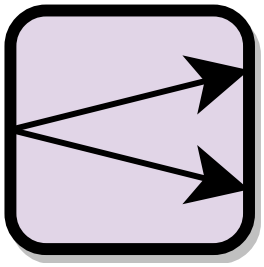


Figure 2-52: The toggle switch gadget

Our reduction is simpler than that in [34], and we show hardness for specific gadgets instead of general reachability switching games, which are equivalent to instance-specified

gadgets.

Theorem 67. *One-player motion planning with each of the toggle switch and the set switch is NP-hard.*

Proof. We provide essentially identical reductions from 3SAT to the two motion-planning problems. In the reduction, the player will never be able to traverse a gadget more than two times, so the difference between the toggle switch and the set switch is irrelevant. Each gadget will begin in the state which sends the robot to the "top" exit, and after a single traversal moves to the state which sends the robot to the "bottom" exit. We will describe the reduction in terms of the set-down switch, but it is equivalent for the toggle switch.

First, we build a single-use tunnel, which is a set-down switch where the bottom exit leads nowhere. The robot can pass through the single-use tunnel once and exit at the top, but traversing it again makes the robot stuck.

For each variable in a 3SAT instance, there is a fork where the player may choose one of two paths. Each path passes through a series of set-down switches, exiting each from the top and setting them to the down state. The paths then merge and then go through a single-use tunnel to arrive at the fork corresponding to the next variable. The robot starts at the first fork, so the beginning of the motion-planning game has the player pick a branch on each fork to traverse, corresponding to an assignment to the 3SAT instance. The number of gadgets in each branch depends on the number of instances of each literal in the formula.

For each clause, there is a 3-way fork, where the player must choose to go through one of the gadgets corresponding to a literal in the clause. If the chosen gadget was already traversed, the robot exits the bottom and proceeds to the next clause. Otherwise, the robot follows the path which goes through that gadget corresponding to a variable choice. At the end of this path, the robot gets stuck, since the only way forward is a single-use path which was traversed during the variable-setting phase. The clauses are connected in series so that in order to reach the goal location, the robot must pass through each variable and then each clause. In order to get through a clause without getting stuck, at least one gadget in the clause must have already been traversed; equivalently, at least one literal in the clause must

be true under the assignment corresponding to the path taken during variable setting. Thus the robot can reach the goal location if and only if the formula has a satisfying assignment. □

2.6 1-player Door Gadgets

These results come primarily from [7] written in collaboration with Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, and Dylan Hendrickson.

In this section, we develop analyze a special case of gadgets called ***door*** gadgets. This can be seen as a formalization of the "door gadget" proof technique used in [5,16,32]. In all cases, a door gadget has two states and three disjoint traversal paths: "traverse", "close", and "open". Each path may be individually ***directed*** (traversable in one direction) or ***undirected*** (traversable in both directions). In addition, the open traversal path may have identical entrance and exit locations, meaning that its traversal changes the door's state but does not move the agent (breaking the k-tunnel assumption). In this way, we can require that traversing the open and close traversal paths force the door's state to open and closed, respectively, but still effectively allow the player to make a choice of whether to open the door (by skipping or including the open traversal, which leaves the agent in the same location either way).

In Section 2.7, we prove that every such door gadget is ***universal***, meaning that any one of them can simulate ***all*** gadgets in the motion-planning-through-gadgets framework of [25,29]. This is comparable to the results in Section 4.3.3 which show certain input/output gadgets can simulate any other input/output gadget in the 0-player model. This result provides the first examples of fully universal gadgets. Further, whenever we prove a gadget is able to simulate a door gadget, we get a self-simulation result similar to the one in Section 2.2.6 for 2-state 2-tunnel reversible deterministic gadgets.

As a consequence of universal simulation, we obtain that 1-player motion planning with any door gadget is PSPACE-hard, but because the simulation is nonplanar, it does not tell us anything about planar motion planning where the gadgets are connected in a planar graph.

In Section 2.8, we introduce two more families of door gadgets. A ***self-closing door*** has two states but only two traversal paths: "open" and "self-close". The self-close traversal is possible only in the open state, and it forcibly changes the state to closed. As before, each

traversal path can be either directed or undirected; and the open traversal forces the state to open, but we allow the open traversal path to have identical start and end locations, which effectively allows optional opening. A ***symmetric self-closing door*** has two states and two traversal paths: "self-open" and "self-close". The self-open/close traversal is possibly only in the closed/open state, respectively, and it forcibly changes the state to open/closed, respectively. (This definition is fully symmetric between "open" and "close".) Each traversal path can be either directed or undirected, but we no longer allow optional traversal.

In Section 2.9, we prove that ***planar*** 1-player motion planning is PSPACE-complete for every door gadget, for every local combinatorial planar embedding of every type door gadget except for one (which we only prove NP-hard). Thus, all that is needed to prove a new game PSPACE-hard is to construct any single supported door gadget, and to show how to connect the door entrances/exits together in a planar graph. In particular, the crossover gadgets previously constructed for Lemmings [16, Figure 2(e)], Legend of Zelda: Link to the Past and Donkey Kong Country 1, 2, and 3 [5, Figures 28 and 20], and Super Mario Bros. [5, Figure 5] are no longer necessary for those PSPACE-hardness proofs: they can now be omitted. (See Section 5 for details.) Our result should therefore make it easier in the future to prove 2D games PSPACE-hard. Because of their reduced conceptual complexity — only two traversal paths, which behave essentially identically for symmetric self-closing doors — we have found it even easier to prove games PSPACE-hard by building self-closing door gadgets.

2.7 Doors

In this section, we adapt the door framework of [5, Section 2.2] (a cleaner presentation of the framework from [16]) into the motion-planning-through-gadgets framework.

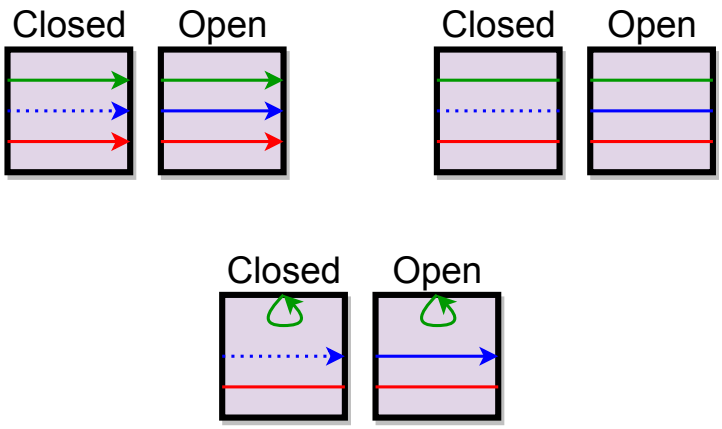


Figure 2-53: Left: A directed open-required door. Right: An undirected open-required door. Bottom: A mixed open-optional door.

2.7.1 Terminology

We define a ***door*** to be a gadget with an ***opening*** port or tunnel, a ***traverse*** tunnel, and a ***closing*** tunnel, and each of the tunnels may be directed or undirected. The opening port/tunnel opens the traverse tunnel, and the closing tunnel closes the traverse tunnel. Throughout this Section, the opening port/tunnel will be colored green, the traverse tunnel will be colored blue, and the closing tunnel will be colored red. In addition, a solid traverse tunnel represents an open door, and a dotted traverse tunnel represents a closed door. A ***directed door*** is a door where all tunnels are directed. An ***undirected door*** is a door where all tunnels are undirected. A door that is neither undirected nor directed is a ***mixed door***. An ***open-required door*** is a door with an opening tunnel, and an ***open-optional door*** is one with an opening port. A directed open-required door, an undirected open-required door, and a mixed open-optional door are shown in Figure 2-53.

2.7.2 Hardness

In this section we give a series of simple reductions to show all versions of open-optional, directed, and mixed directed doors can simulate a fully directed door.

Theorem 68. *In 1-player motion planning, any door can simulate its corresponding open-optional door.*

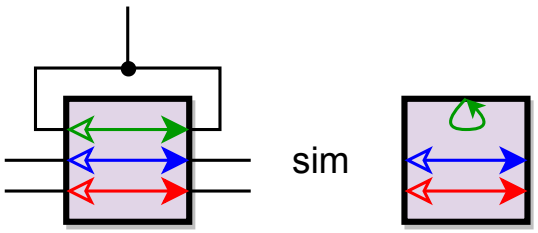


Figure 2-54: An open-required door simulates its corresponding open-optional door. Outlined arrows indicate optionally allowed traversals.

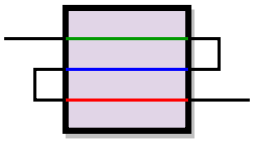


Figure 2-55: Simulation of a diode with an undirected door.

Proof. In case of a door that is not already open-optional, we wire one end of the open tunnel to the other end and wire some point on this loop externally as shown in Figure 2-54. This turns the open tunnel into an open port. □

Theorem 69. *1-player motion planning with any directed door is* PSPACE*-hard.*

Proof. The directed open-optional door trivially simulates the door of [5], and by Theorem 68, the directed open-required door simulates the directed open-optional door. This covers all cases. □

Theorem 70. *1-player motion planning with any mixed door is* PSPACE*-hard.*

Proof. Let D be a mixed door. Then D has a directed tunnel. No tunnel changes its own traversability when crossed, so this tunnel simulates a diode[2]. We wire each tunnel of D through a diode at each end, simulating a directed door, which 1-player motion planning is PSPACE-hard for. □

Theorem 71. *1-player motion planning with any undirected door is* PSPACE*-hard.*

Proof. Let D be an undirected door. To simulate a diode, we wire a path through the opening port/tunnel, then through the traverse tunnel, then through the closing tunnel, as in Figure 2-55. The player can open the traverse tunnel, traverse the traverse tunnel, then close the traverse tunnel. However, if the player tries to go the other way, they will close the traverse tunnel and be unable to continue. Thus, this simulates a diode. As in the proof for

[2]A diode is a 1-tunnel 1-state gadget that consists of just a directed tunnel.

Theorem 70, we wire each tunnel of D through a diode at each end, simulating a directed door. □

2.7.3 Universality

Here, we state and prove an interesting theorem regarding gadget simulations. We show that any gadget can be simulated by door gadgets. At a high level, the reduction does the following:

- Creates a door for each location, each state/location pair, and each transition in the simulated gadget.
- When not inside the gadget, the state/location doors corresponding to the current state of the gadget will be open, the others will be closed.
- When not inside the gadget, the doors corresponding to the transitions and the doors corresponding to the locations are always closed. These gadgets have their traverse tunnel directly connected to their close tunnel, making them self-closing doors.
- The traverse tunnel of the state/location doors gives access to the open tunnels of the transition doors which are accessible from that state/location pair. Each of these leads to the open tunnel of the corresponding location door.
- After traversing the open tunnel of any location door one must cross the close tunnels of all state/location doors.
- The traverse and close tunnel of the transition door gives access to the open tunnels of the state/location doors of the state being transitioned to.
- Finally, the transition and close tunnel of the location door can be traversed, ending at the target location.

Theorem 72. *The open-required directed door can simulate any gadget.*

Proof. Consider an arbitrary gadget G, with a set P of ports, a set S of states, and a set of allowed traversals between the ports and states. For each port p and state s, we add an open-required directed door $D_{s,p}$. For each port p, we add an open-required directed door D_p. For each traversal $(s_0, p_0) \to (s_1, p_1)$, we add an open-required directed door $D_{s_0,p_0 \to s_1,p_1}$. Finally, for each port p, let E_p be an external port of the simulation. We map each state s to the state in the simulation where for all ports p, all $D_{s,p}$ are open and all other doors are closed.

For each directed door D, let $O_0(D)$, $O_1(D)$, $T_0(D)$, $T_1(D)$, $C_0(D)$, and $C_1(D)$ be the opening tunnel input, opening tunnel output, traverse tunnel input, traverse tunnel output, closing tunnel input, and closing tunnel output of D respectively. For each port p and state s, connect E_p to $T_0(D_{s,p})$. For each traversal $(s_0, p_0) \to (s_1, p_1)$, connect $T_1(D_{s_0,p_0})$ to $O_0(D_{s_0,p_0 \to s_1,p_1})$, and connect $O_1(D_{s_0,p_0 \to s_1,p_1})$ to $O_0(D_{p_1})$. Order $S \times P$. Let (s_f, p_f) be the first element of $S \times P$, (s_l, p_l) be the last element, and $\text{next}(s, p)$ be the element directly after (s, p). Also order P, so that p_f is the first element of P, p_l is the last element, and $\text{next}(p)$ is the element directly after p. Then for each port p, connect $O_1(D_p)$ to $C_0(D_{s_f,p_f})$. For each port p and state s where $(s, p) \neq (s_l, p_l)$, connect $C_1(D_{s,p})$ to $C_0(D_{\text{next}(s,p)})$. For each traversal $(s_0, p_0) \to (s_1, p_1)$, connect $C_1(D_{s_l,p_l})$ to $T_0(D_{s_0,p_0 \to s_1,p_1})$, connect $T_1(D_{s_0,p_0 \to s_1,p_1})$ to $C_0(D_{s_0,p_0 \to s_1,p_1})$, and connect $C_1(D_{s_0,p_0 \to s_1,p_1})$ to $O_0(D_{s_1,p_f})$. For each state s and port $p \neq p_l$, connect $O_1(D_{s,p})$ to $O_0(D_{s,\text{next}(p)})$. For each state s_i and port p, connect $O_1(D_{s,p_l})$ to $T_0(D_p)$. Finally, for each port p, connect $T_1(D_p)$ to $C_0(D_p)$ and connect $C_1(D_p)$ to E_p. Figure 2-56 shows an example.

Assume the contraption is in the state mapped to by some state $s \in S$ and the agent tries to enter in port E_p for some $p \in P$. The agent must traverse $D_{s,p}$. Consider an arbitrary pair of state-location tuples $((s, p), (s', p'))$ in G. If $(s, p) \to (s', p')$ is allowed in the transitive closure[3] of G, then the agent can open $D_{s,p \to s',p'}$, and then must open $D_{p'}$, then close $D_{i,j}$ for all $i \in S$ and $j \in P$, then traverse and close $D_{s,p \to s',p'}$, then open $D_{s',j}$ for all $j \in P$, then

[3]The transitive closure of a gadget is a new gadget with transitions for every state/location pair that can be reached from any other state/location pair. One can think of this as the gadget defined by the transitive closure of the state-location graph of the gadget.

traverse and close $D_{p'}$. The end result is that $D_{s',j}$ is open for all $j \in P$, all other doors are closed, and the agent is at port $E_{p'}$. So this simulates the traversal $(s, p) \to (s', p')$. Note that this means the contraption will be in a state that maps to a state of G if the agent is outside the contraption or at an external port.

If the agent can traverse from E_p to $E_{p'}$ for some $p, p' \in P$. The contraption must start in a state that s maps to and end in a state that s' maps to, for some $s, s' \in S$. The only ways to make this transition traverse doors $D_{s,p\to s_1,p_1}, \cdots, D_{s_k,p_k\to s',p'}$ in order, where $((s, p) \to (s_1, p_1)), \cdots, ((s_k, p_k) \to (s', p'))$ is a sequence of allowed traversals in G. Then there is a sequence of allowed traversals that leads from (s, p) to (s', p'), so in the transitive closure of G, $(s, p) \to (s', p')$ is a traversal. Therefore a traversal between external ports is allowed in the contraption if and only if the corresponding traversal is allowed in the transitive closure of G. □

Using the prior simulations we are able to show all door gadgets are universal.

Corollary 73. *Door gadgets can simulate any gadget.*

Proof. Theorems 68, 70, and 71 show that any door gadget can simulate a directed, open-optional door. The traverse tunnel of such a door can simulate a diode. By adding two diodes around the open port of the open-optional door, we can make it a directed open tunnel. We now have shown all types of door gadget can simulate the open-required directed door gadget which can simulate any other gadget by Theorem 72. □

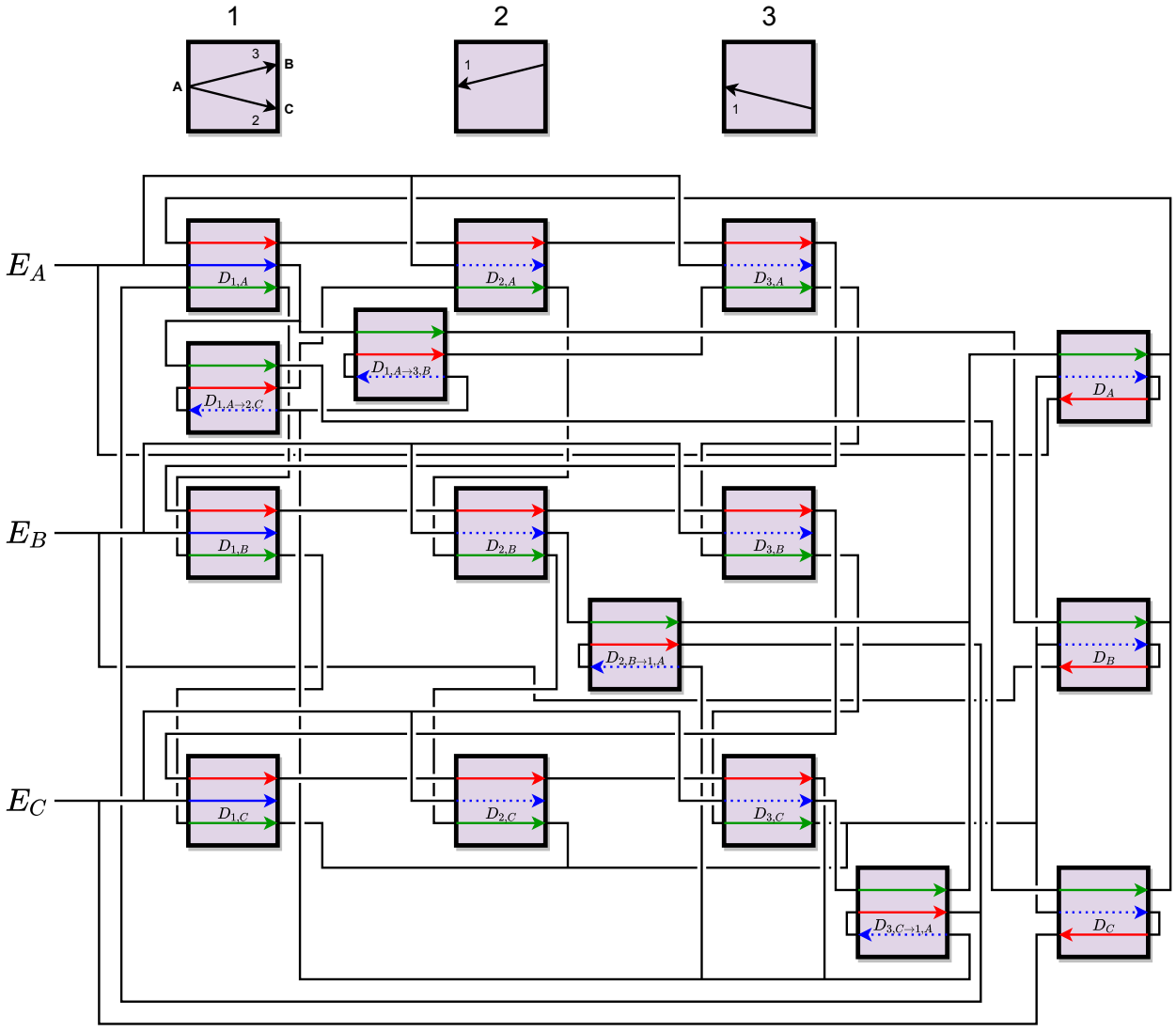


Figure 2-56: Example of a simulation of a gadget with the open-required directed door, as constructed in the proof. The state diagram of the gadget that is simulated is shown on top. This simulation starts in state 1.

2.8 Self-Closing Doors

In this section, we introduce different kinds of self-closing doors and show that 1-player motion planning is PSPACE-hard for them. We then prove certain self-closing doors universal.

2.8.1 Terminology

A ***self-closing door*** is a 2-state gadget that has a tunnel that closes itself when traversed (the ***self-closing*** tunnel), a tunnel/port that reopens said tunnel (the ***opening*** tunnel/port), and no other ports. We will talk about two major kinds of self-closing door. A ***normal self-closing door*** is a self-closing door where the open path/tunnel is always open. A ***symmetric self-closing door*** is a self-closing door where the open path/tunnel is a tunnel and also closes itself when traversed. As with doors, these can be ***directed***, ***undirected***, or ***mixed***, and a normal self-closing door can also be ***open-required*** or ***open-optional***. An "X" on a tunnel indicates that the tunnel closes itself when traversed. A dotted line indicates a closed tunnel and a solid line indicates an open tunnel. For normal self-closing doors, the open path/tunnel will be colored green. Figure 2-57 shows some self-closing doors.

2.8.2 PSPACE-hardness of Self-Closing Doors

In this section we show PSPACE-hardness for 1-player motion planning with any of the self-closing doors. We do so by showing undirected self-closing doors can simulate diodes,

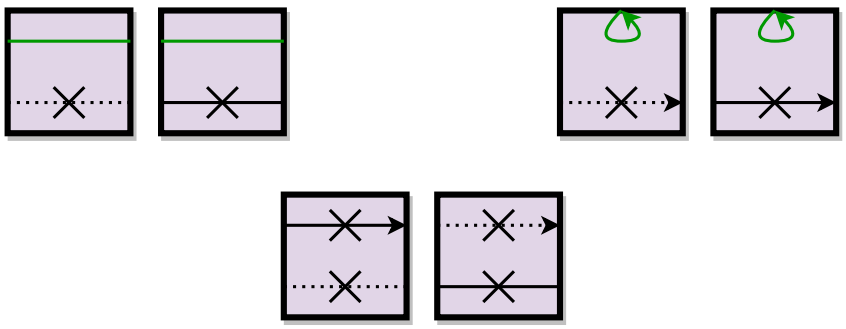


Figure 2-57: Left: An undirected open-required normal self-closing door. Right: A directed open-optional normal self-closing door. Bottom: A mixed symmetric self-closing door.

and self-closing doors without open-optional tunnels can simulate ones with open-optional tunnels. We then prove the main Theorem 76 which gives PSPACE-hardness of the directed, open-optional, normal self-closing door by simulating a directed, open-optional door gadget discussed in Section 2.7.

Lemma 74. *In 1-player motion planning, any normal or symmetric self-closing door can simulate an open-optional self-closing door.*

Proof. In the case of an open-optional normal self-closing door, we are done. In the case of an open-required normal self-closing door, we do the same thing we did for the proof for Theorem 68. In the case of a symmetric self-closing door, we pick a tunnel to be the opening tunnel and do what we did for Theorem 68. This simulates an open-optional self-closing door. □

Lemma 75. *1-player motion planning with the undirected open-optional normal self-closing door can simulate a directed open-optional normal self-closing door.*

Proof. We can simulate a diode by wiring 2 undirected open-optional normal self-closing doors as shown in Figure 2-60. The player can enter from the left, open the left self-closing door, traverse it, and do the same for the right self-closing door. The player cannot enter from the right. If the player tries to open the left self-closing door and then leave, the player still cannot enter from the right. If the player tries to open the right self-closing door and then leave, they will not be able to leave. So this simulates a diode. We can wire a diode to each side the self-closing tunnel to get a directed self-closing tunnel which can be applied to make the undirected self-closing door directed. □

Theorem 76. *1-player motion planning with the directed open-optional normal self-closing door is* PSPACE*-hard.*

Proof. We can simulate a diode by wiring the opening port to the input end of the self-closing tunnel. The player can open the self-closing tunnel then traverse it, but cannot go the other way because the self-closing tunnel is directed. Then we show that we can duplicate

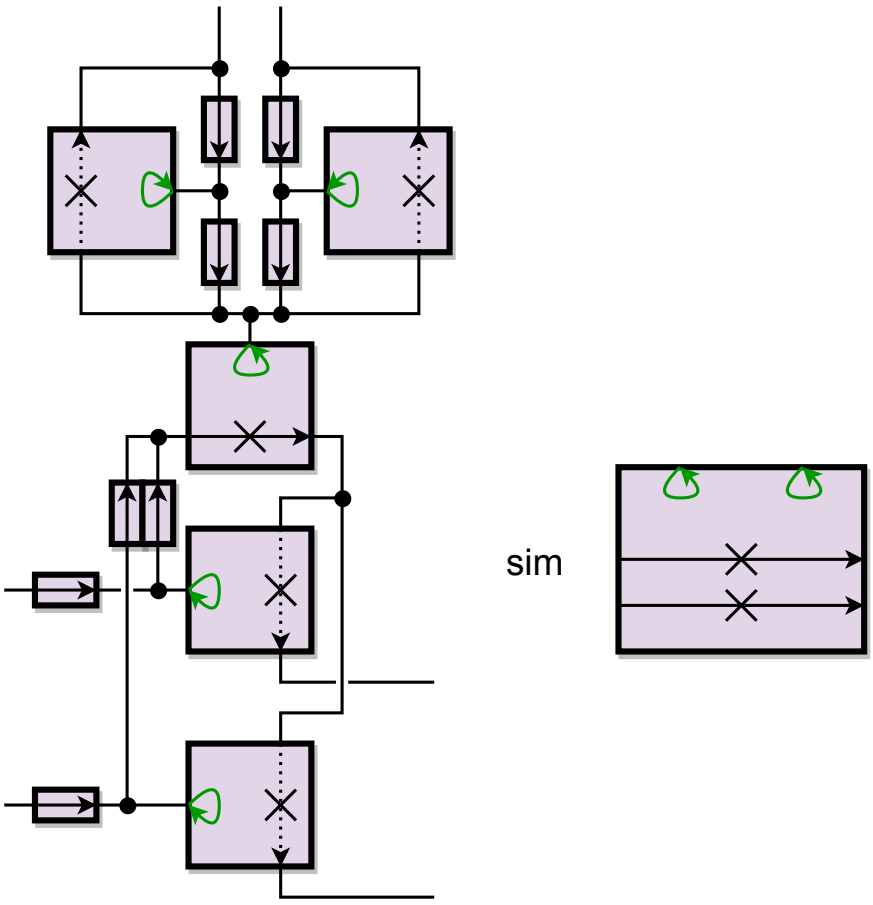


Figure 2-58: The directed open-optional normal self-closing door can simulate a version of itself with the opening port and the self-closing tunnel duplicated. Note that the opening port duplicator is planar.

the open port and the self-closing tunnel as in Figure 2-58. We then actually triplicate the open port and duplicate the self-closing tunnel, and wire them up to simulate the directed open-optional door as shown in Figure 2-59, for which PSPACE-hardness is known. □

Chaining the simulations in Lemmas 74 and 75 with Theorem 76 we obtain PSPACE-hardness for all variations.

Corollary 77. *1-player motion planning with any normal, symmetric, or open-optional normal self-closing door is* PSPACE*-hard.*

Since we show PSPACE-completeness of self-closing doors by simulating a door gadget which is shown to be universal in Theorem 73, universality follows for self-closing doors.

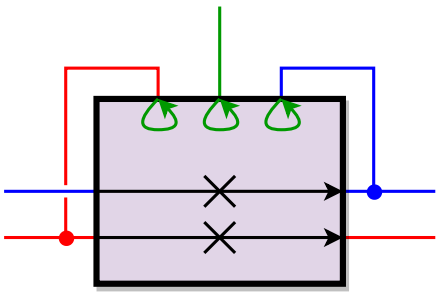


Figure 2-59: Simulation of the directed open-optional door. Green wires correspond to the opening port; blue wires correspond to the traverse tunnel; and red wires correspond to the closing tunnel. Note that the player has no reason to not open the gadget after traversing the blue wire.

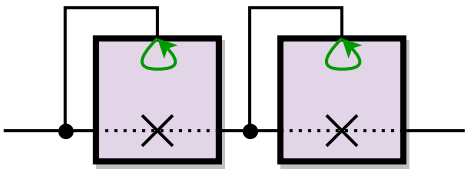


Figure 2-60: Undirected open-optional normal self-closing door simulating a diode

Corollary 78. *Any self-closing door (directed or undirected, open-optional or open-required, normal or symmetric) can simulate any gadget.*

2.9 Planar Doors

In this section, we show that 1-player planar motion planning with any normal or symmetric self-closing door is PSPACE-hard. In addition, we show that it is PSPACE-hard for all but one door and NP-hard for the remaining case.

2.9.1 Terminology

In 2D, we care about the arrangement of ports in a gadget. For ***planar motion planning*** problems we want a ***planar*** system of gadgets, where the gadgets and connections are drawn in the plane without crossings. Planar gadgets also specify a clockwise ordering of their ports, although we consider rotations and reflections of a gadget to be the same. A single gadget type thus corresponds to multiple planar gadget types, depending on the choice of the order of locations. For a planar system of gadgets, the gadgets are drawn as small diagrams with points on their exterior corresponding to their ports and connections are drawn as paths connecting the points corresponding to the ports without crossing gadget interiors or other

connections.

2.9.2 PSPACE-hardness for Planar Self-Closing Doors

For completeness, we give a proof that the planar directed open-optional normal self-closing door is PSPACE-hard. This result was also given in [6].

Theorem 79. *1-player planar motion planning with the directed open-optional normal self-closing door is* PSPACE*-hard.*

Proof. Since Theorem 76 shows PSPACE-completeness in the non-planar case, it will suffice to build a crossover gadget. First, we wish to duplicate the opening ports as in the prior proof. We show how to do so in Figure 2-58. Note that this time we cannot directly duplicate the self-closing tunnel as the construction from Theorem 76 uses crossovers. We can also simulate a diode as proven in Theorem 76 since the construction is planar. We use these to simulate a pair of self-closing doors where the opening ports alternate which door they open, shown in Figure 2-61. If the agent enters from port 1 or 4, they will open door E or F, respectively, and then leave. If the agent enters from port 2, they can open doors A, B, and C. Assume they then traverse door B. If they then open door E, they would have to traverse door C, maybe open F, and get stuck. So instead of opening door E, the agent traverses door A, ending up back at port 2 with no change except that door C is open. Entering port 2 or 3 gives the opportunity to open door C without being forced to take a different path, so leaving door C open does not help. So instead of traversing door B, the agent traverses door C. The agent is then forced to go right and can open door F. Then they are forced to traverse door B. If the agent opens door E, they will be stuck, so the agent traverses door A instead and returns to port 2, leaving door F open. Similarly, if the agent enters from port 3, the only useful thing they can do is open port E and return to port 3.

Using this, we then simulate a directed crossover as in Figure 2-62 and simulate an undirected crossover as in Figure 2-63, removing the planar restriction and reducing this problem to Theorem 76. In the simulation of a directed crossover, the agent must open the

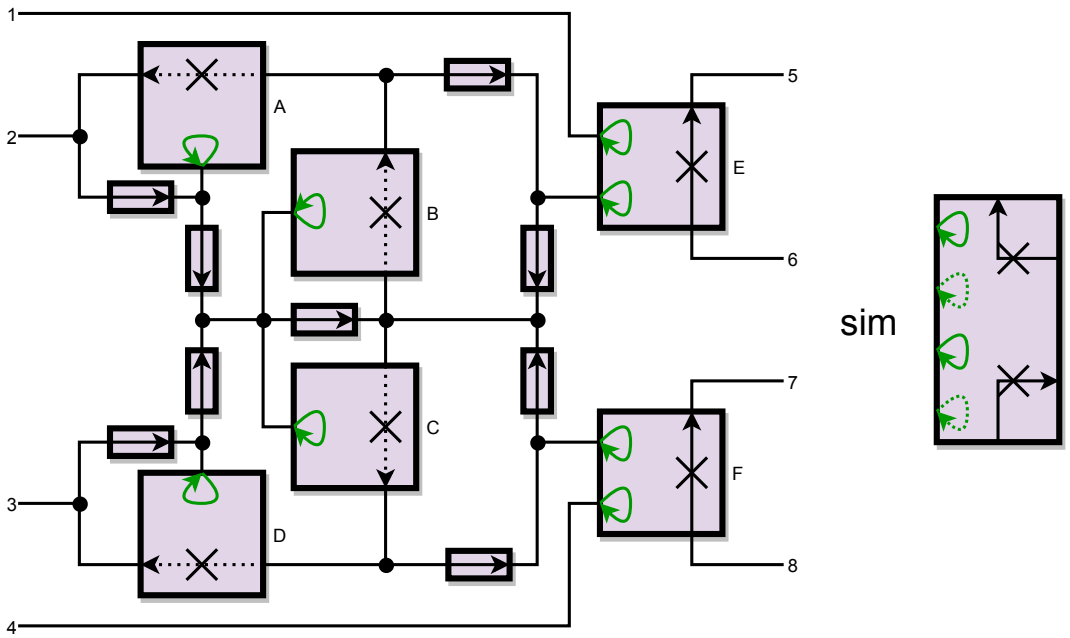


Figure 2-61: Directed open-optional normal self-closing door simulating the gadget on the right, where solid opening ports control the top self-closing tunnel and dotted opening ports control the bottom self-closing tunnel. The gadgets and external ports are labelled to help with the proof.

left tunnel of a gadget and then open both tunnels of the other one, forcing them to cross over, since the only path forward goes through the left tunnels of both gadgets. □

Theorem 80. *1-player planar motion planning with any normal or symmetric self-closing door is* PSPACE*-hard.*

Proof. Any normal or symmetric self-closing door can simulate a diode as shown in Figure 2-64(a–f). Then we can simulate the directed open-optional normal self-closing door as shown in Figure 2-65(a–d). Finally we apply Theorem 79 to show PSPACE-hardness. □

2.9.3 PSPACE-hardness for Planar Doors

We will show that 1-player planar motion planning with almost any door is PSPACE-hard by showing that 1-player planar motion planning with almost any fully directed door is

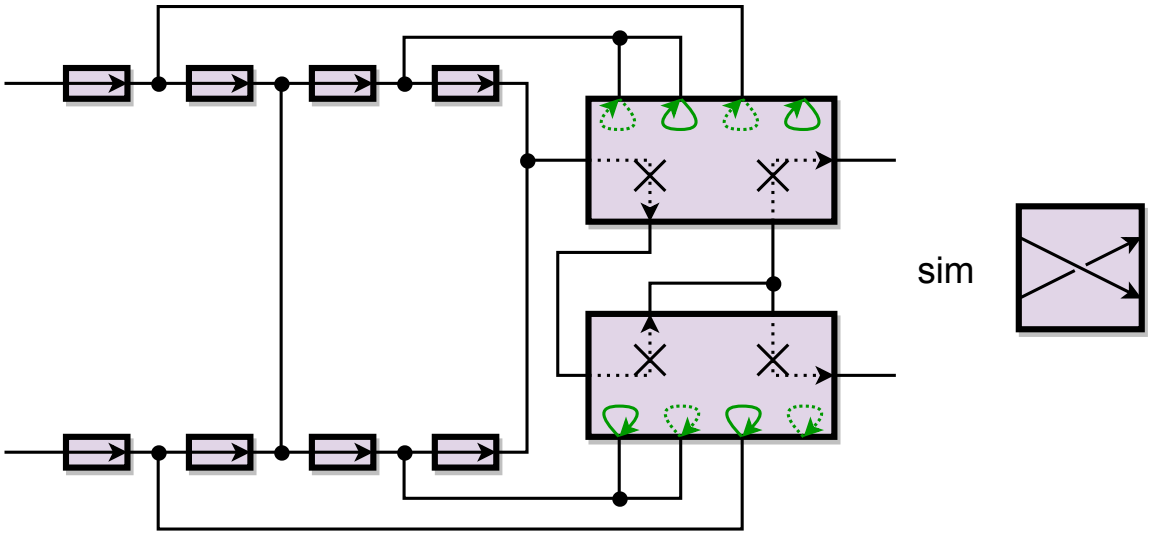


Figure 2-62: Directed open-optional normal self-closing door simulating a crossover.

PSPACE-hard and that mixed and undirected doors can planarly simulate at least one of the PSPACE-hard fully directed doors.

We first note that the diode construction in Theorem 71 is planar. This is examined in more detail in Lemmas 81 and 82. Since undirected and partially directed doors can planarly simulate at least one fully directed door, it suffices to prove hardness for all fully directed doors.

Next, we show hardness for all fully directed doors with at least one pair of crossing tunnels. If we collapse adjacent opening ports to optional opening ports as in Theorem 68, this leaves 12 fully directed doors with no crossing tunnels, shown and named in Figure 2-66. The 12 cases can be enumerated by first considering placing the traverse and close tunnels in either a parallel and anti-parallel orientation, then noting that there are four regions made by these two tunnels in which to place the open tunnel, leading to $4 \cdot 2$ cases with the open optional port and $2 \cdot 2$ cases with a directed open tunnel. Theorem 85 proves PSPACE-completeness for 11 of the 12 of these cases, while Theorem 87 proves NP-hardness for the remaining Case 8: OTtocC door.

Proofs for 11 of the 12 of these cases are given in Theorem 85. Finally, we show NP-hardness for the remaining Case 8: OTtocC door in Theorem 87.

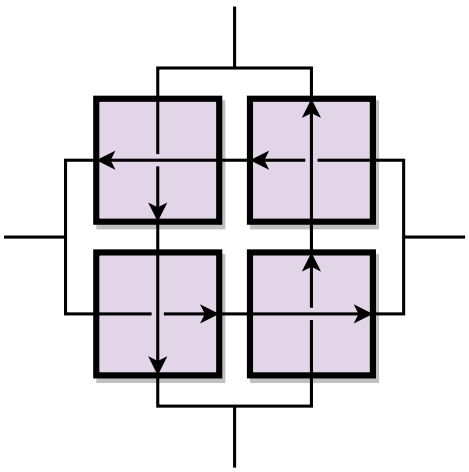


Figure 2-63: Directed crossover simulating an undirected crossover.

Next, we show hardness for all fully directed doors with at least one pair of crossing tunnels. If we collapse adjacent opening ports to optional opening ports as in Theorem 68, this leaves 12 fully directed doors with no crossing tunnels, shown and named in Figure 2-66. The 12 cases can be enumerated by first considering placing the traverse and close tunnels in either a parallel and anti-parallel orientation, then noting that there are four regions made by these two tunnels in which to place the open tunnel, leading to $4 \cdot 2$ cases with the open optional port and $2 \cdot 2$ cases with a directed open tunnel. Theorem 85 proves PSPACE-completeness for 11 of the 12 of these cases, while Theorem 87 proves NP-hardness for the remaining Case 8: OTtocC door.

Lemma 81. *Any mixed door can planarly simulate some fully directed door which is not the Case 8: OTtocC door.*

Proof. Consider an arbitrary mixed door M. Since M is mixed, it has a directed tunnel. No tunnel changes its own traversability when crossed, so this tunnel simulates a diode. We wire each undirected tunnel of M through diodes at each end pointing in the same direction. This simulates a directed door. If M is not the door in Case 8: OTtocC, we are done. Otherwise, flip one set of diodes wired through an undirected tunnel of M, simulating a different directed door. □

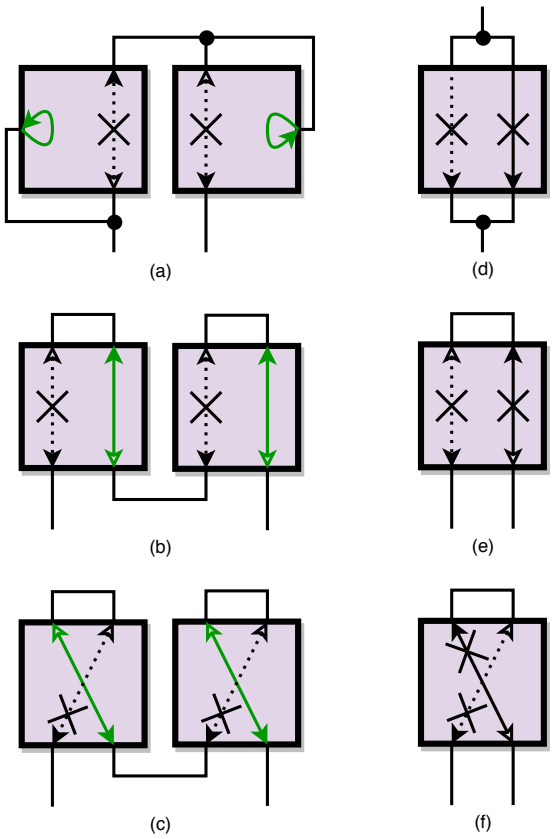


Figure 2-64: Six types of self-closing doors simulating diodes. Filled-in arrows indicate directions that are required to exist, and outlined arrows indicate optional directions. Case (a) is the same as Figure 2-60.

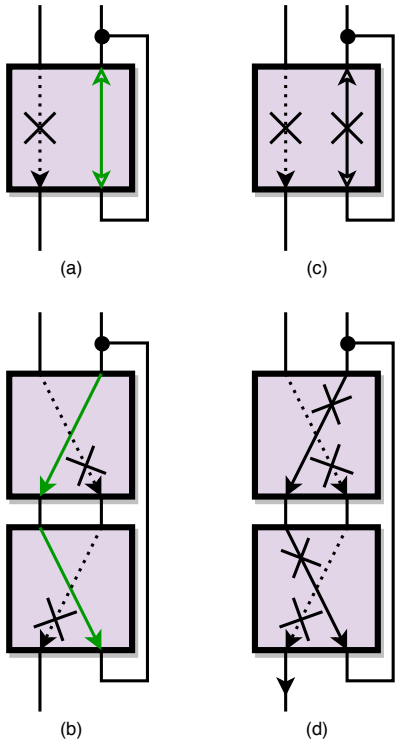


Figure 2-65: Four types of directed self-closing doors simulating the directed open-optional normal self-closing door. Filled-in arrows indicate directions that are required to exist, and exactly one of the outlined directions must exist.

Lemma 82. *An undirected door can planarly simulate a fully directed door which is not the Case 8: OTtocC door.*

Proof. Consider an arbitrary undirected door U. We wire an external wire to a port of the opening port/tunnel. The player can visit the port, or if it is a tunnel, cross the tunnel both ways, to open the gadget. If the opening port/tunnel was a tunnel, this turns it into a port, making the gadget U'. Consider the order of the ports of the opening port O, the traverse tunnel $\{T_0, T_1\}$, and the closing tunnel $\{C_0, C_1\}$ around the edge of U', and label the ports p_0, p_1, p_2, p_3, p_4. We want to show that a traverse tunnel port is adjacent to a closing tunnel port. Assume not. Without loss of generality, let $p_0 = T_0$. Then $\{p_1, p_4\} = \{T_1, O\}$. But

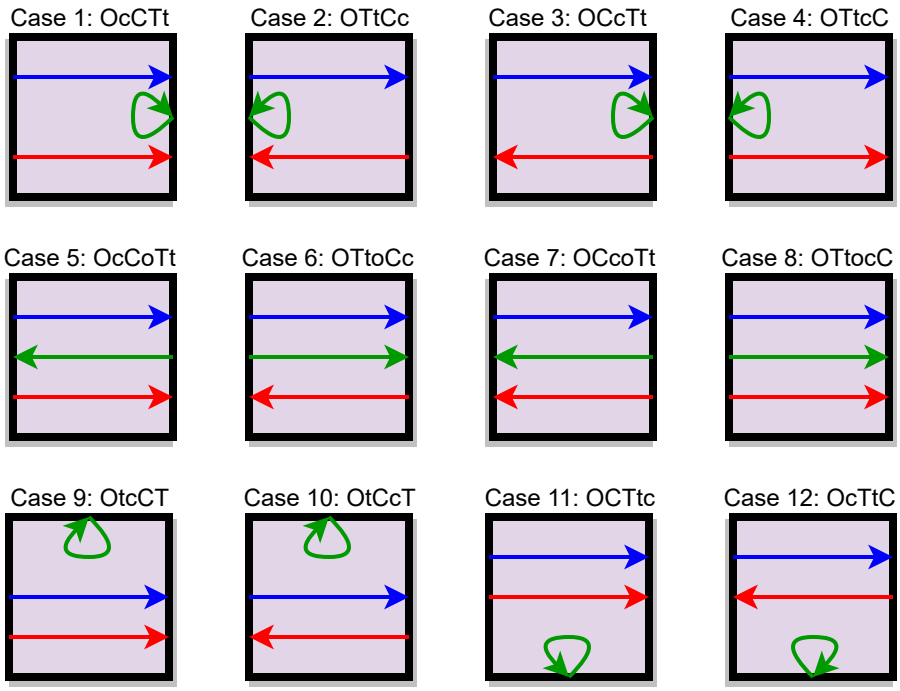


Figure 2-66: The twelve cases of a planar directed door without internal crossings. Opening tunnels with adjacent ports are merged into opening ports.

then $\{p_2, p_3\} = \{C_0, C_1\}$, and one of $\{p_2, p_3\}$ must be adjacent to a traverse tunnel port, a contradiction. Since one of the traverse tunnel ports, say T_1, is adjacent to one of the closing tunnel ports, say C_0, we wire T_1 to C_0 without blocking an opening port or opening tunnel port. This simulates a directed open-optional normal self-closing door: The player can open the gadget by going to the opening port (or if it is a tunnel, by going through the tunnel and back). If the gadget is open, the player can go through the traverse tunnel and then the closing tunnel, but cannot go the other way. If the gadget is closed, the player cannot go either way through the traverse-tunnel-closing-tunnel path. □

Theorem 83. *1-player planar motion planning with any directed door with an internal crossing is* PSPACE*-hard.*

Proof. If the opening tunnel crosses the closing tunnel, then we have a crossover because these tunnels are always open. If the opening tunnel crosses the traverse tunnel, then we start the door open and have a crossover because neither tunnel closes itself or the other.

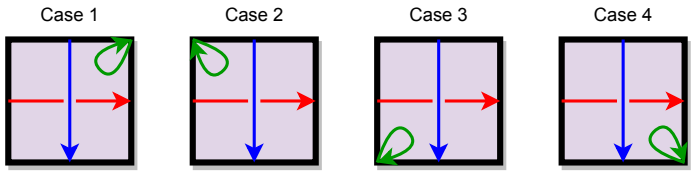


Figure 2-67: The four cases where the traverse tunnel crosses the closing tunnel but the opening port/tunnel does not cross either and can thus simulate a port.

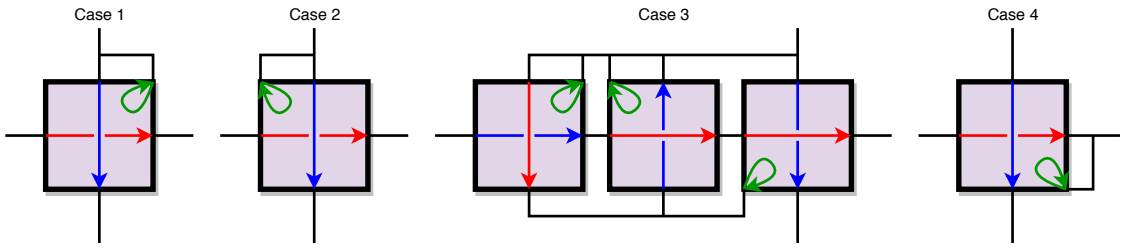


Figure 2-68: All four cases of the traverse tunnel crossing the closing tunnel can each simulate a crossover.

Otherwise, the traverse tunnel crosses the closing tunnel and the opening port/tunnel can simulate an opening port. Then we have four cases, as shown in Figure 2-67. In cases 1, 2, and 4, we can simulate a crossover by connecting the opening port to either the input of the traverse tunnel or the output of the closing tunnel to ensure that the traverse tunnel is open when we need to use it. (Figure 2-68).

Case 3, however, is more tricky, as both of these ports are separated from the opening port by other ports. We use 2 copies to provide a path from the input of the traverse tunnel to the opening port without giving access to the close tunnel. The horizontal path of the crossover involves crossing from the left door to the right door, which is allowed as long as the left door is open. To take the vertical path, the player opens the middle door, goes down closing the left door, opens the right door, traverses the middle door, opens the left door (to keep the horizontal path open), and traverses the right door. The player can leave partway through this traversal, but this does nothing useful. So all doors with internal crossings can simulate crossovers, removing the planarity constraint. □

Before continuing, we prove another gadget, the directed tripwire lock, is PSPACE-complete. Recall that a tripwire lock is a 2-state 2-tunnel gadget with an undirected tunnel

that is traversable in exactly 1 state and an undirected tunnel that toggles the state of the gadget [25]. The ***directed tripwire lock*** is similar except that its tunnels are directed.

Lemma 84. *1-player planar motion planning with the parallel directed tripwire-lock is PSPACE-hard.*

Proof. The parallel directed tripwire lock can simulate the antiparallel directed tripwire lock, as in Figure 2-69. Crossing the tripwire in gadget 2 forces the agent to cross the tripwire in gadget 1, so exactly 1 of the locks of gadgets 1 and 2 are locked. Similarly, exactly 1 of the locks of gadgets 3 and 5 are locked. Crossing the tripwire in gadget 4 forces the agent to exit the simulation (or be stuck), and is also the only way to exit when the agent comes from the top left port, so the lock in gadget 4 is unlocked after an even number of crossings of the top path and locked after an odd number of crossings. Since the locks of gadgets 1 and 2 are anti-correlated, if the agent wants to unlock the lock in gadget 1, it must afterward cross the lock in gadget 3. But said lock must be unlocked by going in a loop through the lock in gadget 4, which is unlocked only after an even number of crossings of the top path. So during an even-indexed (second, fourth, etc.) crossing of the top path, the lock in gadget 1 must be locked before the agent can leave. During an odd-indexed crossing, the agent can take the loop through the lock in gadget 4, unlock the lock in gadget 1, take the loop again to unlock the exit, and exit. So the top path behaves like a directed tripwire for the lock in gadget 1, which is the bottom path.

The parallel and antiparallel directed tripwire locks together simulate a directed tripwire-lock-tripwire with antiparallel tripwires (Figure 2-70). In this gadget going through the top or bottom pairs of tripwires flips which of the middle locks is closed. If the top and bottom set of locked tunnels are paired, then it blocks the middle pathway; however, if they are opposite, the connection in the middle can be used to traverse them.

This gadget in turn simulates a crossover (Figure 2-71), removing the planarity constraint. Here both pathways lock two of the gadgets before reaching the center preventing the agent from exiting via a disallowed path, and then they route through those gadgets again opening them back up and restoring the directed crossover to its original state. □

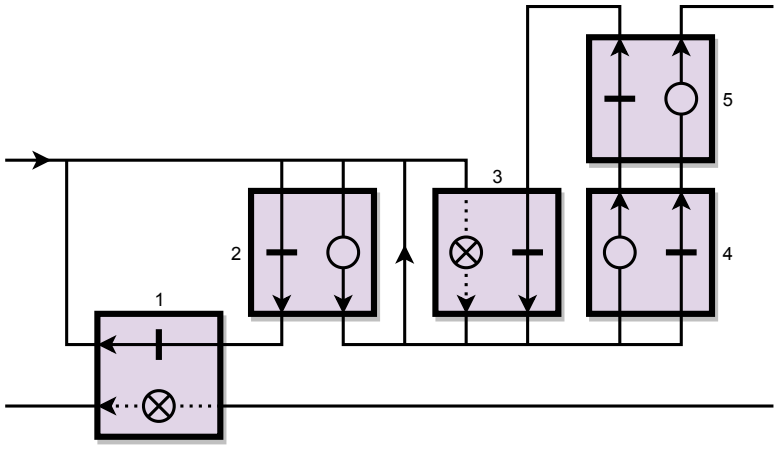


Figure 2-69: Simulation of an antiparallel directed tripwire lock with a parallel directed tripwire lock. Gadgets are labelled to aid explanation.

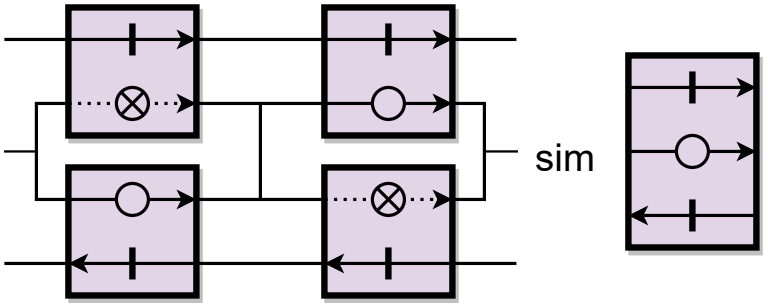


Figure 2-70: Simulation of a directed tripwire-lock-tripwire with antiparallel tripwires using both parallel and antiparallel directed tripwire locks.

For directed doors, there are only the cases without internal crossings left. If the opening port/tunnel is a tunnel and its ports are adjacent, we easily simulate an opening port, reducing the number of cases to consider. There are twelve cases, shown in Figure 2-66. We name these cases based on the cyclic order of ports, with exits-only having lowercase letters.

Theorem 85. *1-player planar motion planning with any directed door without internal crossings except the Case 8: OTtocC door is* PSPACE*-hard.*

Proof. We divide into multiple cases. Note the cases are numbered according to Figure 2-66, not in the order they are addressed in this proof.

Case 2: OTtCc, Case 10: OTcCt, and Case 12: OcTtC doors. In all these doors the opening port/tunnel is a port, and the traverse tunnel output is adjacent to the closing

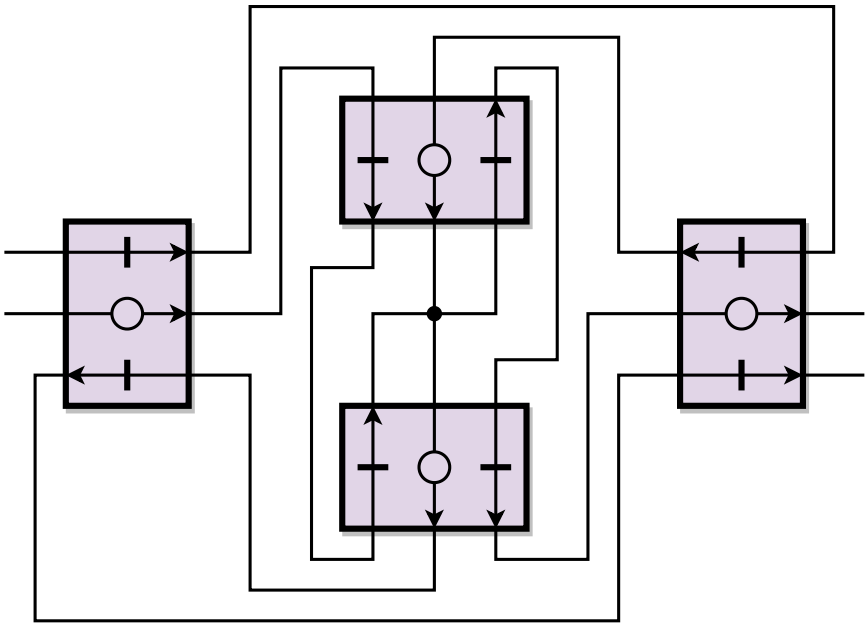


Figure 2-71: Simulation of a crossover using a directed tripwire-lock-tripwire with antiparallel tripwires.

tunnel input. Thus, we can simulate a directed open-optional self-closing door by wiring the traverse tunnel output to the closing tunnel input and by attaching a wire to the open port, and these wires do not cross each other. Then this reduces to Theorem 80.

Case 1: OcCTt door. can simulate the directed version of the tripwire lock, as shown in Figure 2-72. We will refer to the gadgets numbered left to right. The lock is simply the traverse tunnel on door 1. In the two simulated states we will either have doors 1 and 3 open or door 2 open. If door 2 is open, when traversing the tripwire tunnel we can go through the traverse tunnel allowing us to open doors 1 and 4. With door 4 now open, we can go through its traverse tunnel opening door 3, and then closing door 4 on the way out. This leaves us with doors 1 and 3 open. Going through the tripwire tunnel again closes door 1 but allows us to go through the traverse tunnel of door 3, allowing us to open door 2. Doors 3 and 4 are then closed on the way out. There are states where we could fail to open all of these doors while traversing the close tunnel, but this will leave the gadget with strictly less traversability and thus the agent will never want to take such a path. Thus the Case 1:

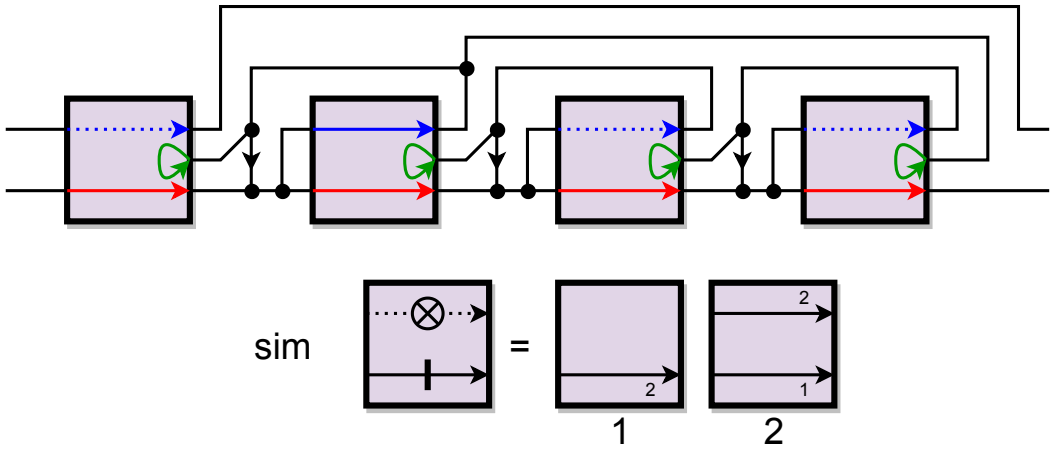


Figure 2-72: The Case 1: OcCTt door simulates the parallel directed tripwire lock. In addition, the state diagram of the directed tripwire lock. Arrows are drawn directly on wires to represent diodes.

OcCTt door is PSPACE-complete by Lemma 84.

Case 3: OCcTt door. This door can simulate a directed open-optional normal self-closing door (Figure 2-73). If the agent enters from port O (the opening port), they can open doors 2 and 3. If they then leave, they have accomplished nothing because door 2 was already open, and door 3 can be opened from port O anyway and cannot be traversed from port T_0 or T_1 as we will see later. So they close door 2 instead. Then they can open door 1 and they are forced to traverse door 3. The agent can then reopen door 2 and return to port O. Now all the doors are open. If the agent then enters from port T_0, then they are forced to close door 3. They can then open door 1 (useless), and then they are forced to traverse door 2 and close door 1, leading to port T_1. The agent could not have taken this path initially because door 1 was closed, and they cannot take it again without visiting port O because they just closed door 1.

Case 4: OTtcC door. This door can simulate a directed open-optional normal self-closing door (Figure 2-74). If the agent enters from port O, they can open door 1, and then they are forced to close door 2. Continuing this loop does nothing, so the agent then returns to port O. Now door 1 is open and door 2 is closed. If the agent then enters from T_0, then they are

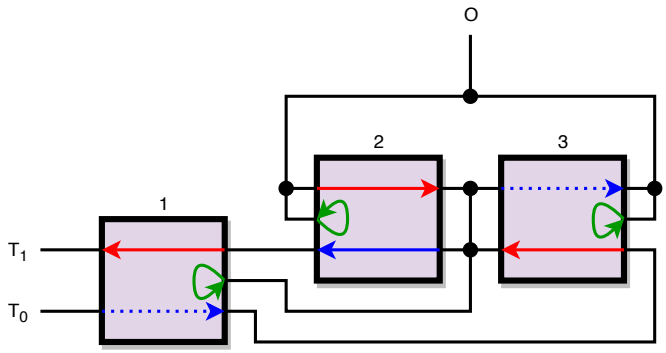


Figure 2-73: Simulation of a self-closing door with the Case 3: OCcTt door. The simulation starts in the closed state. Ports and gadgets are labelled.

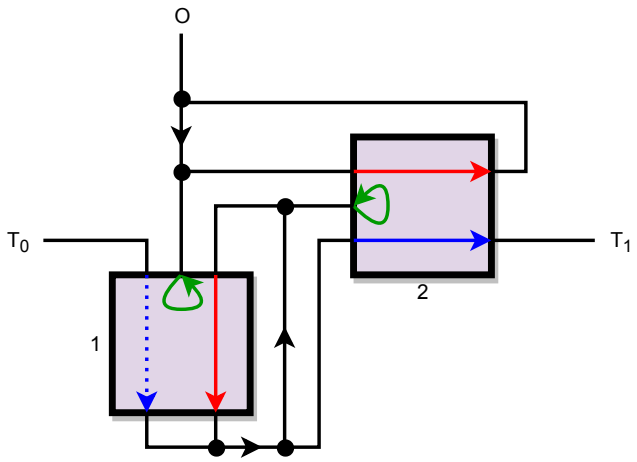


Figure 2-74: Simulation of a self-closing door with the Case 4: OTtcC door. The simulation starts in the closed state. Ports and gadgets are labelled.

forced to traverse door 1. They can then open door 2 and then they are forced to close door 1. Continuing the loop does nothing, so the agent has no other option but to traverse door 2 to port T_1. The agent could not have taken this path initially since door 1 was closed, and they cannot take it again without visiting port O because they closed door 1.

Case 6: OTtoCc door. This door can simulate a directed open-optional normal self-closing door (Figure 2-75). If the agent enters from port O, they are forced to close door 3. If the agent then traverses door 2, they are forced to open door 3 and return to port O, accomplishing nothing. So the agent has no other option but to close door 1. If the agent tries to open door 2, they get stuck, so they instead open door 1. Continuing the loop

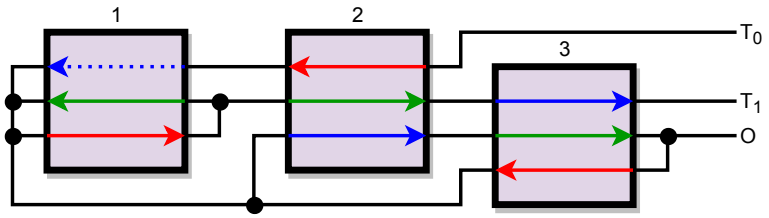


Figure 2-75: Simulation of a self-closing door with the Case 6: OTtoCc door. The simulation starts in the closed state. Ports and gadgets are labelled.

involving door 1 does nothing, so the agent then traverses door 2, opens door 3, and returns to port O. Now door 1 is open. If the agent enters from port T_0, then they are forced to close door 2, traverse door 1, and close door 1. Reopening door 1 puts the agent back into the situation of being forced to close door 1, so the agent instead opens door 2 and traverses door 3 to port T_1. The agent could not have taken this path initially since door 1 was closed, and they cannot take it again without visiting port O because they closed door 1.

Case 5: OtToCc door. This door can simulate the Case 6: OTtoCc door, which has been covered, by effectively flipping the traverse tunnel. (Figure 2-76). Door 1 is the gadget that we flip the traverse tunnel of. If the agent enters from port T_0, they must open door 2, the close door 2. If door 1 is open and the agent then traverses it, they are back to a previous position with nothing changed. Instead, the agent opens door 3. If the agent then closes door 3, they get stuck because door 2 is closed. So they must close door 2 (again) or traverse door 3. These actions lead to the same situation. If the agent opens door 3 (again), they are back to the same situation that occurred after opening door 3 the first time. If door 1 is open, the agent then traverses door 1. Then they must open door 2. Closing door 2 leads to a previous situation, so the agent then traverses door 3. If the agent then traverses door 1 (again), they must open door 2 (again), leading to a previous situation. So they instead open door 3. Closing door 2 and traversing door 3 lead to different previous situations, so the agent then closes door 3, and then is forced to traverse door 2 to port T_1, leaving all the doors unchanged. If door 1 is not open, then the agent is unable to leave.

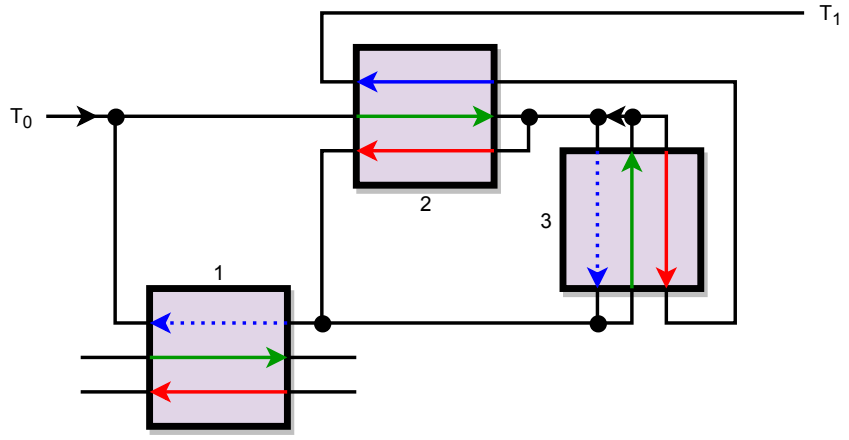


Figure 2-76: Simulation of the Case 6: OTtoCc with the Case 5: OcCoTt door. The traverse tunnel of the leftmost gadget is effectively flipped.

Case 7: OtTocC door. This door can simulate a directed open-optional normal self-closing door (Figure 2-77). If the agent enters from port O, they must open door 1, then close door 2. If the agent then closes door 3, they get stuck because door 2 is closed. The agent can traverse door 1 and leave via port O, but they can also open and then traverse door 3 and then do the same thing, which is advantageous. So the agent opens and traverses door 3, then traverses door 1 to port O. Now door 1 is open, door 2 is closed, and door 3 is open. If the agent enters from port T_0, they must close door 1, then open door 2, then traverse door 3. Opening door 3 and then traversing it is a no-op, and door 1 is closed, so the agent closes door 3 and then must traverse door 2 to port T_1. This leaves door 1 closed, door 2 open, and door 3 closed. The agent could not have taken this path initially because door 3 was closed, and cannot take it again without visiting port O first for the same reason.

Case 9: OtcCT door. This door can simulate a directed open-optional normal self-closing door (Figure 2-78). If the agent enters from port O, they can open door 1 and must close door 2. If the agent later enters from port T_0, then they must traverse door 1. They then can open door 2 (and must, since that is the only way out) and must close door 1. Then the agent traverses door 2 to port T_1. The agent could not have taken this path initially because door 1 was closed, and cannot take the path again without visiting port O first for the same

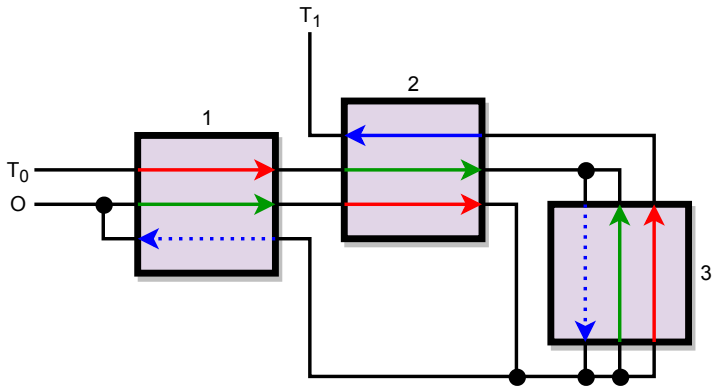


Figure 2-77: Simulation of a self-closing door with the Case 7: OCcoTt door.

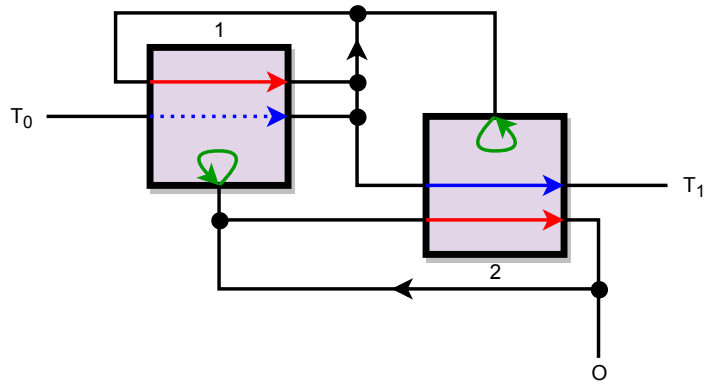


Figure 2-78: Simulation of a self-closing door with the Case 9: OtcCT door.

reason.

Case 11: OCTtc door. This door can simulate the Case 12: OcTtC door, which has been covered, by effectively flipping the traverse tunnel. (Figure 2-79). Door 1 is the gadget that we flip the traverse tunnel of. If the agent enters from port T_0, then they must traverse the bottom-left diode. The agent can then open doors 2 and 3 but must pick one to close. If they close door 2, they get stuck. So the agent closes door 3. Going to open doors 2 and 3 again leads to a previous situation, so the agent instead traverses door 2. Traversing door 1 (if it is open) leads to a previous situation, and traversing door 5 leads to being stuck. The agent traverse the right diode, and can open doors 4 and 5 but must pick one to close. Closing door 4 leads to a previous situation, so the agent then closes door 5, then must traverse door 4. Going to open doors 4 and 5 again leads to a previous situation. If door 1 is

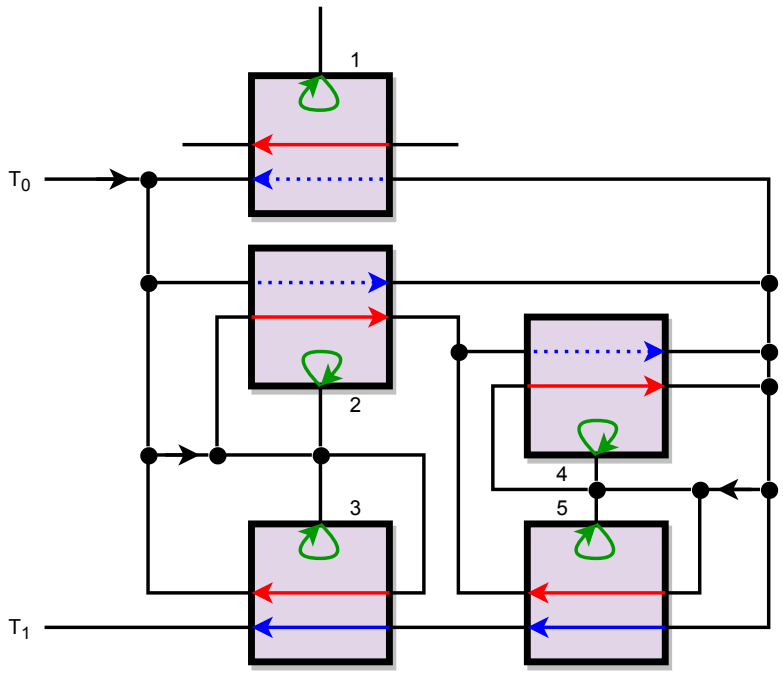


Figure 2-79: Simulation of the Case 12: OcTtC with the Case 11: OCTtc door. The traverse tunnel of the topmost gadget is effectively flipped.

open, then the agent traverses door 1. Traversing door 2 leads to a previous situation, so the agent then opens doors 2 and 3. Closing door 3 leads to a previous situation, so the agent closes door 2, then must traverse door 4. Traversing door 1 leads to a previous situation, so the agent instead opens doors 4 and 5. Closing door 5 leads to a previous situation, so the agent then closes door 4. Traversing door 1 or going to open doors 4 and 5 both lead to previous situations, so the agent then traverses door 5 and must traverse door 4, leaving via port T_1 and leaving all the doors unchanged. If door 1 is not open, however, then the agent cannot leave because door 3 is closed and the only way to open it is through door 1's traverse tunnel.

This covers all the planar directed doors without internal crossings except the OTtocC door, finishing the proof. □

Theorem 86. *1-player planar motion planning with any door except the door in Case 8: OTtocC is* PSPACE*-hard.*

Proof. This follows from Theorems 83, 85, 81, and 82, as those cover all the cases. □

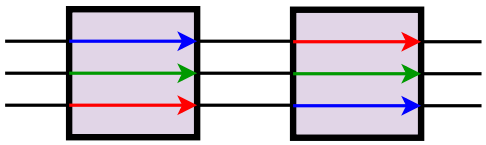


Figure 2-80: Simulation of parallel double-close door with the Case 8: OTtocC door.

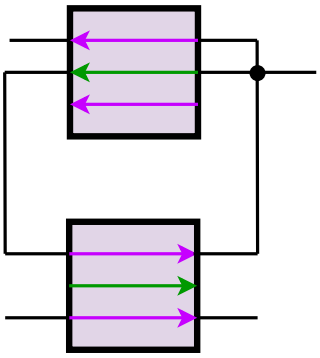


Figure 2-81: Simulation of an antiparallel NAND gadget with a parallel double-close door.

Theorem 87. *1-player planar motion planning with the door in Case 8: OTtocC is* NP*-hard.*

Proof. We show how to simulate antiparallel NAND gadgets, which is NP-hard by Lemma 48. First, Figure 2-80 shows how to combine two Case 8: OTtocC doors to build a door-like gadget with an open tunnel and two traverse–close tunnels, where traversing the open tunnel opens both traverse–close tunnels, and traversing either traverse–close tunnel closes the other traverse–close tunnel. Next, Figure 2-81 shows how to combine two of these gadgets to build an antiparallel NAND gadget. The top tunnel in the top gadget is initially closed, forcing the agent to open it and thus close the bottom tunnel of the bottom gadget, which is possibly only if the bottom tunnel of the bottom gadget was not already traversed. Because the open tunnel of the bottom gadget is not connected to anything, both tunnels of the bottom gadget will remain closed once closed. □

Chapter 3

Multi-Player

In this chapter we explore results related to the 2-player and team imperfect information models. We give a partial characterization for k-tunnel reversible deterministic gadgets, showing that gadgets with interacting tunnels are EXPTIME-complete for the 2-player model, shown in Section 3.2.1, and RE-complete for the team imperfect information model, shown in Section 3.2.2. However, it is unclear if motion planning with non-interacting tunnels remain easy in the 2-player or team models, and in fact we suspect the 1-toggle to be at least PSPACE-hard. For the polynomially bounded case, we show that single-use gadgets suffice for PSPACE-completeness for the 2-player model, shown in Section 3.3.1, and the single use gadget suffices for NEXPTIME-hardness for team imperfect information, whon in Section 3.2.2. This gives a full classification for DAG gadgets and a partial classification for LDAG gadgets.

These results come primarily from [29], coauthored with Dylan Hendrickson and Erik Demaine. Many of the proofs and ideas in this section, especially for reversible, deterministic gadgets come from Dylan Hendrickson.

3.1 Simple Relations Between Models

The 2-player model is strictly stronger than the 1-player model, as the opponents can be disconnected from their goal giving a trivial reduction to the 1-player case. In addition, so long as the gadgets have at least one transition, one may use the opponent to create a polynomial size counter, effectively simulating the shortest path victory condition for 1-player. This is relevant for cases like the optional door-opening gadget in Section 2.3.6 where it is in P for the 1-player reachability game, but NP-complete for the 1-player shortest path version. Since we do not currently have a PSPACE-completeness proof for the model with optional door opening (and some reason to suspect it may not be PSPACE-hard) this simple timing reduction gives us the currently best known bound of NP-hardness for 2-player reachability motion planning with the optional door opening gadget.

This also shows another case where the visibility model for the Team Imperfect Information problem is relevant. If we chose a stronger visibility model, one in which we could freely assign gadgets which can be monitored by players or one in which locations gave an arbitrary visibility mapping to other gadgets, then there would be a trivial reduction from Team Imperfect Information to 2-player motion planning by giving the players perfect information. However, when we restrict visibility to be along location connectivity, as in the case we explore, giving the players perfect information would require connecting all gadgets together. Although this modification seems unlikely to reduce the computational complexity of the problem, it certainly seems like an obstacle in motion planning cases like those involving the 1-toggle, where verifying an unknown state of the gadget will give the player the ability to change the gadget (depending on the state).

3.2 Multi-Player Reversible Deterministic Gadgets

Here we revisit k-tunnel reversible deterministic gadgets in the 2-player and team imperfect information models. We reduce from 2-player Constraint Logic and Team Private Constraint Logic which are natural multi-player versions of Non-deterministic Constraint Logic which

was the target problem for 1-player motion planning with k-tunnel reversible deterministic gadgets. Since these proofs essentially take that reduction and modify it to deal with the new complications which arise in these models, it is recommended that the reader review the reduction in Section 2.2 before reading this section in depth. Section 3.2.1 will show EXPTIME-completeness for 2-player motion planning with k-tunnel reversible deterministic gadgets that contain interacting tunnels. Section 3.2.2 will show RE-completeness for Team Multiplayer Imperfect Information motion planning with k-tunnel reversible deterministic gadgets that contain interacting tunnels.

3.2.1 2-Player Unbounded Motion Planning

In this section, we analyze 2-player motion planning games with k-tunnel reversible deterministic gadgets. We show that any such game which includes a gadget with interacting-tunnels is EXPTIME-complete. We do so by a reduction from 2-player unbounded constraint logic, allowing us to reuse some of the work in the prior section. In addition to building the single player AND and OR vertices, we show how to adapt the gadgets to allow different players to have control of different edges. We also build up the needed infrastructure to enforce turn taking in the simulated game.

The construction of crossovers using interacting-k-tunnel reversible deterministic gadgets with two states should allow one to show hardness for the planar version of this problem with those gadgets and any others that simulate them. Care must be taken with the layout, timing, and interaction between crossovers so we do not go on to prove such a result in this thesis. Unfortunately, the crossover created by the locking 2-toggle in Section 2.2.4 does not suffice and thus leaves the question partially open. In addition, the question of noninteracting-k-tunnels reversible deterministic gadgets has not been resolved. We are not able to show problems with such gadgets are easy, and Section 3.3.1 suggests they should be at leastPSPACE-hard.

Lemma 88. *2-player motion planning with any set of gadgets is in EXPTIME.*

Proof. A configuration of the maze consists of the state of each gadget and the location

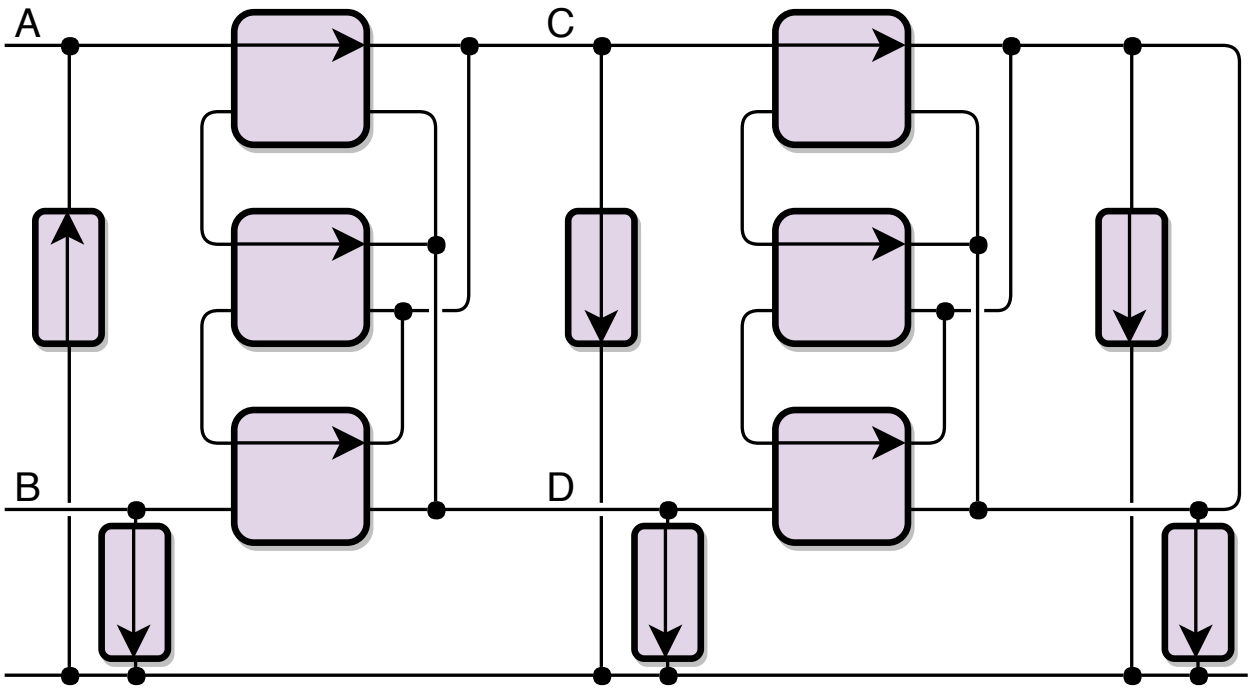


Figure 3-1: The timer gadget used in the 2CL reduction, made of PL2Ts and 1-toggles. In order to travel between A and B, a player must travel between C and D three times. The timer can be extended to the right; two iterations are shown.

of the robot, and has polynomial length. There is a polynomial-space alternating Turing machine which nondeterministically guesses moves for each player and keeps track of the configuration, using existential quantifiers for player 1 and universal quantifiers for player 2. This Turing machine accepts exactly when player 1 has a forced win. Thus the problem is in APSPACE = EXPTIME. □

Theorem 89. *2-player motion planning with the locking 2-toggle gadget is* EXPTIME*-complete.*

Proof. This game is in EXPTIME by Lemma 88. We use a reduction from 2-player Constraint Logic (2CL) to show EXPTIME-completeness. See Section 1.4.1 for a definition of 2CL.

We begin by describing a timer gadget, shown in Figure 3-1. Suppose one player has access to the bottom line. They can enter the gadget at A, and begin going through the timer, eventually reaching a victory gadget at B. The timer has two key properties:

1. Reaching B takes a number of transitions exponential in the size of the timer. In order to get from A to B, the player goes though the top PL2T to C, recursively travels from C to D, goes around the loop through the top two PL2Ts, goes back from D to C, traverses the bottom loop, once again goes from C to D, and finally proceeds to B. If traveling between C and D takes m transitions, then traveling between A and B takes $3m + 6$ transitions. If the timer gadget is repeated k times, it takes at least 3^k transition to get from A to B.

2. A player in the timer has an opportunity to exit the timer at least every 2 turns, and exiting takes 1 turn; in particular, they can always exit within 3 turns while progressing the timer. The player uses a 1-toggle to exit to the bottom line. They can then later reenter using the same 1-toggle, resuming their work on the timer where they left off. If the player is in the timer, the next step in progressing the timer is either traversing a loop between to PL2Ts, which takes 2 transitions, or moving horizontally between timer segments, which takes 1 transition. Thus in 3 transition, the player can complete the current or next step and exit to the bottom line.

The constraint logic gadgets are similar to those used in Theorem 8 for the 1-player game, with the modification shown in Figure 3-2. We have added 1-toggles allowing a player at an edge to visit and configure the incident vertices, without allowing the player to travel to other edges. Each player's goal location is inside the gadget corresponding to their target edge, so that they can reach it if they can flip the edge.

Unlike the 1-player version, we need gadgets to enforce the turn order. The overall construction is shown in Figure 3-3. The maze consists of three main regions: the Existential area, the Universal area, and the constraint logic. Each player will spend most of their time in their own area, occasionally entering the constraint logic to flip an edge. The players' areas are designed to enforce turn order and progression of the game. A player can never enter the other player's area.

There is a single L2T separating the constraint logic area from each player's area. This prevents both players from being in the constraint logic at the same time.

Each player's area contains an edge selection gadget, which consist of a locking 2-toggle for each edge they can control. The other line in the L2T is accessible by entering the constraint logic area and passing through a delay line composed of four 1-toggles, and is connected to the corresponding edge gadget. In order to access an edge gadget, the player must activate the appropriate L2T, which requires deactivating the previously activated L2T. This ensures that only one edge gadget is accessible by each player at any time. There is a 1-toggle separating the edge selection gadget from the rest of the player's area, so that switching the selected edge requires at least 4 turns (we use one tunnel of a L2T for a 1-toggle).

Each player's area has a timer, of length t_E for Existential and t_U for Universal. If a player finishes their timer, they win.

Each player begins inside their edge selection gadget, and Existential goes first. The game begins with Existential picking an edge and going to the constraint logic area, while Universal goes to their timer.

A round of normal play proceeds as follows:

- Existential moves from edge selection to the constraint logic area. Universal is currently in their timer.

- Existential enters the constraint logic, walks to their selected edge, and flips it. Universal continues working on their timer.

- Existential returns through their constraint logic delay line. Once they pass the first 1-toggle, Universal finishes their current step in the timer and exits, moving towards edge selection.

- Existential begins working on their timer. Universal selects an edge, enters the constraint logic, and flips the edge.

- Universal returns through their constraint logic delay line. Once they pass the first 1-toggle, Existential exits their timer and moves to edge selection.

- Existential selects an edge as Universal enters their timer.

Suppose Universal has just flipped an edge gadget; they have nothing to do but return through the delay line of length 4. When Universal is past the first 1-toggle, Existential will leave their timer to flip an edge. Universal might try turning around to go back to the constraint logic area. It takes Universal at least 6 turns to flip the edge back, during which Existential has enough time to select an edge and reenter their timer. The game is now in the same situation as before, except that Existential has progressed their timer; thus Universal does not want to do this.

Universal might instead try waiting at the central L2T after Existential has selected an edge. Existential will then go to their timer, forcing Universal to exit eventually. When Universal is not next to the central L2T, Existential exits their timer and moves to constraint logic. Because of the 1-toggle separating edge selection from the central L2T, for Universal to change their selected edge, they must spend multiple turns away from the L2T, allowing Existential to enter constraint logic; similarly if Universal works on their timer, Existential can enter constraint logic. So Universal has no choice but to pass the turn to Existential.

Since Existential can always exit their timer within 3 turns, and Universal has three more 1-toggles to get through when Existential begins looking to exit, Existential will reach edge selection before Universal can reach edge selection, so Existential will be the first player ready to enter constraint logic again. Nothing Universal can do will prevent Existential from taking the next turn in the 2CL game. Similarly after Existential flips an edge, Universal will be able to take a turn next. So either player can force the alternation of constraint logic turns.

The sizes of the timers are chosen to satisfy the following. First, if Existential cannot win the constraint logic game, Universal should win, so Universal's timer is shorter: $t_U < t_E$. Second, if Existential can win the constraint logic game, Existential should win first, even if Universal ignores the constraint logic game and just works on their timer. If the constraint logic graph has n edges, it takes at most 2^n constraint logic turns for Existential to win. Each constraint logic turn for Existential takes 6 turns to select an edge and return to the

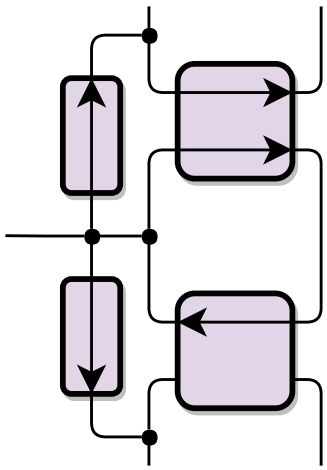


Figure 3-2: A modified edge gadget for the 2CL reduction. A player can visit the vertex gadgets attached to the edge gadget, and then return to the edge gadget.

constraint logic, 8 turns to cross the constraint logic delay line twice, 4 turns to access and flip an edge, and up to 5 turns to access and configure an incident vertex, so 25 turns in total during which Universal can work on their timer. Both players might be in their timers simultaneously at most 4 times each cycle, and each time for at most 4 turns, so Universal spends at most 41 turns in their timer for each constraint logic turn. Thus, since it takes Universal at least 3^{t_U} turns to win through the timer, we need $41 \cdot 2^n < 3^{t_U}$; $t_U = n + 6$ suffices, and we can set $t_E = 2n + 12$.

Using these timer sizes, it is clear that the constraint logic game will resolve before either timer if the players follow normal play. We need the timers so that Universal cannot force a draw by sitting in the constraint logic forever, preventing Existential from winning; Existential will progress on their timer if Universal attempts this.

Hence Existential has a forced win in the motion planning game if and only if they have a forced win in the constraint logic game. Since 2CL is EXPTIME-complete, the 2-player game on systems of locking 2-toggles is EXPTIME-hard. The maze used in the reduction has only $O(n)$ L2Ts. □

Theorem 90. *2-player motion planning with any interacting-k-tunnel reversible deterministic gadget is EXPTIME-complete.*

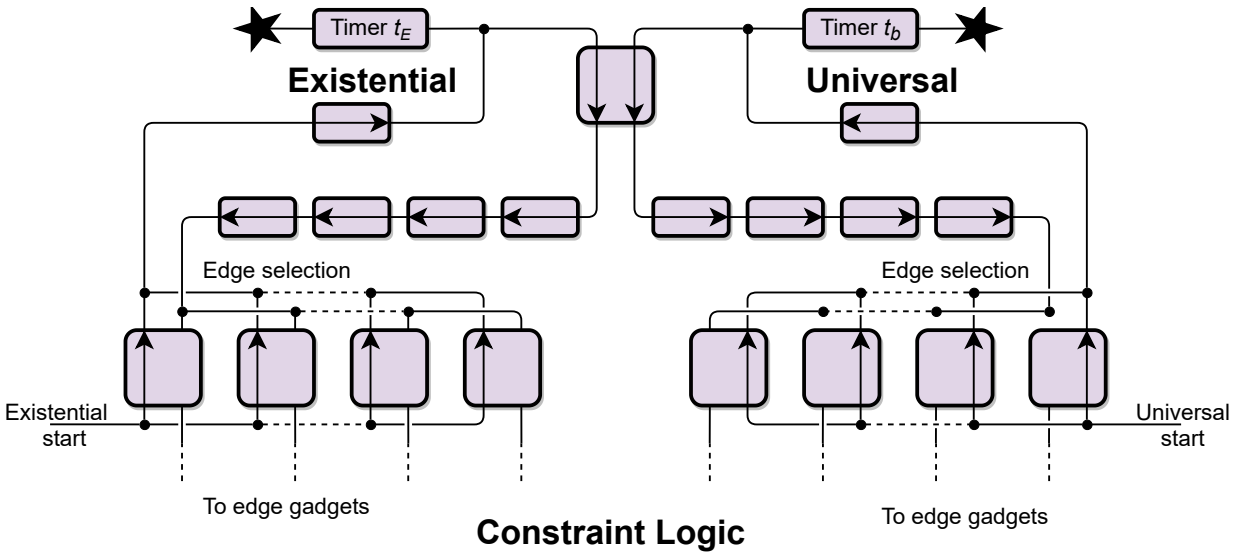


Figure 3-3: The overall structure and turn enforcement gadget. Each player's edge selection area has a L2T for each edge that player can flip; four are shown for each player. The bottom line from each such L2T connects to the corresponding edge gadget. The timers are as shown in Figure 3-1, with t_E and t_U repetitions. The inside connection to each timer is connected to its access line, and the outside connection (to a win gadget) is at U in Figure 3-1. The goal location past each timer is for the player whose side it is on.

Proof. This game is in EXPTIME by Lemma 88. We adapt the 2CL reduction in the proof of Theorem 89. Replace each locking 2-toggle in that 2CL reduction with the simulation of a locking 2-toggle from the arbitrary gadget in Theorem 5. In the new maze, each tunnel in a simulated L2T takes 6 transitions to traverse, so the game goes 6 times slower.

The simulation still works with two players, as long as both players do not have access to the gadget at the same time. Each L2T in the turn enforcement area is accessible only by one player, and only one player can be in the constraint logic area at any time. The only L2T both players have simultaneous access to is the central gadget which gives access to the constraint logic area, so we look more carefully at that gadget.

The state with both edges traversable is shown in Figure 2-5 (the 1-toggle simulation still works). Note that the simulation is of an APL2T, but the gadget in the 2CL reduction is a PL2T; this is not a problem because we are not concerned with planarity. Suppose both players approach the gadget, one from the right on the top line and one from the left on the bottom line. Whoever reaches the gadget first should "win the race," and lock out the

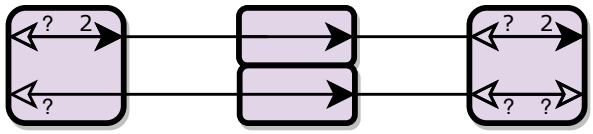


Figure 3-4: Another state of the construction shown in Figure 2-5. The leftmost gadget is in state 1, and the rightmost gadget is in state 3.

other player. The simulation implements this correctly, provided that the player who arrives first is a full turn ahead in the L2T maze, or 6 turns ahead in the new maze. The only time the players might be within 6 turns of each other is at the very beginning of the game, so we put a delay of 6 turns for Universal to get from their start location to edge selection to ensure Existential wins the race by 6 turns. If a player would arrive less than 6 turn before the other player, they should go to their timer instead; since this is a zero-sum game and the players would have to collaborate to break the simulation, one player will choose not to.

The other way players can interact at this gadget is when one player is exiting the constraint logic area, and the other player is waiting just outside and enters as soon as they can. The state of the simulation is shown in Figure 3-4 (the other possible state is symmetric). One player, say Existential, has traversed the top edge to enter the constraint logic area, and is about to exit by traversing the top line to the right. Universal is waiting at the left end of the bottom line, ready to enter the constraint logic area. The leftmost gadget prevents Universal from making any transitions until Existential begins exiting. Once Existential begins exiting, the leftmost gadget switches to state 2, so Universal can follow parallel to Existential and one turn behind. As long as Existential continues through the construction at full speed, Universal interacts with the construction as though Existential has already finished their traversal, so it correctly simulates a L2T. Again breaking the simulation would require the players to cooperate, and the game is zero-sum, so at least one player will ensure the simulation works. □

3.2.2 Team Unbounded Reversible Deterministic

In this section, we show that team imperfect information games with interacting-k-tunnel reversible deterministic gadgets is RE-complete, implying the problem is Undecidable. The reduction is from Team Private Constraint Logic (TPCL); see Section 1.4.1 for a definition. We use many of the ideas and constructions from Section 3.2.1, but various modifications are needed to deal with the additional player and the model of player knowledge. Recall in this model we have three players on two different teams, each controlling a single robot. All players start knowing the configuration of the entire game; however, after that point players can only observe the states of the gadgets that their robots can reach via the connection graph. Adaptations for the planar version and the complexity of such games with noninteracting-tunnel gadgets remains open as in Section 3.2.1.

Lemma 91. *Team motion planning with any set of gadgets is in RE (recursively enumerable).*

Proof. Suppose the Existential team has a forced win on some system of gadgets, and consider the tree of possible positions when Existential follows their winning strategy. The branches in the tree correspond to choices the Universal team might make. Since Existential forces a win, every branch of the tree is finite. Since Universal has finitely many choices at each turn, the tree is finitely branching. Since our tree does not contain an infinite degree vertex or an infinite length path, then by Kőnig's infinity lemma [48] the tree is finite. In particular, there is a finite bound on the number of turns it takes for Existential to win, so the winning strategy can be described in a finite amount of space. So there are countably many potential winning strategies, and we can sort them lexicographically.

Given a potential winning strategy, the problem of determining whether it is actually a winning strategy is decidable: an algorithm can explore every choice Universal might make, and see whether Existential always wins. There are only finitely many choices to check because the strategy only describes a finite number of turns.

We use the following algorithm to determine whether Existential has a forced win. For each potential winning strategy in lexicographic order, check whether it is a winning strategy.

If it is, accept. This algorithm accepts whenever Existential has a forced win, and runs forever otherwise, so it recognizes the games in which Existential has a forced win. □

Although [27] only mentions undecidability and notRE-completeness, it follows that TPCL isRE-complete. Containment in RE is given by an argument nearly identical to the proof of Lemma 91. The proof of undecidability is ultimately by a reduction from acceptance of a Turing machine on an empty input, which isRE-complete, implying that TPCL isRE-hard.

Theorem 92. *Team motion planning with the locking 2-toggle gadget is*RE*-complete (and thus undecidable).*

Proof. Containment in RE is given by Lemma 91. ForRE-hardness, we use a reduction from TPCL, with a similar construction as in the proof of Theorem 89. The overall construction is shown in Figure 3-5. Capital letters label L2Ts, and lowercase letters label lengths of delay lines. The two tunnels in the same L2T are labelled the same, instead of being positioned next to each other. The three players U, E_1, and E_2 each have their own region. Each region contains an edge selection area with k edges initially active, access to the constraint logic, and some additional gadgets. We need to ensure the following:

1. Turn order is enforced. That is, the players take turns in the order U, E_1, E_2, and neither team can gain anything by deviating from this. We use L_1 and L_2 to prevent U from being in the constraint logic area at the same time as E_1 or E_2, and appropriate delays to ensure each player is ready for their turn. The timer in E_2's region forces U to eventually pass the turn to E_1.

2. Each player can flip up to k edges each turn. If k edges are initially accessible for each player, the edge selection area allows them to select any k of their edges, and a player must end their turn in order to change their selection.

3. The Existential players have the correct information about the state of the game. Each of them has a visibility area, which allows them to see the orientation of the appropriate

constraint logic edges. We must not allow E_1 and E_2 to both access the same L2T, as they could then use it to communicate. So we need a more complicated mechanism to prevent both Existential players from being to the constraint logic area at the same time.

For visibility, we modify the edge gadget as shown in Figure 3-6. The appropriate line is connected to each Existential player's visibility area if they should be able to see that edge.

A round of normal play proceeds as follows:

- U begins their turn by passing down through L_1 and L_2. E_1 waits next to V, and E_2 walks through their timer.
- U flips some edges, and returns, passing V. When E_1 sees this happen, they go to their visibility area, and then select k edges. E_2 continues in the timer.
- U finishes exiting through the delay b. Once U has passed L_1, E_1 enters the constraint logic area. E_2 reaches the end of the timer, finds S to be closed, and comes back.
- U is stuck on the side of L_1 away from the constraint logic area, and can select edges. E_1 flips edges and returns to just below L_1. E_2 goes to their visibility area, and then selects edges.
- After a number of turns large enough that both Existential players are definitely ready, E_1 exits L_1. The same round, E_2 enters L_2, passing the turns from E_1 to E_2.
- E_2 takes their turn. U waits just to the right of L_2, and E_1 waits above X.
- E_2 exits L_2 and goes to the timer. U passes through L_2 to take their turn, and E_1 waits.

We place each player's starting location to be at the end of a chain of 1-toggles leading to their region, so they arrive after an appropriate delay. We can set U to have no delay and E_1 and E_2 to have $2k$ delay, so U has time to select edges before the Existential players

arrive. The first turn has slightly strange timing since E_2 starts the timer later than normal, but this is not important.

We consider ways in which player might deviate from normal play, and see that in each case they do not gain anything by deviating.

U enters the constraint logic through L_2 as soon as E_2 passes L_2 on their way out, at which point E_2 enters the timer. U need to be able to take a full turn and go back through S before E_2 reaches the end of the timer; this takes up to $2(b+c+2)+2+11k$ turns, since flipping each edge now takes up to 11 turns. So we need $t > 2(b+c+2)+2+11k$. The timer forces U to return through S within $t+2$ turns, since otherwise E_2 wins.

The gadget V lets E_1 know when U is done, since E_1 can see whether U is past V while waiting at L_1. Specifically, E_1 waits until they see U stay past V for $2c$ turns, and then return. For U to be unable to flip edges after this, we need $4c > t$. Then E_1 goes to visibility and sees the current configuration, selects k edges for their next turn, and waits at L_1 again. For E_1 to have time to do this before U gets out, we need $b > 2k+2$.

Once U exits L_1, E_1 goes in and flips edges. The delay d ensures that if E_1 (or E_2) flips any edges, then U will be ready for their next turn; we need $2d > 2k+4$. E_2 returns through the timer, checks visibility, and selects edges. If E_2 enters constraint logic before E_1 leaves, U can win through X and Y, so E_2 must wait until E_1 leaves. The Existential players coordinate using the fact that the length of an entire round is bounded, so they can wait long enough to ensure that they are both ready, and then E_1 exits X immediately before E_2 enters Y. Since E_1 was past L_1, U is locked outside of L_1, so E_2 can get past L_2; the E_1 can safely pass the turn to E_2.

While E_1 is past X, U might try going through Z and X, trapping E_1. In this case, E_2 can win through Z, so U will only go through Z if both X and Y are traversable.

During E_2's turn in the constraint logic, E_1 must not be past X to prevent U from winning through X and Y. So U can go through L_1, and go through L_2 as soon as E_2 exits. That is, E_2 cannot pass the turn back to E_1.

E_2 might try to stay in the timer, forcing U to stay out of the constraint logic to prevent

E_2 from winning through S. Then E_1 might be able to take extra turns in the constraint logic. If the Existential team attempts this, U will win through P and Q. If U goes through R and P when Q is not traversable in order to trap E_1, E_2 will win through R; these three L2Ts are analogous to X, Y, and Z.

Assuming the constraints mentioned are satisfied, no player or team can usefully deviate from normal play, and normal play simulates the TPCL game. Thus Existential has a forced win in the team motion planning game if and only if they have a forced win in the TPCL game.

We can satisfy all the constraints, e.g by $b = 2k+3$, $c = 8k+7$, $d = k+3$, and $t = 31k+27$ (the constraints are not tight, but they suffice). The number of L2Ts in the resulting system of gadgets is only linear in the number of edges in the constraint logic graph. □

Theorem 93. *Team motion planning with any interacting-k-tunnel reversible deterministic gadget is*RE*-complete.*

Proof. Containment in RE is given by Lemma 91. ForRE-hardness, we adapt the TPCL reduction in Theorem 92 to work for the arbitrary gadget. As in the 2-player case of Theorem 3.2.1, it is almost sufficient to replace each L2T with the simulation in Theorem 5. We examine the L2Ts that are shared between two players.

First, L_1 and L_2 are analogous to the central L2T in Theorem 3.2.1: if two player are racing to enter, the player who should win is at least 6 turns ahead, and if one player exits and another enters, is works correctly.

For S, P, Q, R, X, Y, and Z, we use a single copy of the arbitrary gadget with 5 extra gadgets for delay, instead of the simulation. Considering the gadget as in Figure 2-2, we use state 1, and put the bottom edge in the position next to a win gadget. For S, Q, Y, R, and Z, if the bottom edge is traversed from state 2, the game is over, so the gadget is never in a state other than 1 or 2 while the game is going. For P and X, we know that U cannot safely wait past those gadgets, so the game must be about to end in Universal victory if they ever reach state 3.

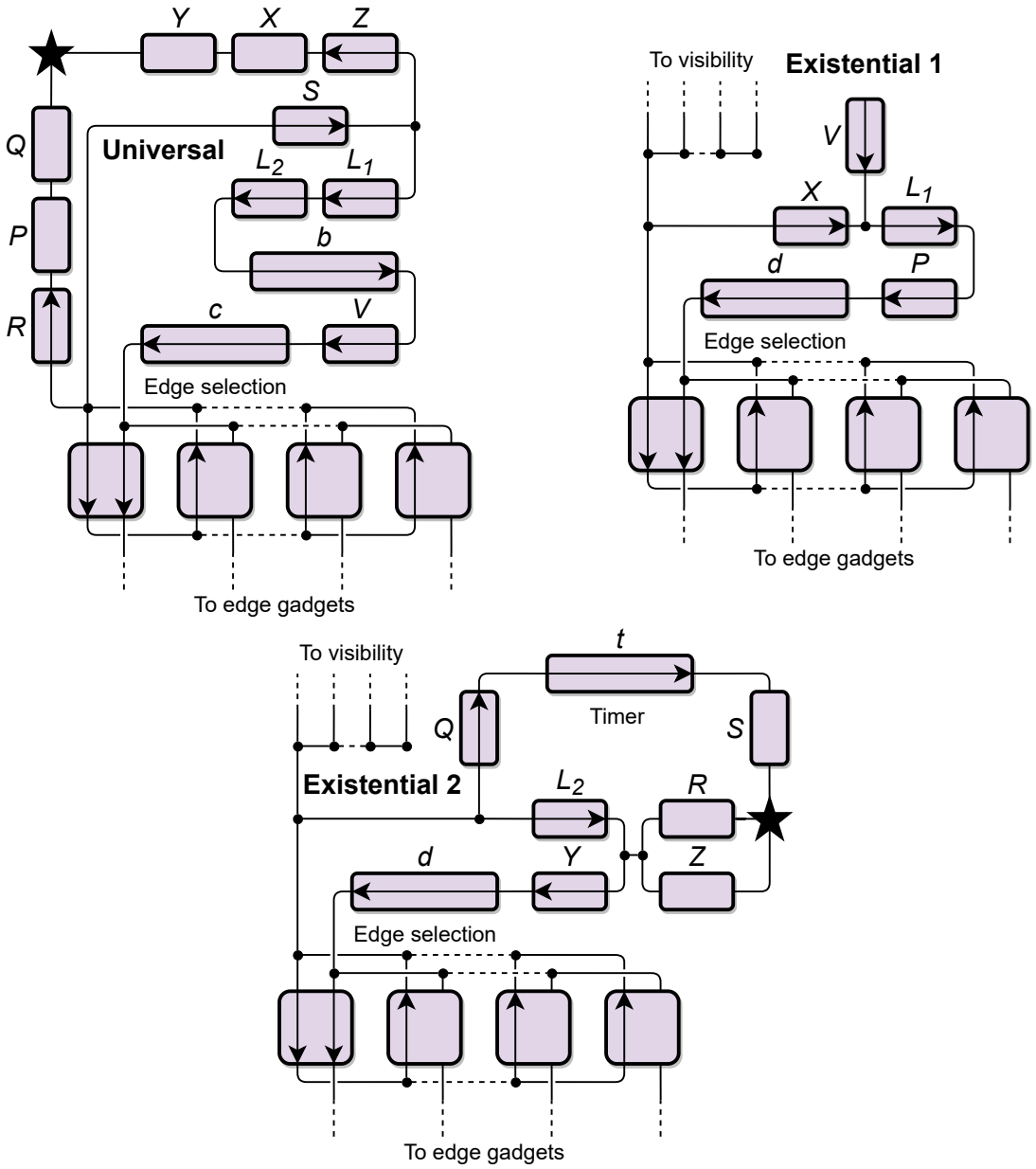


Figure 3-5: The turn enforcement gadget for the team game. Each player has their own region which contains an edge selection area, a path to the edge gadgets they can control, and some other constructions. Each Existential player has a visibility area which allows them to see the state of some edge gadgets in constant time. There is no good layout for the whole gadget, so we use pairs of 1-toggles that share a (capital) label to represent L2T. Long boxes with lowercase labels represent chains of 1-toggles with length given by the label. The win gadgets are for the obvious players, and the tunnels currently not traversable (P, Q, R, S, X, Y, and Z) will directed toward the win gadget when they become traversable.

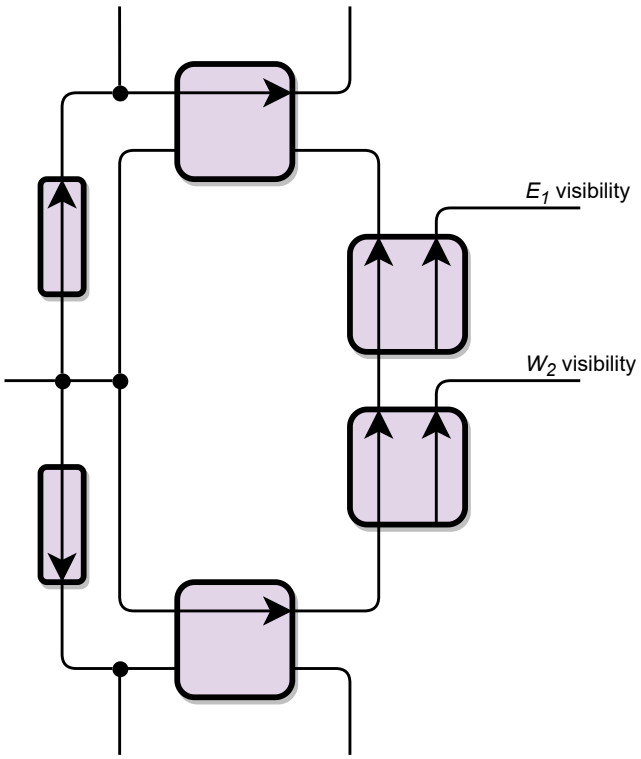


Figure 3-6: An edge gadget for the TPCL reduction. This is the same as a 2CL edge gadget, except two L2Ts have been added that allow E_1 or E_2 to see the state of the edge if it is connected to their visibility area, but they cannot make any transitions.

For V and the visibility gadgets on edges, we use the construction in Figure 3-7. U has three paths to choose from in the process of crossing the bottommost 1-toggle, and always two of them are align with that 1-toggle, so U has two options. The Existential player, say E_1 can see the state of a gadget in all three paths, and thus determine the orientation. If E_1 goes through one of these gadgets, U will use the other path. If there were only one path, E_1 could go through the gadget, forcing U to either not flip that edge or get a gadget into an unknown state (for L2Ts, we used the fact that E_1 could never traverse that tunnel in one direction). This visibility gadget allows E_1 to see the orientation of a constraint logic edge or V without being able to interfere.

Once we make these replacements, the new maze with the arbitrary gadget has a forced win by Existential if and only if the maze with L2Ts did. □

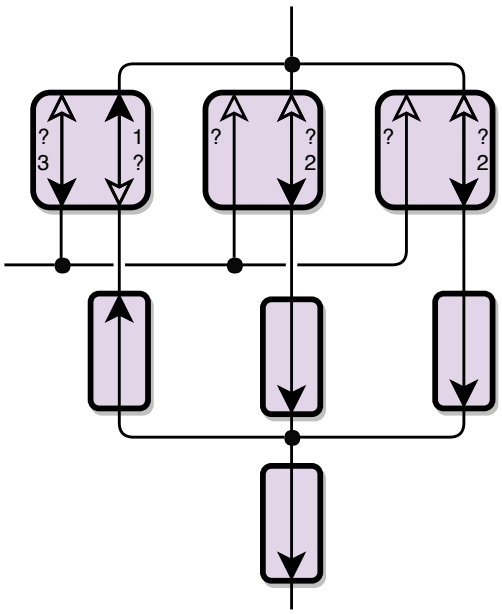


Figure 3-7: A visibility gadget for the TPCL reduction. The Universal player can travel between the top and bottom, and a Existential player can enter the side to see which direction was traversed most recently.

3.3 Multiplayer Polynomially Bounded Gadgets

In this section we give a full classification for DAG gadgets in the 2-player model, Section 3.3.1, and the team imperfect information model, Section 3.3.2, by showing the single use gadget (and thus all non-trivial DAG gadgets) are hard in those models. The reductions resemble a partisan variation on Generalized Geography. These results also provide lower bounds for many LDAG gadgets, however, a characterization there seems much more difficult to achieve. The upper bounds are no longer obvious as the number of moves in these games is no longer bounded. Further, examples like optional door opening gadgets seem unlikely be to hard as the players have no incentive to open traversals for their opponent unless they themselves will need to traverse that tunnel to win.

3.3.1 2-Player Bounded Motion Planning

In this section, we show that it is PSPACE-complete to decide who wins in a 2-player race with any nontrivial DAG gadget (having at least one transition). To do so we give a

construction that shows hardness for single-use paths and single-use one-way gadgets by a reduction from QBF. A simpler construction is possible, but this construction is more easily adapted to the team game in Section 3.3.2. This gives us a nice example of the 2-player local motion planning problem fitting into the canonical complexity class for two-player bounded games. It is also of interest because of how incredibly simple this gadget is. Two-location gadgets trivially do not have interacting tunnels (there is no other tunnel to interact with) and thus the 1-player version of these problems are contained in NL by Theorem 2.

Lemma 94. *2-player motion planning with any set of DAG gadgets is in* PSPACE*.*

Proof. Since each gadget can undergo only a polynomial number of transitions, the length of the game is polynomially bounded. An alternating Turing machine which uses $\forall$ states to pick Universal's moves and $\exists$ state to pick Existential's moves can simulate the game in polynomial time, so the motion planning problem is in AP = PSPACE. □

Lemma 95. *2-player motion planning with the single-use bidirectional gadget is*PSPACE*-complete.*

Proof. Containment inPSPACE follows from Lemma 94. ForPSPACE-hardness, we reduce from quantified boolean formulas (QBF). See Section 1.4.2 for a definition of QBF.

We begin by describing the gadgets used in the reduction. The variable gadget is shown in Figure 3-8. Most of the gadget consists of two branches, corresponding to a variable and its negation. Each branch has a series of forks separated by single-use paths. There will be a number of forks depending on the number of occurrences of a literal in the formula; two forks are shown. Each side of each fork has two single-use paths in series. The game will be constructed so that Existential always prefers the top side of a fork to be traversable, and Universal prefers them to be not traversable; the top of a fork will be used later in evaluating the formula.

During the game, both players will pass through each variable gadget, with one player taking each of the two branches. Existential will take the bottom side of each fork on their branch, and Universal will take the top side. Afterwards, only the branch which Existential

took will have forks whose top sides are traversable. Thus we consider the assignment of the variable to be the literal corresponding to the branch Existential takes.

Suppose both players are at the left end of a variable gadget, and it is Player 1's (who may be Existential or Universal) turn. Player 1 picks a branch, and Player 2 must walk down the other branch. Player 1 arrives at the right end of the branches immediately before Player 2. If Player 1 proceeds along the bottom path, Player 2 wins, so Player 1 must take the top path, which takes one turn longer. After traversing the variable gadget, both players are at the right end, and it is Player 2's turn, so the other player gets to choose a branch in the next variable gadget.

The clause gadget is shown in Figure 3-9. There are three paths from the left end to the right end, corresponding to the literals in a clause. Each path goes through a fork in a variable gadget. After variables are assigned, the single-use paths on each end of the fork are used, as are either those on the top or those on the bottom of each fork. If the top single-use paths are used, that path through the clause gadget is blocked, and if the bottom paths are used, that path is open. Existential will ultimately win by traversing each clause gadget, so Existential prefers to use the bottom side of a fork, and Universal prefers to use the top side.

Each path has a large amount of delay (gadgets in series) before and after the fork, so that trying to use the clause gadget during variable assignment results in losing before reaching the end of the delay.

The race gadget is shown in Figure 3-10. It ensures both players proceed though variable gadgets as fast as possible. Let Player 1 be the player who reaches the race gadget first in this situation, immediately before Player 2; they are also the player who did not pick the assignment of the last variable. If Player 1 takes the bottom path, Player 2 will win, so Player 1 takes the top path. Then Player 2 takes the bottom path, and now the two players have been separated.

If Player 1 arrives more than a turn ahead of Player 2, they can take the bottom path. The next turn, before Player two can do anything at the race gadget, Player 1 wins. If Player 2 reaches the race gadget first, they can take the top path and win.

Given a quantified boolean formula with V variables and C clauses, we construct a system of gadgets as follows. We assume the QBF has alternating quantifiers beginning with $\exists$. There is a series of variable gadgets connected end-to-end corresponding to the variables of the formula, in the order of quantification. The goal location inside each variable gadget is a win for alternating players, beginning with Universal. The branches of the variable gadget corresponding to x correspond to the literals x and $\neg x$. Each branch of that variable gadget has enough forks that each instance of a x or $\neg x$ in the formula corresponds to a fork, and the two branches have the same number of forks.

There is a clause gadget for each clause in the formula, connected in series. The three branches of a clause gadget correspond to the three literals in the clause. Each branch goes through the fork in the appropriate variable gadget corresponding to that instance of the literal. The delay before and after each fork consists of $9C + 3V$ single-use paths. The right end of the last clause is connected to a Existential goal location.

A race gadget is connected to the right end of the last variable gadget, with the goal locations such that Player 1 is the player with a win gadget inside the last variable gadget. The path with a Existential win gadget, which Universal will walk down, is followed by $C(18C + 6V + 1) + 2$ of single-use paths in series leading to a Universal win gadget. The other path, which Existential will walk down, is connected to the first clause gadget.

Both players begin at the left end of the first variable gadget, and Existential goes first.

The game begins with Existential choosing a branch of the first variable gadget, corresponding to a choice of variable, and Universal taking the other branch. Then Universal chooses a branch of the second variable gadget, choosing the assignment of the variable based on the path Existential is forced to take. The players continue to take turns assigning variables. If either player deviates from this, such as by going into the delay in a clause gadget or by going backwards along another path, the other player will reach the race gadget first and win; the delay in clause gadgets is long enough to ensure that they do not have time to get through the clause gadget before losing. Otherwise both players arrive at the race gadget, and are sent down different branches.

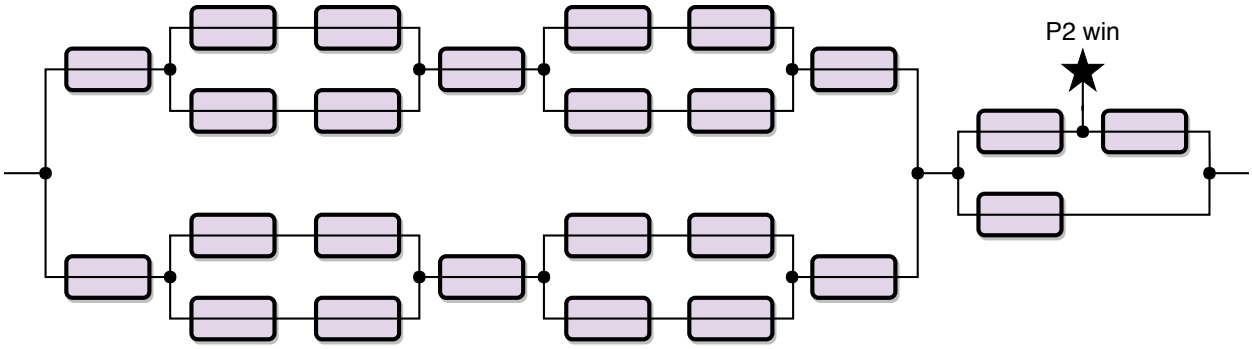


Figure 3-8: A variable gadget. The players arrive at the left, each take one path across, and exit at the right.

Existential then proceed through each clause in series. Each branch of a clause is traversable if and only if the corresponding literal is true (since Existential took the bottom side and Universal took the top side of each clause). The single-use paths between forks ensure that Existential cannot do anything other than progress through each clause gadget. If the formula is satisfied, Existential has a path through the clauses, and wins after $C(18C + 6V + 1)$ turns. If the formula is not satisfied, Universal, who is walking down their long path, wins after slightly longer. Thus Existential has a forced win if and only if the quantified formula is true. □

Lemma 96. *2-player motion planning with the single-use one-way gadget is*PSPACE*-complete.*

Proof. We again reduce from QBF. In the reduction in Lemma 95, neither player ever has to move through a single-use gadget to the left. Thus we can replace each bidirectional single-use gadget with a one-way single-use gadget pointing to the right, and the reduction still works. □

Corollary 97. *2-player motion planning with any nontrivial DAG gadget is*PSPACE*-complete.*

Proof. As noted in Section 2.3 all DAG gadgets contain a single-use transition. This can be bidirectional or one-way, which are both shown to bePSPACE-hard in Lemmas 95 and 96. Containment inPSPACE is given by Lemma 94. □

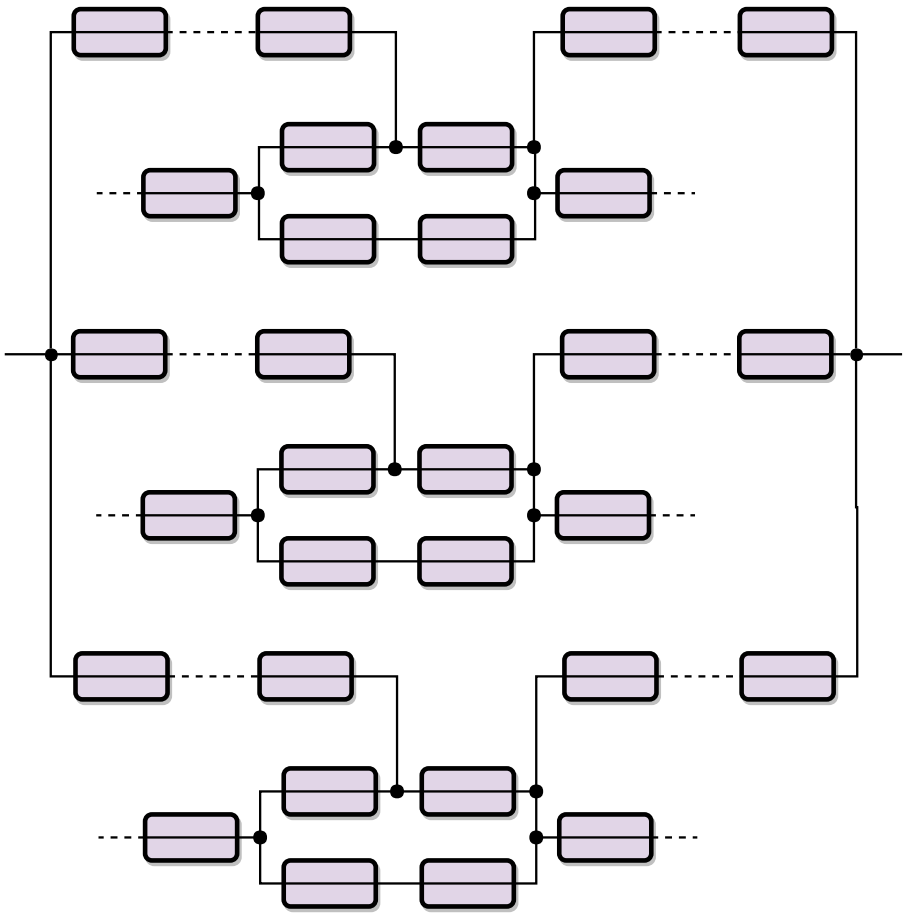


Figure 3-9: A clause gadget. Each literal is also part of a variable gadget. Each branch has a long series of gadgets so that it takes a large amount of time to traverse.

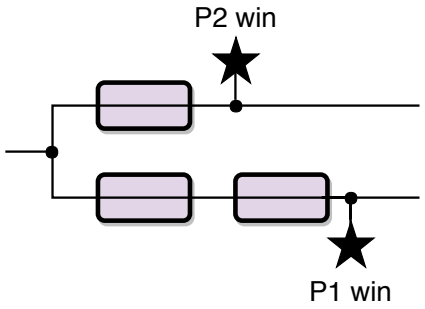


Figure 3-10: A race gadget. If Player 1 arrives at the left immediately before Player 2, each player ends up on one of the right exits. Otherwise, the player who arrives first wins.

3.3.2 Team Bounded Motion Planning

In this section we characterize the complexity of team imperfect information motion planning games with DAG gadgets. Since DAG gadgets are inherently bounded, the problem is in NEXPTIME, shown in Lemma 98. We go on to show in Lemma 99 that any nontrivial DAG gadget is NEXPTIME-complete by first giving a reduction from dependency quantified boolean formula (DQBF) for the single-use gadget. We then show that this proof adapts for single-use one-way gadgets. Since all DAG gadgets with at least one transition contain at least one of these, we achieve hardness for all such DAG gadgets.

Lemma 98. *Team motion planning with any set of DAG gadgets is in NEXPTIME.*

Proof. A ***partial history*** for a player is the sequence of visible gadget states and moves made by that player, up to some point in the game. A ***strategy*** is a family of functions, one for each Existential player, that assign to each possible partial history a legal move from the position at the end of the partial history.

Since the gadget is a DAG, the game lasts a polynomial number of turns. Each player has polynomially many choices for each move, so there are only exponentially many possible sequences of moves, and only exponentially many possible partial histories for each player. Thus a strategy can be written in an exponential amount of space.

To determine whether Existential has a forced win in the team game, first nondeterministically pick a strategy. Then, for each possible sequence of moves the Universal players could make, simulate the game with the Existential players following the strategy. If Universal ever wins, reject; if Existential always wins, accept. This nondeterministic algorithm accepts if and only if there is some strategy Existential can use to force a win. The algorithm runs in exponential time because there are exponentially many sequences of moves the Universal players might make, and the game for each such sequence takes a polynomial amount of time to simulate. Thus the algorithm decides the team game on systems of the gadget in NEXPTIME. □

Lemma 99. *Team motion planning with the single-use bidirectional gadget is NEXPTIME-complete.*

Proof. Containment in NEXPTIME follows from Lemma 98. For NEXPTIME-completeness, we reduce from dependency quantified boolean formulas (DQBF). See Appendix 1.4.2 for a definition of DQBF. In this reduction Existential represents the existential variables and Universal represents the universal variables.

The reduction uses the same gadgets as that in Lemma 95, except that the clause gadget is modified as in Figure 3-11. This allows the Existential player checking the formula to try each literal, and return to the start of the clause gadget if the literal is false. This is necessary because the Existential player cannot see the state of the literals until arriving at them. For variable gadgets, we do not include the portion with a win gadget for Player 2 (the rightmost quarter or so in Figure 3-8), since we no longer want players to alternate choosing variables.

We construct the system of gadgets as follows. The overall structure is shown in Figure 3-12. For each set of variables $\vec{x}_1$, $\vec{x}_1$, $\vec{y}_1$, and $\vec{y}_2$, there is a corresponding set of variable gadgets (without the win gadget component) connected in series, followed by a race gadget. For simplicity, we will put C forks in each branch of each variable, where the formula has C clauses, though usually we need far fewer. Then each variable gadget takes $k = 3C + 1$ turns to traverse. We call the top path of a race gadget the ***fast exit*** and the bottom path the ***slow exit***, since (in normal play) the first (second) player to arrive leaves through the fast (slow) exit. It will become clear which player each win gadget in a race gadget is for.

The turn order will be U, then E_1, then E_2. Both U and E_1 start at the beginning of the variable gadgets for $\vec{x}_1$. E_2 starts next to a delay line of length d_1. The fast exit of the race gadget for $\vec{x}_1$ and the end of this delay line both connect to the beginning of the $\vec{x}_2$ variable gadgets. The slow exit connects to a delay line of length d_2. The end of this delay line and the fast exit of the $\vec{x}_2$ race gadget connect to the beginning of the $\vec{y}_1$ variable gadgets, and the slow exit connects to a delay line of length d_3. The end of this delay line is connected to the *slow* exit of the $\vec{y}_1$ race gadget and the beginning of the $\vec{y}_2$ variable gadgets. The fast exit of the $\vec{y}_1$ race gadget is connected to yet another delay line of length d_4. The slow exit of the $\vec{y}_2$ race gadget is connected to a long delay line of length d_5 followed by a win gadget

for U, and the fast exit is connected to a longer delay line of length $d_5 + 3$.

This all serves to accomplish the following. First, U chooses the assignment for $\vec{x}_1$ accompanied by E_1, so E_1 learns the assignment. Then U and E_1 are separated, and U assigns $\vec{x}_2$ accompanied by E_2. Next, E_1 chooses $\vec{y}_1$ accompanied by U, and finally E_2 chooses $\vec{y}_2$ accompanied by U. The delays d_1 through d_4 are chosen so that the Existential players arrive at exactly the right time; we have $d_1 = |\vec{x}_1|k + 1$, $d_2 = |\vec{x}_2|k - 1$, $d_3 = |\vec{y}_1|k$, and $d_4 = |\vec{y}_2|$. If a player deviates during variable assignment, they will arrive at their next race gadget too late, and lose.

The end of the final delay line for E_1, of length d_4, is connected to the first clause gadget, and the clause gadgets are connected in series corresponding to the clauses of the formula. The delay lines in each branch of each clause gadget have length Vk, where V is the number of variables; this ensures that if a player enters one of the delay lines during variable selection, an opponent will reach a race gadget and win before they accomplish anything. The end of the last clause gadget is connected to a win gadget for E_1. When E_1 reaches each clause gadget, they try the literals one at a time. When they cross the delay line to the fork, if the fork is traversable, they move on to the next clause. Otherwise they return through the other delay line and try the next literal. Each clause takes up to $6Vk + 1$ turns to cross.

If the formula is satisfied, E_1 eventually gets through all the clauses and wins. Otherwise, U wins after walking through their delay line of length d_5, which we can set to $C(6Vk+1)+1$.

We have seen that no player or team can benefit by deviating from normal play, and normal play is equivalent to the game corresponding to the DQBF. Thus Existential has a forced win if and only if the DQBF is true. □

Lemma 100. *Team motion planning with the single-use one-way gadget is NEXPTIME-complete.*

Proof. The reduction in Lemma 99 still works when we replace each single-use bidirectional gadget with a one-way bidirectional gadget. We have to be a bit more careful than in Lemma 96: of the two paths in a clause gadget from the beginning to a fork, we need one

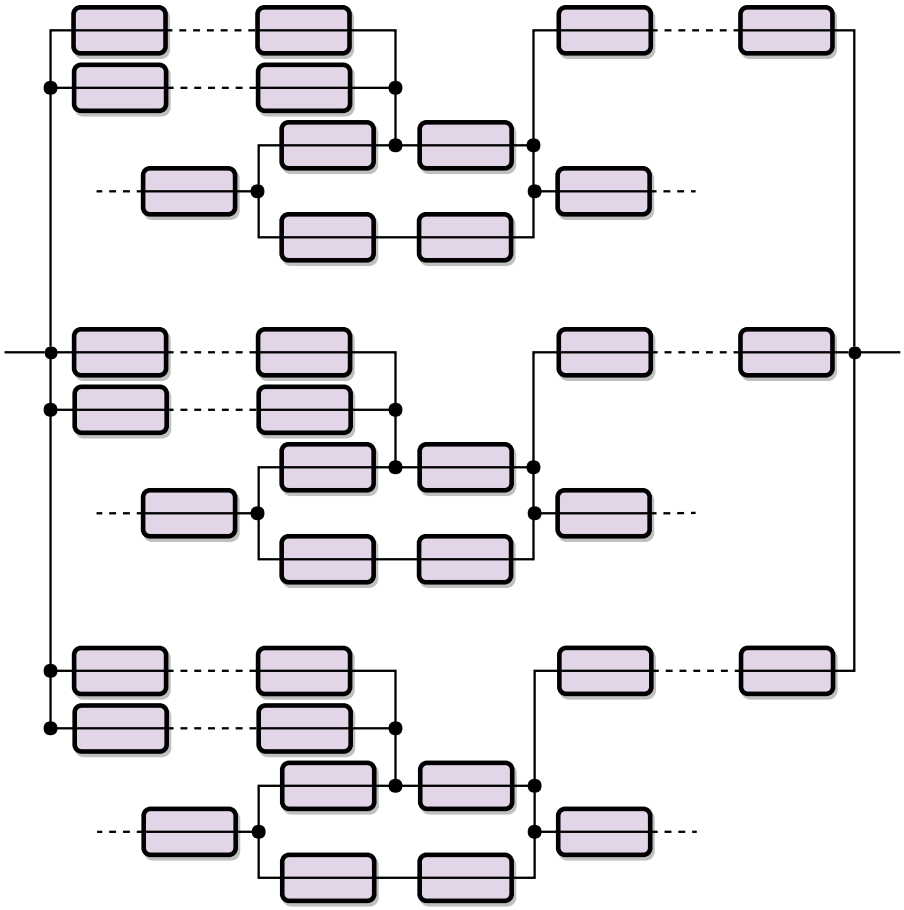


Figure 3-11: A clause gadget for team games. There are now two paths from the entrance of the clause to each fork, so the Existential player traversing the clause can return if they discover the fork is not traversable.

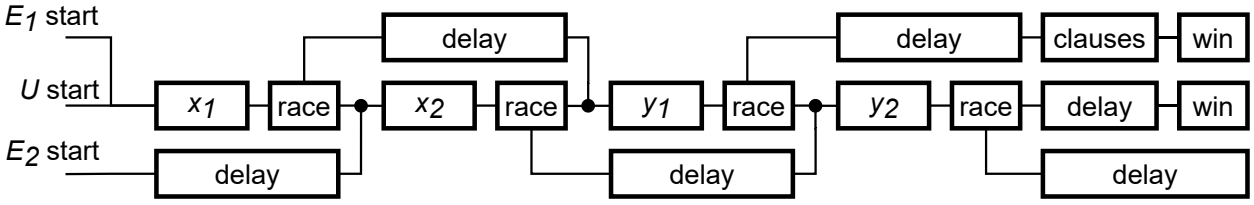


Figure 3-12: The high-level structure of the DQBF reduction.

path to point to the right and the other to point to the left, allowing E_1 to return from that fork. All other gadgets point to the right. □

Corollary 101. *Team motion planning with any nontrivial DAG gadget is NEXPTIME-complete.*

Proof. Every DAG gadget has a single-use transition, which may be either bidirectional or one-way. Both cases are shown to be NEXPTIME-hard in Lemmas 99 and 100. Containment in NEXPTIME is Lemma 98. □

Chapter 4

Zero Player

In this chapter, we study the complexity of zero-player motion planning with deterministic gadgets from several classes. In Section 4.1, we consider input/output gadgets with a single input. In Section 4.2, we consider bounded gadgets and show P-completeness for some classes of k-tunnel LDAG gadgets and input/output gadgets. Finally, in Section 4.3, we consider unbounded 2-state input/output gadgets with multiple inputs, which are naturally PSPACE-complete.

Work in this chapter on input/output gadgets is taken primarily from [9], done in collaboration with Joshua Ani, Erik Demaine, and Dylan Hendrickson. Work on 0-player LDAG gadgets was done in collaboration with the 6.892 Open Problem Session.

In addition to the primary gadget under consideration in this section, we further allow the ***merge gadget*** which is a 1-state, 3-location input-output gadget that has two transitions, one from each of its two inputs to its single output. We often denote this gadget by simply having two edges in the connection graph come together. Note that this gadget is not output-disjoint or reversible. However, having some non-output-disjoint gadget is needed to allow output disjoint gadgets to allow locations to be involved in more than a single cycle. The merge gadget is also implicit in the switching graphs model of [34] which allows in-degree greater than one.

A very simple, but useful gadget which can be constructed with the merge gadget is the

dead-end gadget. This can be constructed by connecting an exit to a merge gadget which then loops back to its other input. This will trap the agent indefinitely if the agent ever reaches that exit.

For deterministic gadgets on tunnels, considered in Section 4.2, to have any hope of hardness we need some way to have an out-degree greater than one. Thus we allow the ***rotate*** gadget which is a 1-state, 3-location gadget with traversals from each location to the next in a clockwise (or counterclockwise) order. This gadget is chosen for its simplicity and its use in various other deterministic models which care about reversibility, such as asynchronous ballistic reversible circuits [36] and time-reversible generalized Landon's Ants [57]. For our k-tunnel gadgets we use the "bounce" model based on [36] for which agents will reverse direction if they encounter an exit with no location in the connection graph or a location with no available transition.[1] For our results on input-output gadgets, we avoid ever having this situation and thus the hardness constructions work regardless of how behavior in these cases is defined.

We conclude this introduction with the simple lemma that will be used several times which shows containment in PSPACE.

Lemma 102. *Zero-player motion planning with any set of gadgets is in* PSPACE.

Proof. In polynomial space, we can keep track of the current configuration of a system of gadgets and current location of the robot. Thus we can simply simulate the zero-player motion planning problem until either the robot reaches the goal location, the robot reaches a dead-end, or it makes more transitions than there are configurations, and thus is stuck in a cycle. □

4.1 Zero-Player Single-Input Gadgets

In this section, we consider zero-player motion planning with deterministic single-input input/output gadgets. If the gadgets are described (for concreteness, using transition graphs)

[1] It seems likely this was chosen to make the computation model conservative in addition to being reversible.

as part of the instance, this is equivalent to the explicit zero-player reachability switching games of [34]. In our language, [34] shows that zero-player motion planning with instance-specified deterministic single-input input/output gadgets is NL-hard. As pointed out in [34], the proofs in [41], which only considered ARRIVAL, also apply to explicit zero-player reachability switching games. In our language, they show that zero-player motion planning with instance-specified deterministic single-input input/output gadgets is in UP $\cap$ coUP (which is contained in NP $\cap$ coNP).

We strengthen the NL-hardness result of [34] by showing that zero-player motion planning with just the toggle switch is NL-hard. This is a straightforward modification of the proof of NL-hardness in [34]; we present the full argument for completeness and to translate it to our terminology. There is still a large gap between the lower bound of NL-hard and the upper bound of UP $\cap$ coUP.

Theorem 103. *Zero-player motion planning with the toggle switch is NL-hard.*

Proof. We reduce from reachability in directed graphs, which is NL-complete. We first replace every vertex v with out-degree $k > 2$ with a sequence of k vertices each with out-degree at most 2: if v has edges to $u_1, \ldots, u_k$, we replace v with $v_1, \ldots, v_k$ with edges $v_i \to v_{i+1}$ and $v_i \to u_i$, and edges to v now go to v_1. Next, remove any vertices with out-degree 1 by setting their incoming edges to instead go to the target of their unique outgoing edge. Every vertex now has out-degree exactly 2. This can be done in logarithmic space and does not affect reachability.

Now we use a construction based on that in [34]. Let V be the set of vertices in the modified graph G, where we are interested in a path from s to t. Our system of gadgets has $|V|$ toggle switches, named (v, i) for $v \in V$ and $1 \leq i \leq |V|$. For a vertex $v \neq t$ with edges to u_1 and u_2 and $i < |V|$, the outputs of (v, i) are connected to the inputs to $(u_1, i+1)$ and $(u_2, i+1)$. For $v \neq t$, both outputs of $(v, |V|)$ are connected to the input to $(s, 1)$. Finally, for each i both outputs of (t, i) are connected to the goal location, which then leads back to $(s, 1)$. The start location is the input to $(s, 1)$.

When it moves through this system, the robot follows paths G starting from s and counts

the number of steps taken, resetting after $|V|$ steps. If it reaches the goal location, it must have entered (t, i) for some i, and thus there is a path (of length $i - 1$) from s to t.

The robot must enter $(s, 1)$ infinitely many times, so it must use each output of $(s, 1)$ infinitely many times. By induction, it uses every toggle switch reachable from $(s, 1)$ infinitely many times. If there is a path from s to t with length $i < |V|$, then $(t, i + 1)$ is reachable from $(s, 1)$, so the robot reaches the goal location. □

4.2 Zero-Player Bounded Gadgets

In this section we give a series of proofs showing the P-completeness of bounded deterministic 2-state input/output gadgets, as well as deterministic LDAG gadgets with distant-opening or distant-closing tunnels and the rotate gadget. The reductions will all be from circuit simulation and will primarily differ in the universal circuit we simulate.

Theorem 104. *Zero-player motion planning with any bounded deterministic gadget is in P.*

Proof. Let k be the maximum number of state changes the gadget can make, and suppose we have a system with n copies of the gadget. Then gadget states can change at most kn times. Between consecutive state-changes, the robot can visit each gadget at most once without being in a loop, so consecutive state-changes are separated by at most n traversals. Hence after kn^2 traversals, the robot must be in a cycle which involves no state-changes. So we can solve the problem in polynomial time by simulating the robot for kn^2 steps and seeing whether it reaches the goal location by then. □

Theorem 105. *Zero-player motion planning with the switch/set-up line or the set-up switch/set-up line is P-hard.*

Proof. We provide a reduction to each of these problems from the problem of evaluating a circuit with only NOR gates and fanout, which is P-complete [42]. The two reductions are nearly identical: we present the reduction for the switch/set-up line, and the reduction for the set-up switch/set-up line is the same with each gadget replaced. We shall see that the

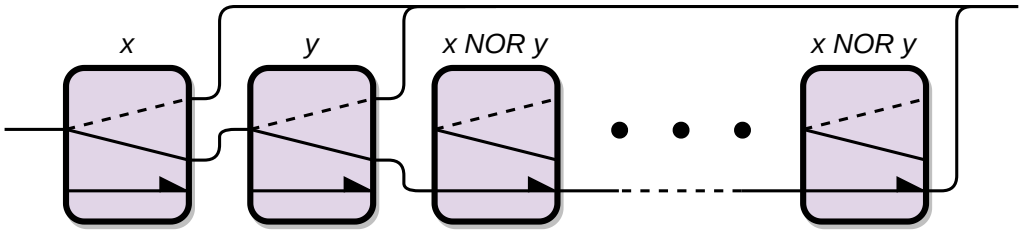


Figure 4-1: A NOR gate for P-hardness of zero-player motion planning with the switch/set-up line. If neither x nor y is set to true (up), the robot sets each x NOR y gadget to true.

robot never goes over a switch multiple times, so these two systems of gadgets behave the same.

Our reduction builds a system of switch/set-up lines which has one gadget for each input to a NOR gate; this gadget indicates whether the input is true or false, and is initially set to false. The robot will evaluate each NOR gate in order by depth, setting the gadgets for outputs of that gate to true if appropriate. This is accomplished with the gadget in Figure 4-1. For each NOR gate, we build one of these gadgets, where x and y are the inputs, and the gadgets labeled x NOR y are the outputs (and inputs of other NOR gates). There are as many output gadgets as the fanout of this NOR gate. The entrance and exit to the NOR-gate gadgets are connected in series, in order by depth.

To complete the construction, we place the start location at the entrance to the first NOR gate. The exit of the last NOR gate enters a switch which holds the output of the final NOR gate, and the goal location is the top output of that switch. Every switch/set-up line starts in the down state except for those that correspond to true inputs to the circuit.

When the robot moves through this system of gadgets, in goes through each NOR gate in order. If either x or y is set to true (i.e., in the up state), the robot leaves x NOR y false, but if x and y are both false, it goes through the set-up lines to set x NOR y true. This correctly computes x NOR y, and by induction it computes the value of the circuit. At the end, the robot reaches the goal location if the value is true and gets stuck in a nearby dead-end if the value is false. □

Corollary 106. *Zero-player motion planning with any bounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs is P-complete.*

Proof. Lemma 61 tells us that every output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs simulates either the switch/set-up line or the set-up switch/set-up line. Combined with Theorem 107 we have our theorem. □

Theorem 107. *Zero-player bouncing motion planning with the rotate gadget, the merge gadget, and a gadget containing either a door opening or a door closing is P-hard.*

Proof. We provide a reduction to each of these problems from the problem of evaluating a circuit. For door closing gadgets we will simulate NOR and fanout, and for door opening gadgets we will simulate NOT, AND, and fanout [42].

Similar to the prior proof, we will use use whether the agent has crossed certain tunnels and thus opened or closed doors as our circuit values. Figure 4-2 shows the construction of a NOR gate from undirected door closing gadgets. If the agent has passed through either the gadgets labeled x or y the corresponding top tunnel will be closed and the agent will bounce off the closed tunnel continuing on without crossing the gadget labeled x NOR y. Figure 4-3 shows the construction of an AND gate from undirected door opening gadgets. If the agent has passed through both the gadgets labeled x and y the corresponding top tunnels will be opened and the agent will go through crossing the gadget labeled x AND y. Figure 4-4 shows the construction of a NOT gate from undirected door opening gadgets. If the gadget labeled x has been traversed, the agent will take the upper loop never crossing NOT x. If it has not been traversed, the agent will bounce back then taking the lower loop before taking the upper loop bouncing again and continuing to the out location.

Figure 4-5 shows how to adjust the construction if the door opening gadgets are directed. The door closing gadgets can be adjusted in the same manner, and the NOT gadget works in both the directed and undirected case.

As in the prior proof, we take an ordering of the gadgets which respects the topological ordering of the gates in the circuit problem. We connect the IN and OUT lines in sequence according to this ordering, and attach the start location to the first input and the goal location after the opening/closing tunnel of the final circuit output. We also use another rotate and a dead-end right before the final gadget, so if the agent bounces off this final

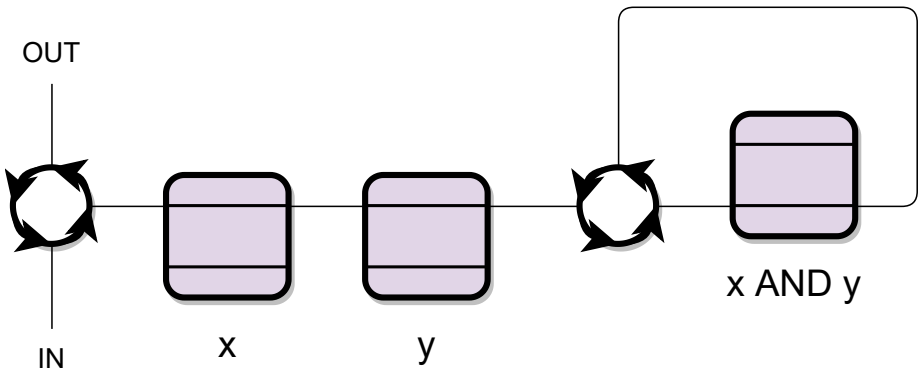


Figure 4-2: A NOR gate for zero-player motion planning with an undirected door closing.

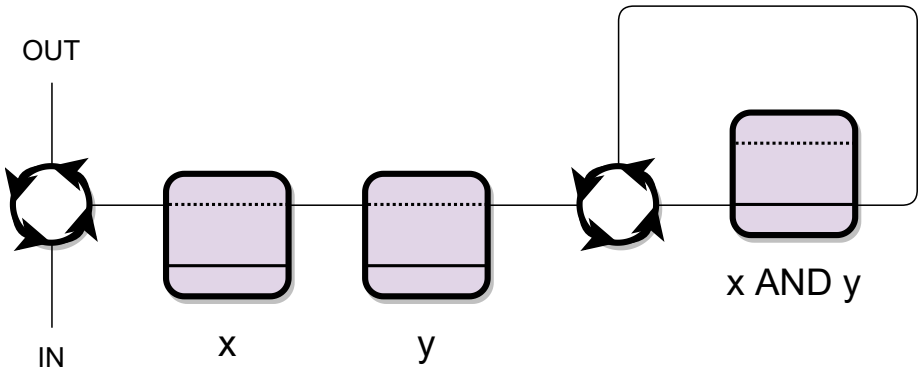


Figure 4-3: An AND gate for zero-player motion planning with an undirected door opening.

gadget (meaning the circuit evaluated to false) the agent will go into the dead end rather than bouncing back to the start and potentially interacting with different gadgets on its continued path. □

Unfortunately, it is not clear whether gadgets with non-interacting tunnels are still in NL in the zero-player model. Thus we do not have a full dichotomy for LDAG gadgets.

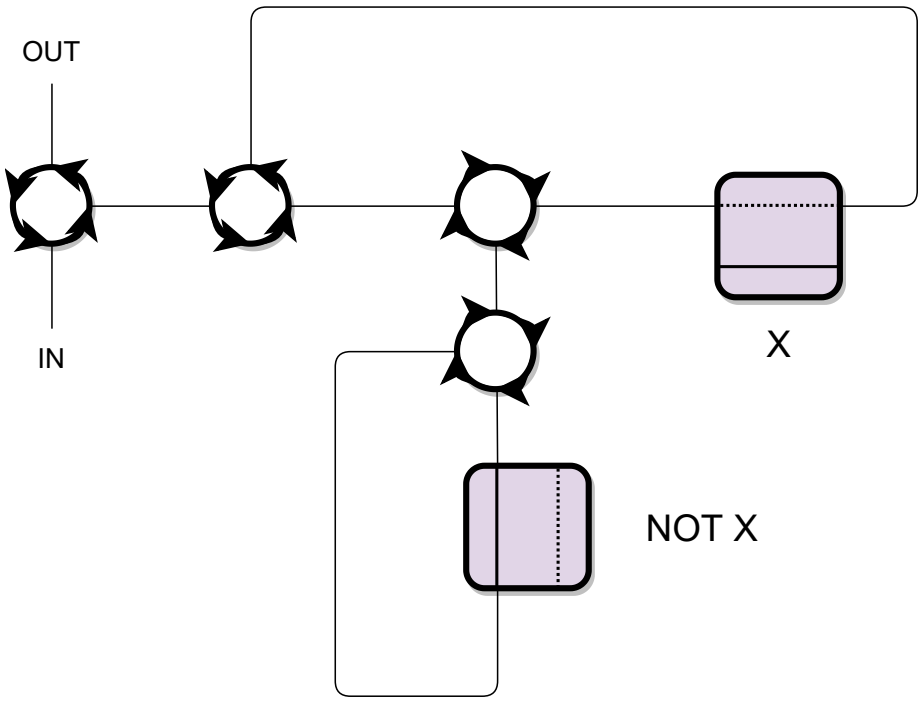


Figure 4-4: A NOT gate for zero-player motion planning with an undirected door opening.

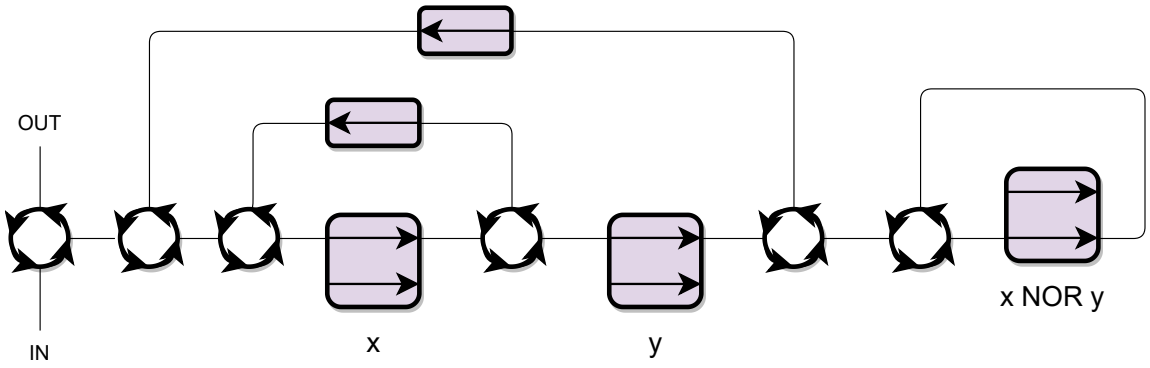


Figure 4-5: An AND gate for P-hardness of zero-player motion planning with an directed door opening.

4.3 Unbounded Input/Output Gadgets

In this section, we consider zero-player motion planning with an unbounded output-disjoint deterministic 2-state input/output gadget which has multiple nontrivial inputs. We show that this problem is PSPACE-complete for every such gadget through a reduction from Quantified Boolean Formula (QBF), to zero-player motion planning with the switch/set-up line/set-down line, and by showing that every such gadget simulates the switch/set-up line/set-down line. We also show that the switch/set-up line/set-down line (and thus every unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs) can simulate every deterministic input/output gadget in zero-player motion planning.

4.3.1 Edge Duplicators

Many of our simulations involve building an ***edge duplicator***, shown in Figure 4-6. An edge duplicator is a construction with two inputs A and B and two outputs A' and B', such that the location the robot leaves corresponds to the location the robot enters, and these two paths intersect. This allows us to place a set line or toggle line along the intersection, making $A \to A'$ and $B \to B'$ both set lines or toggle lines which control the same gadget.

If we have access to an edge duplicator, we can duplicate tunnels in gadgets. Note that this is not enough to duplicate switches, since we would have to account for both exits getting duplicated.

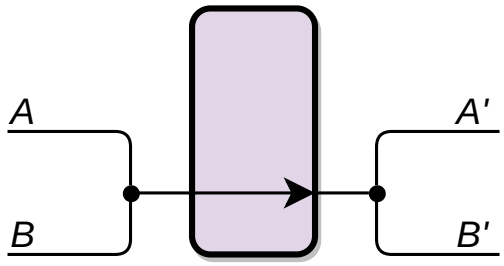


Figure 4-6: The schematic of an edge duplicator. A robot entering at A or B exits at A' or B', respectively, having gone over the central path. This duplicates the edge in the center.

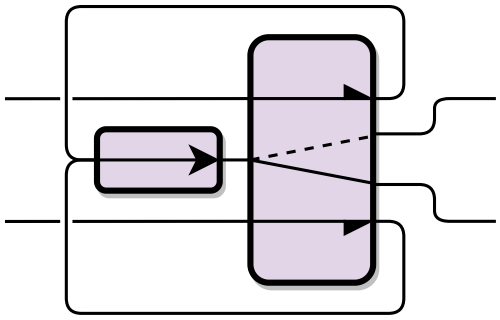


Figure 4-7: An edge duplicator for the switch/set-up line/set-down line. A robot entering on the left sets the state of the switch, goes across the duplicated tunnel, and exits based on the state it set the switch to.

4.3.2 PSPACE-hardness of the switch/set-up line/set-down line

In this section, we show that zero-player motion planning with the switch/set-up line/set-down line is PSPACE-hard through a reduction from QBF. The switch/set-up line/set-down line is a 2-state input/output gadget with three inputs: one sets the state to up, one sets it to down, and one sends the robot to one of two outputs based on the current state.

Theorem 108. *Zero-player motion planning with the switch/set-up line/set-down line is PSPACE-hard.*

Proof. We first build an edge duplicator, shown in Figure 4-7. This allows us to use gadgets with multiple set-up or set-down lines.

Now we present a reduction from QBF. Given a quantified boolean formula where the unquantified formula is 3-CNF, we construct a system of gadgets which evaluates the formula, ultimately sending the robot to one of two locations based on its truth value. The system consists of a sequence of ***quantifier gadgets***, which set the values of variables, followed by the ***CNF evaluation***, which checks whether the formula is satisfied by a particular assignment and reports this to the quantifier gadgets.

Each quantifier gadget has three inputs, called In, True-In, and False-In, and three outputs, called Out, True-Out, and False-Out. The robot will always first arrive at In. This sets the variable controlled by that quantifier to true, and the robot leaves at Out, which

sends it to the next quantifier gadget. Eventually the robot will return to either True-In or False-In, depending on the truth value of the rest of the quantified formula with the variable set to true. Depending on the result, the quantifier gadget either sends the robot to True-Out or False-Out to pass this message to the previous quantifier gadget, or the quantifier gadget sets its variable to false, and again sends the robot to the next quantifier. When it gets a truth value in response the second time, it sends the appropriate truth value to the previous quantifier. The last quantifier communicates with the CNF evaluation instead of with another quantifier.

The universal quantifier gadget is shown in Figure 4-8. The chain of gadgets at the top encode the state of the variable controlled by this quantifier, as has as many gadgets as there are instances of the variable in the formula. The variable is true when they are set to the "left" state and false when they are set to the "right" state.

When the robot enters In, it sets the variable to true and exits Out. If it then returns to True-In, the first time it takes the bottom branch of the switch, sets that gadget to the up state, sets the variable to false, and exits Out again. If it returns to True-In a second time, that means the rest of the formula was true for both settings of the universally quantified variable: it takes the top branch, resets that gadget to down, and exits True-Out. If after either trial the robot enters at False-In, it resets the bottom gadget to the down state and exits False-Out. This is the intended behavior of the universal quantifier: it reports true if the result was true for both settings of the variable, and false otherwise.

The existential quantifier is identical except that True-Out and False-Out are swapped, and True-In and False-In are swapped. It reports false if the result was false for both settings, and true otherwise.

For CNF evaluation, we use the switches controlled by each quantifier to read the value of a variable. For each clause, the robot passes through a switch corresponding to each of the literals in the clause. If all three literals are false, it exits False-Out. Otherwise, it moves on to the next clause, eventually exiting True-Out if all clauses are satisfied. This is shown, for 3 clauses, in Figure 4-9. Ultimately, the robot exits True-Out or False-Out depending on

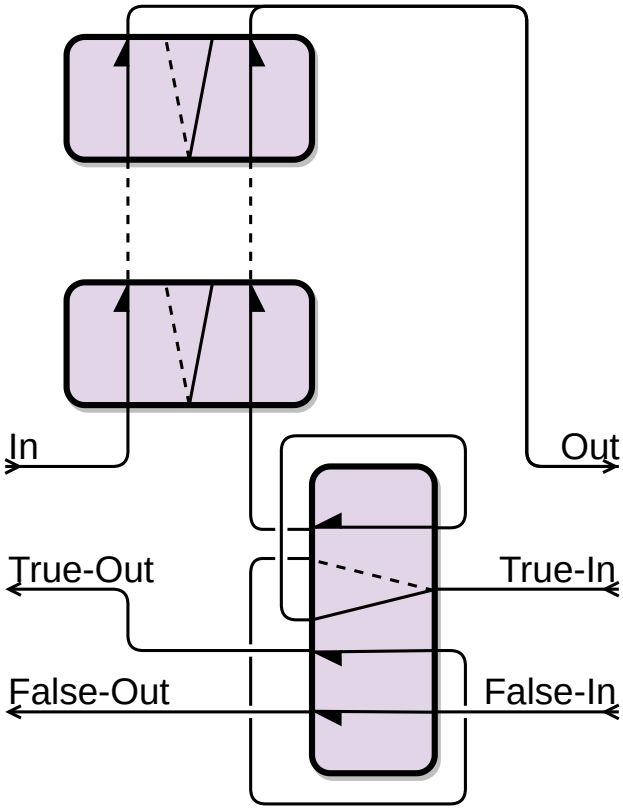


Figure 4-8: The universal quantifier for the switch/set-up line/set-down line. An edge duplicator (Figure 4-7) is used to give the bottom gadget two set-down lines.

whether the formula is satisfied by the current assignment.

It follows by induction that for each quantifier, when the robot arrives at In, it will eventually leave either True-Out or False-Out depending on the truth value of the portion of the formula beginning with that quantifier under the assignment of the earlier quantifiers. Thus, if the robot starts in the first quantifier at In, it reaches True-Out on the first quantifier if and only if the formula is true. □

4.3.3 Other gadgets simulate the switch/set-up line/set-down line

In this section, we show that every unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs simulates the switch/set-up/set-down. We only need to show that the five other gadgets from Lemma 61 simulate the switch/set-

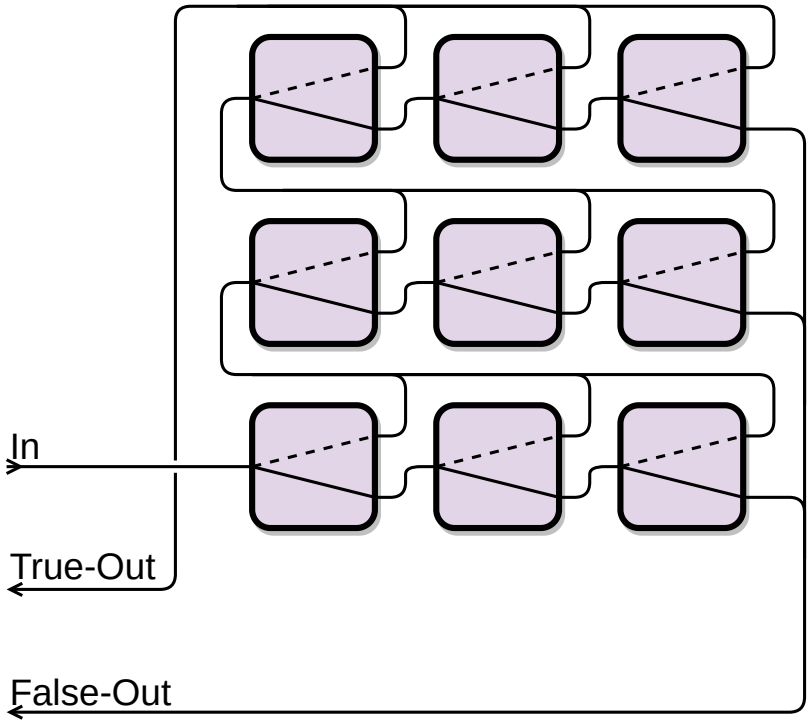


Figure 4-9: Three clauses of CNF evaluation for the switch/set-up line/set-down line; each clause is a row of three switches. The switches are part of gadgets in the quantifiers. We assume the top exit of each switch corresponds to that literal being true; all literals are set to false in this image.

up/set-down. It follows that zero-player motion planning with any such gadget is PSPACE-complete, since we can replace each gadget in a system of switch/set-up/set-down with a simulation of it.

Toggle Switch/Toggle Switch. We begin with the toggle switch/toggle switch, which will be a useful intermediate. It builds an edge duplicator, shown in Figure 4-10. We can merge the two outputs of one of the toggle switches to simulate a toggle switch/toggle line, and then duplicate the toggle line to make a gadget with one toggle switch and any number of toggle lines.

By putting such gadgets in series, we can simulate a gadget with any number of toggle lines and any number of toggle switches. Figure 4-11 shows this for three toggle lines and three toggle switches, which is as large as we need. This simulated gadget can finally simulate the switch/set-up line/set-down line, shown in Figure 4-12.

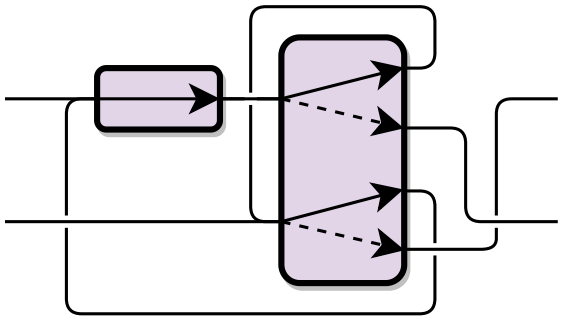


Figure 4-10: An edge duplicator for the toggle switch/toggle switch. The tunnel on the left is duplicated.

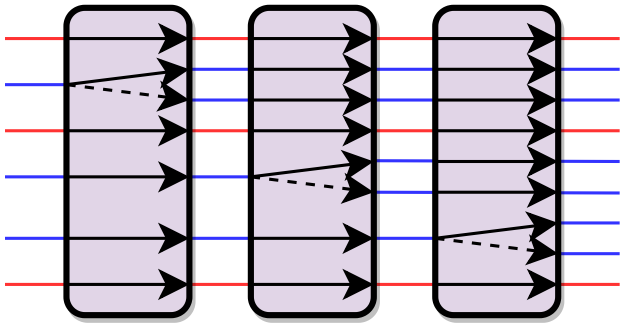


Figure 4-11: A simulation of three toggle lines and three toggle switches from gadgets with one toggle switch and 5, 6, and 7 toggle lines. The red tunnels are toggle lines and the blue tunnels are toggle switches.

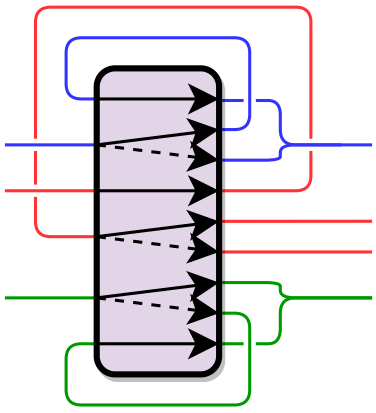


Figure 4-12: A simulation of a switch/set-up line/set-down line from the gadget built in Figure 4-11. Each component of the switch/set-up line/set-down line is made from one toggle line and one toggle switch; the switch, set-up line, and set-down line are red, green, and blue, respectively.

Toggle Switch/Toggle Line. We simulate the toggle switch/toggle switch using toggle switch/ toggle lines, shown in Figure 4-13.

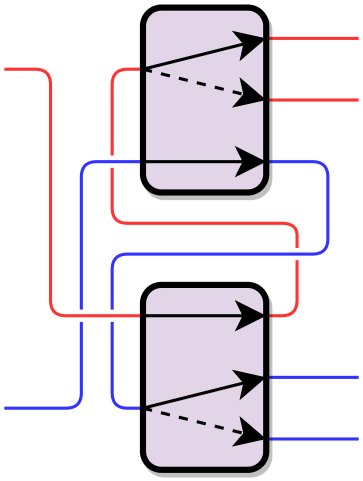


Figure 4-13: A simulation of a toggle switch/toggle switch from the toggle switch/toggle line. Each color corresponds to one of the toggle switches.

Switch/Toggle Line. We first build an edge duplicator, shown in Figure 4-14. We can then duplicate the toggle line and put one copy in series with the switch, constructing a toggle switch/toggle line.

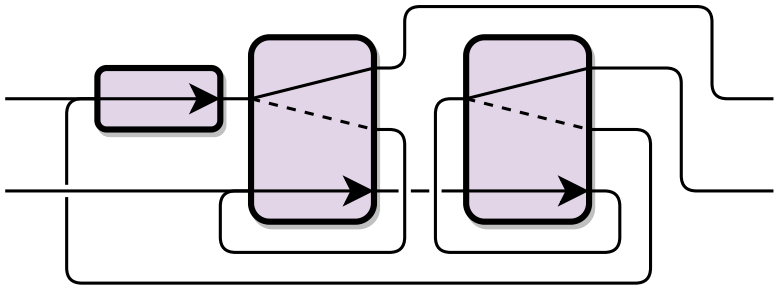


Figure 4-14: An edge duplicator for the switch/toggle line. The leftmost tunnel is duplicated.

Set-Up Switch/Toggle Line. We first build an edge duplicator, shown in Figure 4-15. We then simulate the switch/toggle line, shown in Figure 4-16

Set-Up Switch/Set-Down Line. We simulate a set-down switch/toggle line (equivalent to a set-up switch/toggle line) using the set-up switch/set-down line, as shown in Figure 4-17.

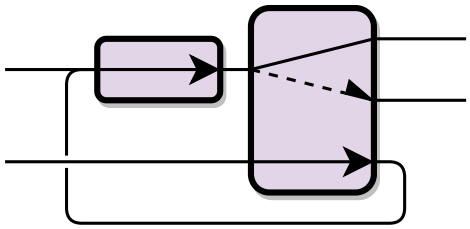


Figure 4-15: An edge duplicator for the set-up switch/toggle line. The leftmost tunnel is duplicated.

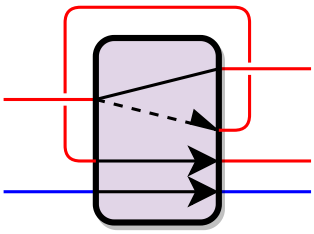


Figure 4-16: A simulation of the switch/toggle line using the set-up switch/toggle line. Red corresponds to the switch and blue corresponds to the toggle line.

Toggle Switch/Set-Up Line. We simulate a set-up line/set-down switch using the toggle switch/ set-up line, as shown in Figure 4-18; this is equivalent to a set-up switch/set-down line.

These simulations, together with Lemma 61, give the following theorem.

Theorem 109. *Every unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs simulates the switch/set-up line/set-down line.*

Corollary 110. *Let G by an unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs. Then zero-player motion planning with G is PSPACE-complete.*

Proof. Containment in PSPACE is given by Lemma 102. All of our simulations preserve PSPACE-hardness: we can reduce from zero-player motion planning with the switch/set-up line/set-down line (shown PSPACE-hard in Theorem 4.3.2) to zero-player motion planning with G by replacing each gadget in a system of switch/set-up line/set-down lines with a simulation built from G. The resulting system of G has the same behavior as the system of switch/set-up line/set-down lines. □

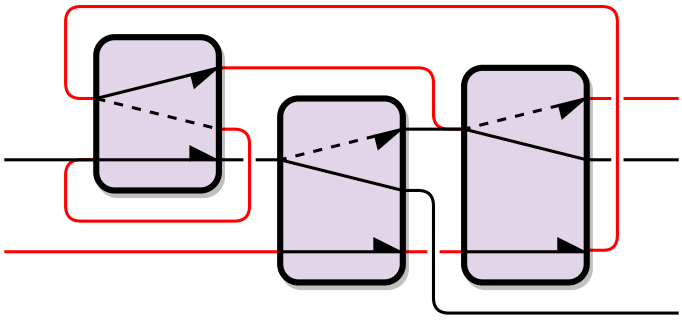


Figure 4-17: A simulation of a set-down switch/toggle line using the set-up switch/set-down line. When the agent is not inside the simulation, rightmost gadget is in the down state and the other two gadgets are in opposite states encode the state of the simulated gadget. Red lines indicate the toggle line: when the agent enters the bottom entrance, it takes one of the internal paths depending on the state and exits the top exit, reversing the state of the left and middle gadgets. When it enters the top entrance, it exits one of the bottom two exits and resets the state to down.

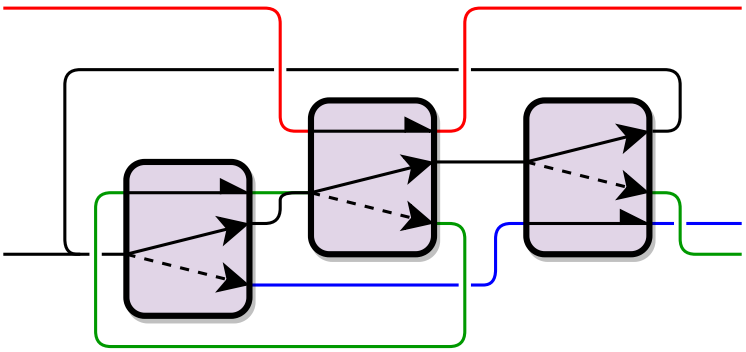


Figure 4-18: A simulation of a set-up line/set-down switch from the set-up line/toggle switch. The state of the simulated gadget is the same as the state of the center gadget. The red path corresponds to the set-up line. When it enters the set-down switch, the robot goes along the blue lines if the state is down, the green lines if the state is up, and the black lines in both cases.

Universality of the switch/set-up line/set-down line

In this section, we show how to simulate an arbitrary deterministic input/output gadget using the switch/set-up line/set-down line, and mention some corollaries of this result. Of particular note is Corollary 114 that in one-player motion planning, the switch/set-up line/set-down line simulates every gadget.

Theorem 111. *The switch/set-up line/set-down line simulates every deterministic input/output gadget in zero-player motion planning.*

Proof. We present simulations of gradually more powerful gadgets. First, the edge duplicator

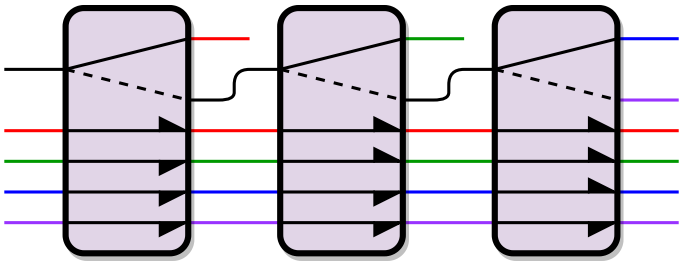


Figure 4-19: A simulation of a 4-switch using the switch/set-up line/set-down line. Colors indicate the outputs corresponding to set lines.

(Figure 4-7) lets us have any number of copies of the set-up and set-down lines.

Next, we simulate a generalization of the switch/set-up line/set-down line which call the k-switch. This gadget has k states, k lines which each set the gadget to a particular state, and an input which does not change the state and sends the robot to one of k locations depending on the state. The switch/set-up line/set-down line is a 2-switch. The simulation for $k = 4$ is shown in Figure 4-19, and generalizes easily to arbitrary k: we need $k-1$ gadgets connected in series, where the ith gadget has i set-up lines and $k - 1 - i$ set-down lines.

We now duplicate the large switch in a k-switch using the construction in Figure 4-20. Thus the switch/set-up line/set-down line can simulate a gadget with any number of states, any number of lines which set it to a particular state, and any number of inputs which send the robot to different outputs depending on the state but do not change the state.

Finally, let G be an arbitrary deterministic input/output gadget. If G has k states and m input locations, we use a k-switch with m copies of the switch to simulate G. The m inputs lead directly to the m switches. For each transition $(\ell, s) \to (\ell', s')$ of G, meaning that when the robot enters at ℓ in state s, it exits and ℓ' and changes the state to s', we connect the output taken in s of the switch corresponding to ℓ to a line which sets the state to s', and connect the output of that line to ℓ'. This encodes the correct behavior for that transition. Since G is deterministic, there is only one such transition for each pair (ℓ, s), so only connect each output of a switch to one input location, as required for zero-player motion planning. □

Corollary 112. *Every unbounded output-disjoint deterministic 2-state input/output gadget*

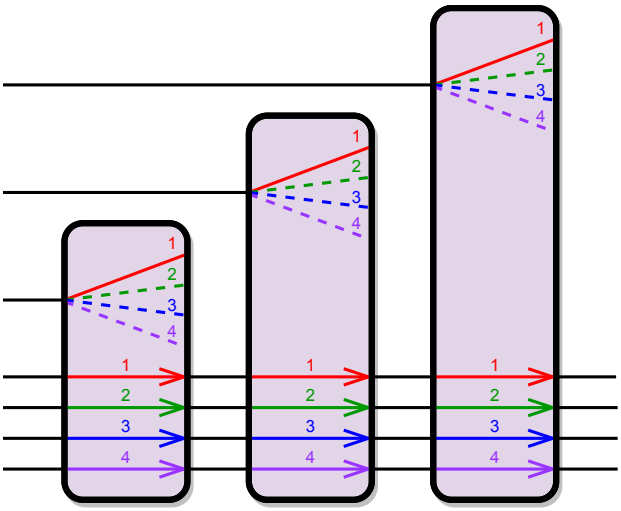


Figure 4-20: Simulating a 4-switch which has three copies of the switch.

with multiple nontrivial inputs simulates every deterministic input/output gadget in zero-player motion planning.

We can make very simple adaptations to these constructions to also show universality for 1-player motion planning.

Corollary 113. *The switch/set-up line/set-down line simulates every input/output gadget in one-player motion planning (that is, we allow multiple input locations in the same connected component of the connection graph, or equivalently allow branching hallways as described in Section 2.5).*

Proof. We use the same construction as in the proof of Theorem 111. If G is nondeterministic—say it has multiple transitions when entering ℓ in state s—we will connect the output taken in s of the switch corresponding to ℓ to multiple input locations. □

Corollary 114. *In one-player motion planning, the switch/set-up line/set-down line simulates every gadget.*

Proof. Let G be an arbitrary gadget. We construct a gadget G' with the same states as G, locations ℓ_{in} and ℓ_{out} for each location ℓ of G, and a transition $(\ell_{in}, s) \to (\ell'_{out}, s')$ for each transition $(\ell, s) \to (\ell', s')$ of G. Clearly G' is input/output: ℓ_{in} and ℓ_{out} are input and

output locations, respectively. Thus by Corollary 113, the switch/set-up line/set-down line simulates G' in one-player motion planning. But G' simulates G simply by connecting both ℓ_{in} and ℓ_{out} to ℓ. □

Corollary 115. *In one-player motion planning, every unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs simulates every gadget.*

Chapter 5

Applications

In this chapter we show several applications of our framework. In Section 5.1 we show new results about variants of block pulling puzzles with gravity, ones with merging, and a simplified proof that push-pull block puzzles are PSPACE-complete. In Section 5.2 we give PSPACE-hardness proofs for eight 3D Mario games. In Section 5.3 we give PSPACE-completeness proofs for more recent Zelda games. In Section 5.3.5 we give a PSPACE-completeness proof for the puzzle game Trainyard which uses fewer features than prior work. In Section 5.4 we show examples where this framework is able to simplify past proofs, including prior results on SNES games and 2-player Mario Kart.

This framework has also been applied by other researchers to reconfiguring robotic swarms with uniform global control. In this model, there are robots which exist in a 2D grid world with obstacles and can simultaneously be given the same movement command which they all execute. Global uniform control could show up in cases like manipulating ferromagnetic particles using a strong uniform magnetic field. In [11] PSPACE-completeness is shown when robots move maximally in a given direction, reducing from 1-player motion planning with a 2-toggle. In [15] PSPACE-completeness is shown when robots move a single step in a given direction, reducing from 1-player motion planning with a crossing toggle-lock. See Section 2.2.6 for details on those gadgets.

Work in this section comes from [6,7,9,12,29] and was written in collaboration with Joshua

Ani, Sualeh Asif, Jeffrey Bosboom, Michael Coulombe, Erik Demaine, Yevhenii Diomidov, Isaac Grosof, Dylan Hendrickson, Lorenzo Najt, Mikhail Rudoy, Sarah Scheffler, and Adam Suhl.

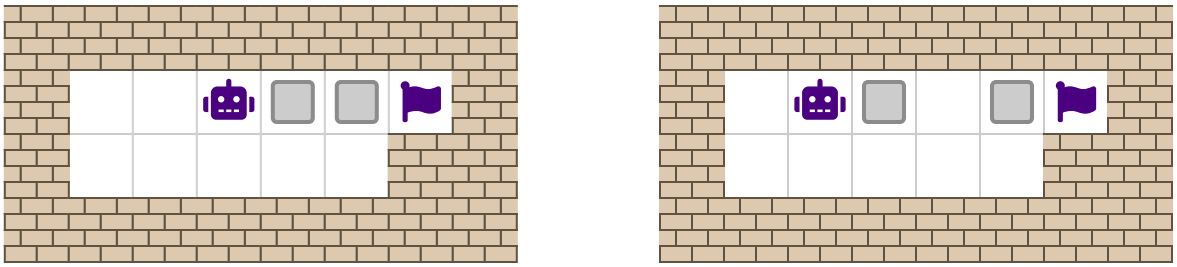
5.1 Block Pushing Puzzles

One interesting and well-studied case of motion planning problems, arising in warehouse maintenance, is when a single robot with $O(1)$ degrees of freedom navigates an environment with obstacles, some of which can be moved by the robot (but which cannot move on their own). Research in this direction was initiated in 1988 [63].

A series of problems in this space arise from computer puzzle games, where the robot is the agent controlled by the player, and the movable obstacles are ***blocks***. The earliest and most famous such puzzle game is ***Sokoban***, first released in 1982 [62]. Much later, this game was proved PSPACE-complete [18,45]. In Sokoban, the agent can ***push*** movable 1×1 blocks on a square grid, and the goal is to bring those blocks to target locations. Later research in ***pushing-block puzzles*** considered the simpler goal of simply getting the robot to a target location, proving various versions NP-hard, NP-complete, or PSPACE-complete [20,28,30].

In Sections 5.1.2 and 5.1.3, we study the Pull series of motion-planning problems [50,53], where the agent can ***pull*** (instead of push) movable 1×1 blocks on a square grid. In Section 5.1.1 we show an alternate proof of the PSPACE-completeness with ***push-pull*** where the agent can both push and pull the movable blocks. Figure 5-1 shows a simple example. This type of block-pulling mechanic (sometimes together with a block-pushing mechanic) appears in many real-world video games, such as Legend of Zelda, Tomb Raider, Portal, and Baba Is You. In Section 5.1.4 we show PSPACE-completeness for the puzzle game Sokobond which is a block pushing puzzle game where blocks are able to bond into larger polyominoes.

The work in this section comes from [29] and [6] written in collaboration with Joshua Ani, Sualeh Asif, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, Scheffler, Sarah and Adam Suhl.



(a) Initial state **(b)** A strength-1 move

Figure 5-1: A pulling-block problem. The robot is the agent, the flag is the goal square, the light gray blocks can be moved, and the bricks are fixed in place. *Robot and flag icons from Font Awesome under CC BY 4.0 License.*

We study several different variants of PULL, which can be combined in arbitrary combination:

1. **Optional/forced pulls:** In PULL!, every agent motion that can also pull blocks must pull as many as possible (as in many video games where the player input is just a direction). In PULL?, the agent can choose whether and how many blocks to pull. Only the latter has been studied in the literature, where it is traditionally called PULL; we use the explicit "?" to indicate optionality and distinguish from PULL!.
2. **Strength:** In PULL-k, the agent can pull an unbroken horizontal or vertical line of up to k pullable blocks at once. In PULL-$*$, the agent can pull any number of blocks at once. Similarly with Push-k the agent is able to push up to k blocks in a row.
3. **Fixed blocks/walls:** In PULL-F, the board may have fixed 1×1 blocks that cannot be traversed or pulled. In the PULL-W, the board may have fixed thin (1×0) walls; this is more general because a square of thin walls is equivalent to a fixed block. Thin walls were introduced in [23].
4. **Gravity:** In PULL-G, all movable blocks fall downward after each agent move. Gravity does not affect the agent's movement.

Table 5.1 summarizes our results for block pulling with gravity: for all variants that include fixed blocks or walls, we prove PSPACE-completeness for any strength, with optional or forced pulls, and with or without gravity, with the exception of PULL?-1FG for which we only show NP-hardness.

Problem	Forced	Strength	Features	Our result
PULL?-1WG	no	$k = 1$	thin walls	PSPACE-complete [§5.1.2]
PULL?-kFG	no	$k \geq 2$	fixed blocks	PSPACE-complete [§5.1.2]
PULL?-$*$FG	no	∞	fixed blocks	PSPACE-complete [§5.1.2]
PULL!-kFG	yes	$k \geq 1$	fixed blocks	PSPACE-complete [§5.1.3]
PULL!-$*$FG	yes	∞	fixed blocks	PSPACE-complete [§5.1.3]

Table 5.1: Summary of results on pulling blocks with gravity.

The only previously known hardness result for this family of problems is NP-hardness for both PULL?-kF and PULL?-$*$F [53]. In some cases, our results are stronger than the best known results for the corresponding PUSH (pushing-block) problem; see [50]. More complex variants PULLPULL (where pulled blocks slide maximally), PUSHPULL (where blocks can be pushed and pulled), and STORAGE PULL (where the goal is to place multiple blocks into desired locations) are also known to be PSPACE-complete [23, 50].

In Section 5.1.2, we prove PSPACE-completeness of most variants with gravity, including all variants with forced pulling and variants with optional pulling and either thin walls or fixed blocks with $k \geq 2$. These reductions are from 1-player planar motion planning with the ***nondeterministic locking 2-toggle***, from Section 2.2.5 and the ***3-port self-closing door***, Section 2.9.2. In Section 5.1.3, we prove NP-hardness for the one remaining case of PULL?-1FG, using the NAND gadget, Section 2.3.8.

Lemma 116. *Every block-pushing puzzles are in* PSPACE.

Proof. The entire configuration while playing on instance of a block-pushing problem can be stored in polynomial space (e.g., as a matrix recording whether each cell is empty, a fixed block, a movable block, the agent's location, or the finish tile). There is a simple nondeterministic algorithm which guesses each move and keeps track of the configuration using only polynomial space, accepting if the agent reaches the goal square. Thus the problem is in NPSPACE, so by Savitch's Theorem [54] it is also in PSPACE. □

5.1.1 Push-Pull Block Puzzles

In this section, we use the results in thesis to provide a simple proof that a Sokoban variant called PushPull-1F is PSPACE-hard, by reducing from motion planning in planar systems of locking 2-toggles (Section 2.2.4). This problem, and many related problems, were considered in [23] and were shown to be PSPACE-complete in [50] by a reduction from nondeterministic constraint logic; our reduction is much more straightforward using the infrastructure of the gadget framework.

Definition 117. *In **PushPull-1F**, there is a square grid containing movable blocks, fixed blocks, an agent, and a goal location. The agent can freely move through empty squares, but ca not move through blocks. The agent can push or pull one movable block at a time. The agent wins by reaching the goal location. The corresponding decision problem is whether a given instance of PushPull-1F is winnable.*

In the notation "PushPull-1F," "PushPull" indicates that the agent can both push and pull, "1" indicates the number of blocks which can be moved at a time, and "F" indicates the existence of fixed blocks [24].

Theorem 118 ([50])**.** *PushPull-kF is* PSPACE*-hard for $k \geq 1$.*

Proof. We reduce from 1-player planar motion planning with locking 2-toggles, shown PSPACE-complete in Theorem 12. The (planar) connection graph is implemented using tunnels built with fixed blocks, and the agent and target location are placed appropriately. It suffices to build a gadget which behaves as a locking 2-toggle.

Such a gadget is shown in Figure 5-2. The two tunnels, currently both traversable, go from top to left and right to bottom. They interact in the center, where traversing either tunnel requires pushing a block into the middle square, which blocks the other tunnel. This is surrounded by four 1-toggles, which prevent additional traversals which are not possible in a locking 2-toggle. Each 1-toggle is a room with 3 blocks, which can only be entered on one side. Upon entry, the agent can move the blocks to reveal the other exit, but doing so requires blocking the entrance taken, which flips the 1-toggle.

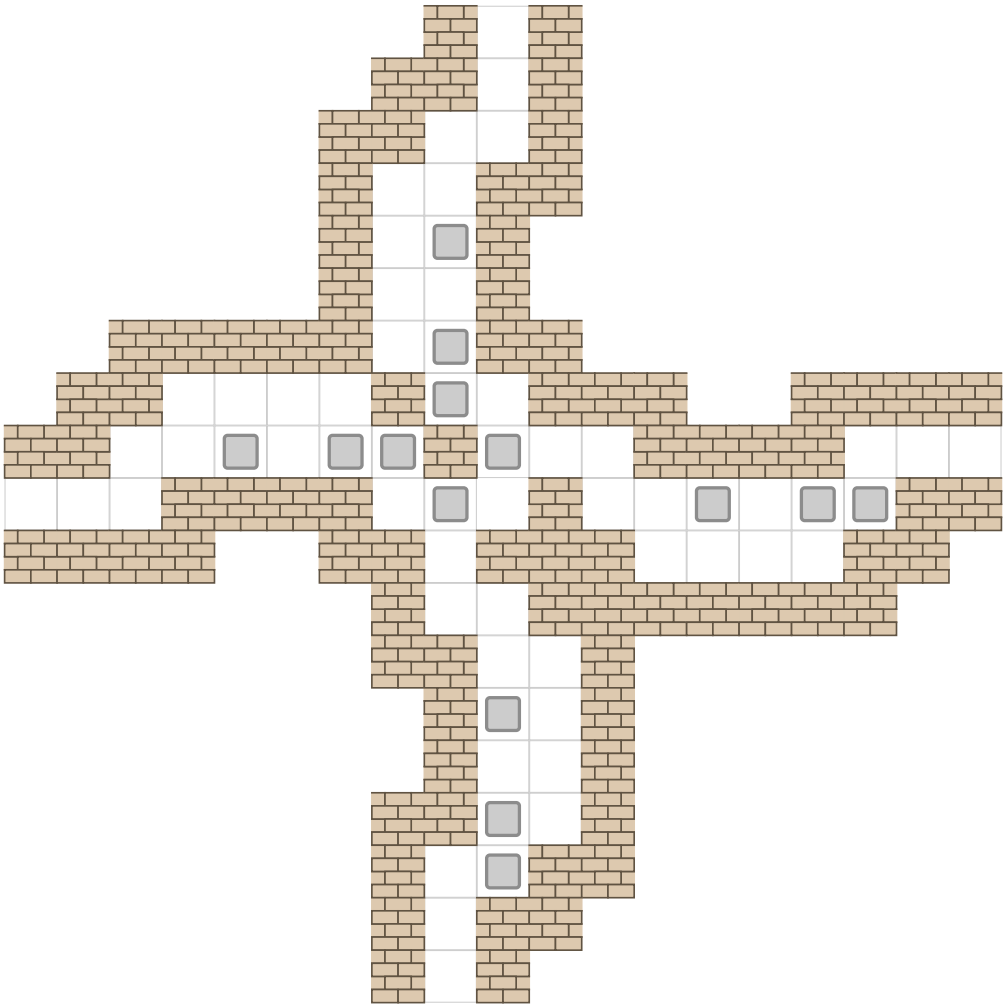


Figure 5-2: A locking 2-toggle in PushPull-1F.

□

5.1.2 Pulling Block with Gravity

In this section, we show PSPACE-completeness results for most of the pulling-block variants with gravity. In Section 2, we introduce and prove results about ***1-player motion planning*** from the motion-planning-through-gadgets framework introduced in [26], which will be the basis for the later proofs. In Section 5.1.2, we show PSPACE-completeness for PULL?-kFG with $k \geq 2$, for PULL?-$*$FG, for PULL?-kWG with $k \geq 1$, and for PULL?-$*$WG. In Section 5.1.3, we show PSPACE-completeness for PULL!-kFG with $k \geq 1$, and for

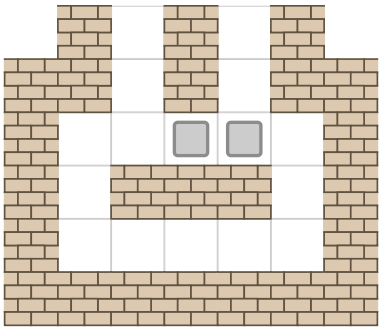


Figure 5-3: 1-toggle in PULL?-2FG.

PULL!-∗FG. The one case missing from this collection is PULL?-1FG, which we prove NP-hard later in Section 5.1.3.

Pull?-kFG

In this section, we show that several versions of pulling-block problems with optional pulling and gravity are PSPACE-complete by a reduction from 1-player motion planning with nondeterministic locking 2-toggles, shown PSPACE-hard in Section 2.2.5.

We begin with a construction of a 1-toggle, and then use those and an intermediate construction to build a nondeterministic 2 toggle.

1-toggle. A ***1-toggle*** is a gadget with a single tunnel, traversable in one direction. When the agent traverses it, the direction that it can be traversed is flipped, meaning that the agent must backtrack and return the way it came in order to be able to traverse it the first way again.

Our 1-toggle construction in PULL?-kFG for $k \geq 2$ is shown in Figure 5-3. In the state shown, it can only be traversed from left to right by pulling both blocks to the left. This traversal flips the direction that the gadget can be traversed—it can now only be traversed from right to left.

Nondeterministic Locking 2-toggle. Our construction of a nondeterministic locking 2-toggle, shown in Figure 5-4, uses two 1-toggles plus a connecting section at the top.

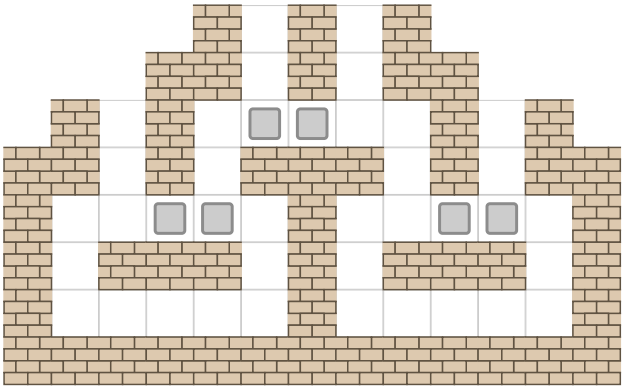


Figure 5-4: Locking 2-toggle in PULL?-2FG.

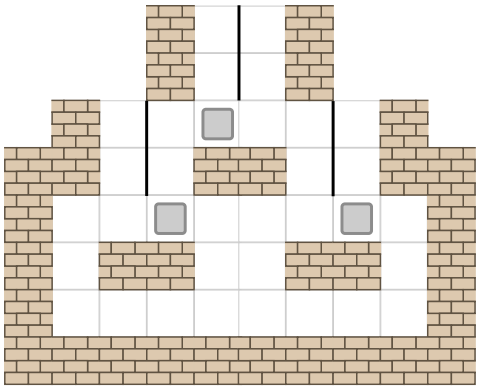


Figure 5-5: Locking 2-toggle in PULL?-1WG.

The configuration shown in Figure 5-4 is a leaf state. The right tunnel is traversable from to right to bottom right. If the agent traverses that tunnel, it can choose whether to pull the top pair of blocks to the right (because pulling is optional), corresponding to the nondeterministic choice in the nondeterministic locking 2-toggle. Both 1-toggles will be in the state where they can be traversed from bottom (outside) to top (inside). One of these paths will be blocked by the top pair of blocks and the other will be traversable, depending on whether the agent chose to pull those blocks. Traversing the traversable path then puts the gadget in a leaf state, either the one shown or its reflection.

It is possible for the agent to pull only one block instead of two, but this can only prevent future traversals, so never benefits the agent.

Theorem 119. PULL?-*kFG is* PSPACE*-complete for* $k \geq 2$ *and* $k = *$.

Proof. Lemma 116 gives containment in PSPACE. For hardness, we reduce from 1-player planar motion planning with the nondeterministic locking 2-toggle, shown PSPACE-hard in Theorem 13. We embed any planar network of gadgets in a grid, and replace each nondeterministic locking 2-toggle with the construction described above in the appropriate state. The resulting pulling-block problem is solvable if and only if the motion planning problem is.

This reduction works for PULL?-kFG for any $k \geq 2$ including $k = *$, because the player only ever has the opportunity to pull 2 blocks at a time. This proof requires optional pulling

because the player must choose whether to pull blocks while traversing a nondeterministic locking 2-toggle. □

Corollary 120. PULL?-k*WG is* PSPACE*-complete for* $k \geq 1$ *and* $k = *$.

Proof. With thin walls, the tunnels can be separated by a thin wall instead of a fixed block, which means that only one block is required in each of the toggles. This is shown in Figure 5-5. The rest of the proof follows in the same manner, demonstrating PSPACE-completeness of PULL?-kWG for $k \geq 1$. □

5.1.3 Pull!-kFG

In this section, we show PSPACE-completeness for pulling-block problems with forced pulling and gravity, using a reduction from 1-player planar motion planning with the 3-port self-closing door, shown PSPACE-hard in Theorem 80.

Theorem 121. PULL!-k*FG is* PSPACE*-complete for* $k \geq 1$ *and* $k = *$.

Proof. Lemma 116 gives containment in PSPACE. We show PSPACE-hardness by a reduction from 1-player planar motion planning with the 3-port self-closing door. It suffices to construct a 3-port self-closing door in PULL!-kFG.

First, we construct a diode, shown in Figure 5-6. The agent cannot enter from the right. If the agent enters from the left, it must pull the left block to the left to advance. If it pulls the left block left and then exits, they still cannot enter from the right, so doing so is useless. The agent then advances and is forced to pull the left block back to its original position. The agent then must pull the right block left to advance, and must actually advance because the way back is blocked. As the agent exits the gadget, it is forced to pull the right block back to its original position. Therefore, the agent can always cross the gadget from left to right and never from right to left, simulating a diode.

Using this diode, we then construct a 3-port self-closing door, shown in Figure 5-7; the diode icons indicate the diode shown in Figure 5-6. The bottom is exit-only. In the closed state, the agent should not enter from the top because it would become trapped between a

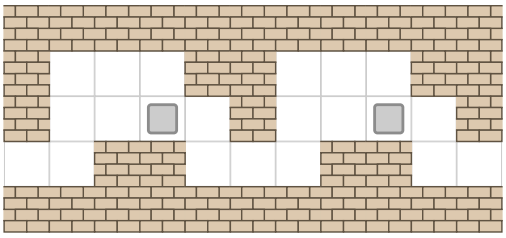
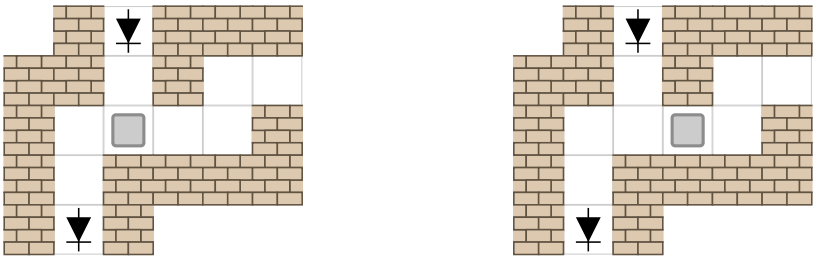


Figure 5-6: A diode in PULL!-kFG.



(a) Closed **(b)** Open

Figure 5-7: A 3-port self-closing door in PULL!-kFG.

block and the wrong end of a diode. The agent can enter from the right, pull the block 1 tile right, and leave, opening the gadget. In the open state, the agent can enter from the top and exit out the bottom, and is forced to pull the block back to its original position, closing the gadget. So this construction simulates a 3-port self-closing door.

Because the player never has the opportunity to pull multiple blocks, this reduction works for all $k \geq 1$ including $k = *$. □

Pull?-1FG is NP-hard

In this section, we show NP-hardness for PULL?-1FG by reducing from 1-player planar motion planning with the crossing NAND gadget from [7]. A ***crossing NAND gadget*** is a three-state gadget with two crossing tunnels, where traversing either tunnel permanently closes the other tunnel. 1-player planar motion planning with the crossing NAND gadget is NP-hard in [7, Lemma 4.9] based on the constructions in [20, 37] which originally reduce from PLANAR 3-COLORING.

Theorem 122. PULL?-*1FG is* NP-*hard.*

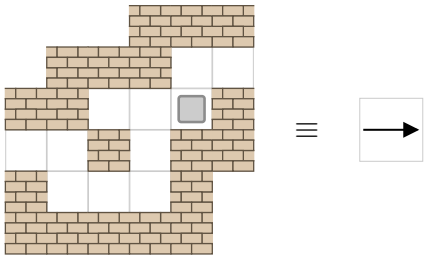


Figure 5-8: Single-use one-way gadget that initially allows traversal from left-to-right and then prevents traversal in both directions.

Proof. We reduce from 1-player planar motion planning with the crossing NAND gadget [7, Lemma 4.9]. First we first construct a "single-use" one-way gadget, shown in Figure 5-8. This gadget can initially can be crossed in one way, but then becomes impassable in both directions.

Figure 5-9 shows our construction of the crossing NAND gadget. Single-use one-way gadgets enforce that the agent must enter through one of the top paths. The agent must pull two blocks to enter the gadget; these blocks end up stacked in the vertical tunnel on top of the block below. The agent cannot exit via the bottom tunnel underneath its entry tunnel: the agent can pull one block into the slot on the bottom, and then can pull one block one square, but that still leaves the third block of the stack blocking off the exit path. The agent cannot exit via the other top path, because it is blocked by the single-use one-way gadget. The only path remaining is for the agent to cross diagonally by pulling the single block in the lower layer into the slot, revealing a path to the exit opposite where the agent entered. After leaving, both the entry tunnel and exit tunnel are impassable because the single-use one-way gadgets have become impassable. If the agent later enters via the other entry tunnel, the agent will be trapped, because it will not be able to leave via the tunnel that was "collapsed" in the initial entry. □

5.1.4 Sokobond

Sokobond [43] is a 2D block pushing game where the blocks are atoms/molecules. Movement is discrete along a square grid. The player starts as a single atom. Each atom except He has

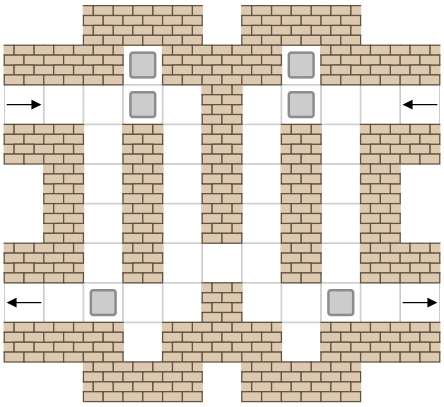


Figure 5-9: Crossing NAND gadget allowing traversal either from the top-left to the bottom-right, or from the top-right to the bottom-left. After being traversed once, the entire gadget becomes impassable in any direction.

some number of free electrons (H has 1, O has 2, N has 3, C has 4). When two atoms that both have free electrons are adjacent, they both lose a free electron and bond into a molecule. Molecules are rigid, so pushing an atom in a molecule results in the entire molecule moving. Atoms/molecules can also push each other.

Sokobond with He atoms is trivially NP-hard as it includes PUSH-$*$ [20]. We show PSPACE-hardness even without He atoms:

Theorem 123. *Completing a level in Sokobond with H and O atoms is* PSPACE*-hard.*

Proof. We reduce from 1-player planar motion planning with a door that is not the Case 8: OTtocC door and use Theorem 86.

Let the player start as an H atom trying to reach another H atom. We can simulate a door that is not the Case 8: OTtocC door as shown in Figure 5-10. To open the door, the player pulls down on the big molecule. The player can go through the traverse tunnel if and only if the molecule is down. When going through the closing tunnel, the player is forced to push up on the molecule, closing the traverse tunnel. The molecule used to simulate a door has no free electrons, so the level can be completed if and only if the player can reach the other H atom. □

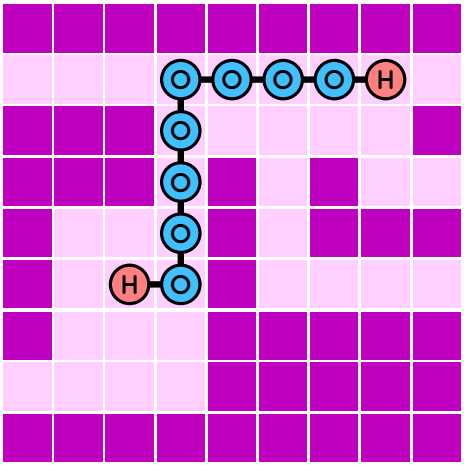


Figure 5-10: Simulation of a door in Sokobond. The opening port is at the bottom left. The traverse tunnel is undirected and runs between the top left and the top right. The closing tunnel is undirected and runs between the middle right and the bottom right.

5.2 3D Mario Games

Super Mario Bros. is one of the most famous games of all time and its computational complexity has been studied in [5] and [32], culminating in a proof of PSPACE-hardness. We continue this line of inquery moving from the classic SNES games of previous papers to more recent 3D Mario games. The reductions give examples of simple proofs using the symmetric self-closing door from Section 2.8. These results come from [7] coauthored with Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, and Dylan Hendrickson. The proofs come primarily from work by Joshua Ani.

5.2.1 Super Mario 64/Super Mario 64 DS

Super Mario 64 is a 3D Mario game for the Nintendo 64 where Mario collects Stars from courses inside paintings to save the princess, who is trapped behind a painting. Super Mario 64 DS is a remake of Super Mario 64 for the Nintendo DS (still in 3D), featuring the same courses as in Super Mario 64 plus new courses, as well as the ability to play as characters other than Mario. In this reduction, we will primarily make use of quicksand, which will defeat Mario if he lands in it, and the ghost enemy Boo.

The Boo is an enemy that (with normal parameters) chases Mario if he is looking away from it and is less than a certain distance away. Once Mario gets too far, the Boo moves back to its original position. Unlike most enemies, jumping on a Boo does not kill it, but instead sends it a short distance forward or backward, which we will use to help Mario cross the quicksand. Some walls stop the Boo but it can go through certain walls that normal Mario cannot go through, we call these Boo-only walls. The Boo is also unable to go through doors. We also make use of one-way walls which Mario and the Boo can go through in one direction but not the other.

For the setup, we use one Boo in Super Mario 64 DS and two Boos in Super Mario 64. Performing a kick while in the air sends Mario a short distance up and can normally only be performed once per jump. But Mario can kick after jumping on a Boo in Super Mario 64 DS even if he already kicked, allowing him to jump on the same Boo. This is not true in Super Mario 64, so jumping on a second Boo is necessary to stall long enough to jump on the first Boo again.

Theorem 124. *Collecting a Star in a Super Mario 64/Super Mario 64 DS course is* PSPACE*-hard assuming no course size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 77), where the target to reach is a Star. The simulation is shown in Figure 5-11.

In the setup below, Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by going through the door on the bottom-left and hopping on the Boo(s) to the top-left. Then Mario lets the Boo(s) chase him a little to turn the Boo(s), and hops on the Boo(s) to push it into the top-right. Finally, Mario goes through the top-left door. Mario cannot just jump to the other side because the distance is too far. He also cannot go into the traverse path because of the Boo-only wall. The Boo(s) will try to go back to its home, but cannot because it is stuck behind a 1-way wall and a regular wall. If Mario does not move the Boo(s) to the top-right, it still cannot get back to its home because of a different 1-way wall, so Mario cannot leave the port 1 to port 2 traversal open.

Mario goes from port 3 to port 4 by going through the top-right door and hopping on

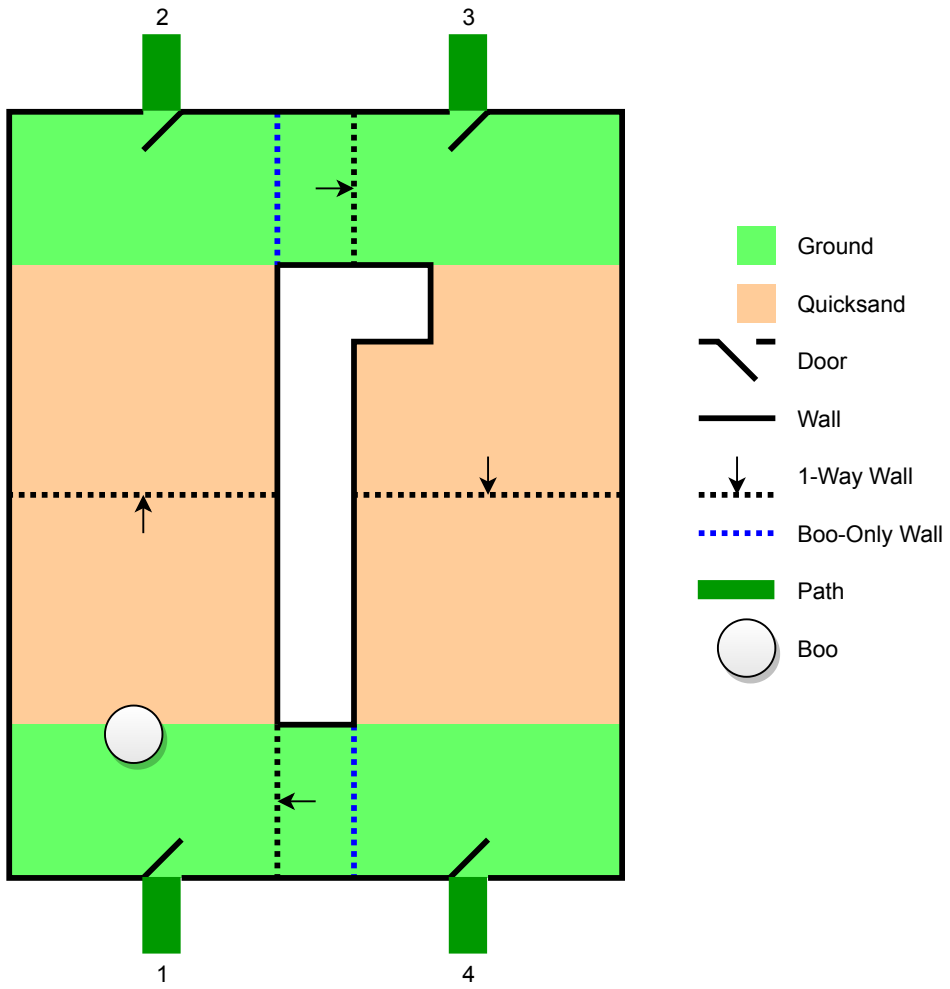


Figure 5-11: Simulation of a symmetric self-closing door in Super Mario 64 DS. In Super Mario 64, there are 2 Boos instead of 1. The ground and quicksand are on the same vertical level. The room is covered by a ceiling. The hallways are too wide to wall jump across.

the Boo(s) to the bottom-right, then going through the bottom-right door. The Boo(s) will go back to its original position at the bottom left on its own.

Mario cannot lure the Boo(s) away from the gadget because it is completely walled in except for the doors, which the Boo(s) cannot go through. □

5.2.2 Super Mario Sunshine

Super Mario Sunshine is a 3D Mario game for the GameCube where Mario is falsely accused of spreading graffiti and is forced to clean it up before he can leave. Like Super Mario 64, this game includes one-way walls. This game features a new device, F.L.U.D.D., attached to Mario's back that allows him to spray water. Lily Pads float on water; the player can ride a Lily Pad and cause it to move by spraying water in the opposite direction. Sludge is an environmental hazard which kills Mario if he touches it. The general goal of a level is to collect Shrine Sprites.

Theorem 125. *Collecting a Shine Sprite in a Super Mario Sunshine level is* PSPACE*-hard assuming no level size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 77), where the target to collect is a Shine Sprite. The simulation of a symmetric self-closing door is shown in Figure 5-12.

The thin water above the sludge prevents the Lily Pad from disintegrating, while preventing Mario from crossing without using the Lily Pad. Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by crossing the 1-way wall and riding the Lily Pad across, then moves the Lily Pad partially across the slit so it can be accessed from the other side. He cannot leak to the section between port 3 and port 4 because the slits are too thin. The sludge is too long to simply jump to the other side, so the Lily Pad is needed. Mario cannot do anything from port 2 because the 1-way wall blocks him from going to port 1. Mario goes from port 3 to port 4 in a similar manner. □

5.2.3 Super Mario Galaxy

Super Mario Galaxy is a 3D Mario game for the Wii where Mario goes to space. He encounters alien creatures along the way and collects Power Stars to restore the power of a spaceship. The game features downward gravity, upward gravity, sideways gravity, spherical

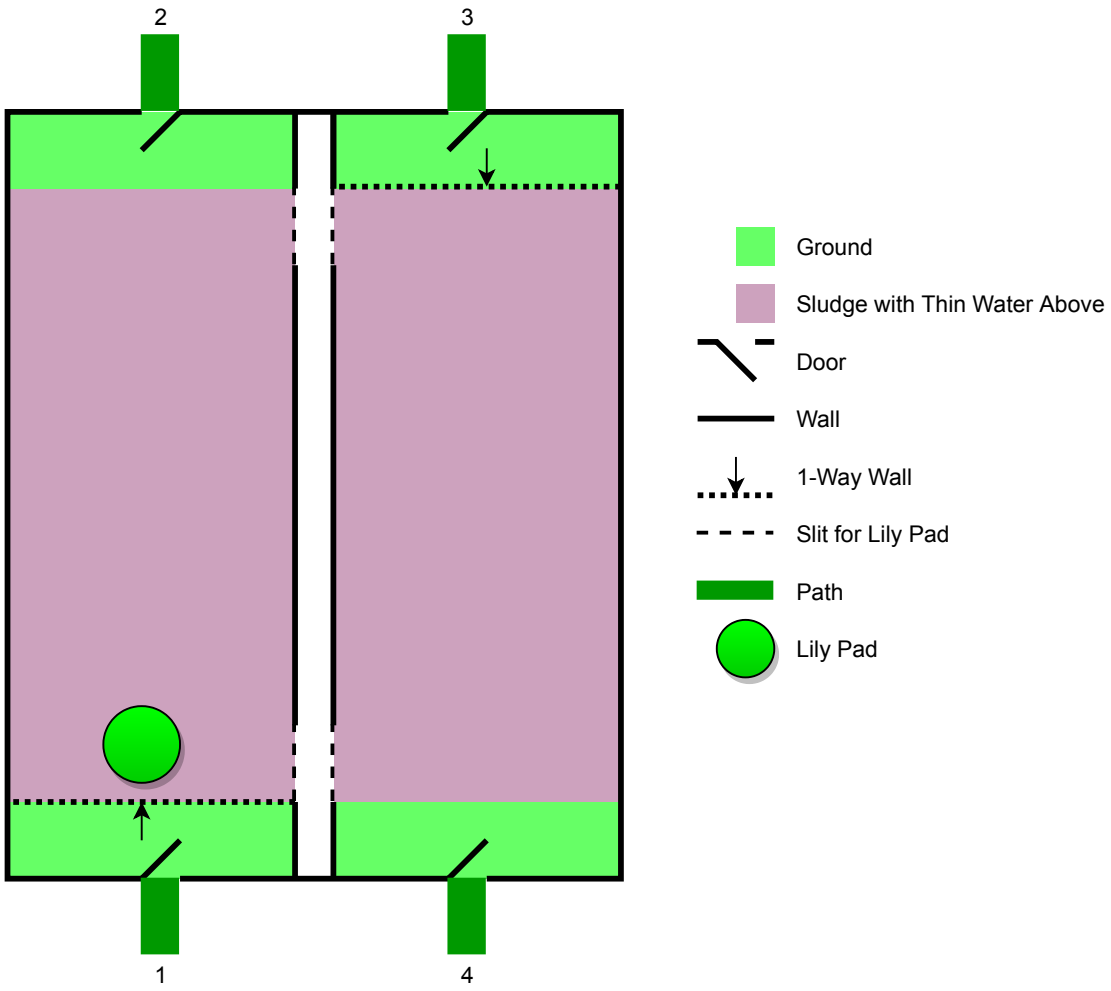


Figure 5-12: Simulation of a symmetric self-closing door in Super Mario Sunshine. The slits allow the Lily Pad to cross without allowing bulky Mario to do so. The hallways are too wide to wall jump across.

gravity, cubical gravity, tubular gravity, cylindrical gravity that allows infinite freefall, W-shaped gravity, gravity that cannot make up its mind, and most importantly, controllable gravity.

Dark matter disintegrates Mario when he touches it, resulting in death. The Gravity Switch changes the direction of gravity when spun and can be spun multiple times.

Theorem 126. *Collecting a Power Star in a Super Mario Galaxy galaxy is* PSPACE*-hard assuming no galaxy size limits.*

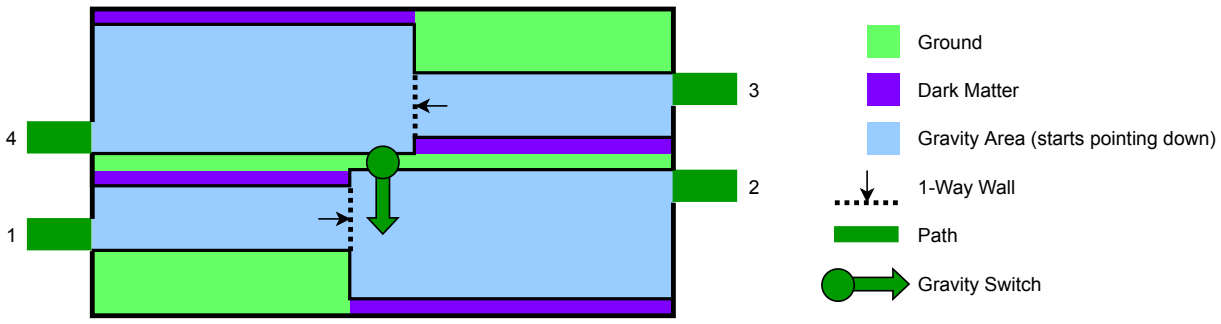


Figure 5-13: Simulation of a symmetric self-closing door in Super Mario Galaxy. This is a side view and is essentially 2-dimensional.

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 77), where the target to collect is a Star. The simulation of a symmetric self-closing door is shown in Figure 5-13.

The Gravity Switch in this construction switches gravity between down and up. Mario goes from port 1 to port 2 by crossing the 1-way wall and hitting the Gravity Switch on his way to the right. This is forced because of a pit of dark matter, and closes the port 1 to port 2 traversal because when gravity points up, attempting the traversal would land Mario on dark matter. At the same time, it opens the port 3 to port 4 traversal. Mario cannot enter port 2 and do anything useful because flipping the Gravity Switch means falling in the pit of dark matter. Mario goes from port 3 to port 4 in a similar manner. □

5.2.4 Super Mario Galaxy 2

Super Mario Galaxy 2 is the sequel of Super Mario Galaxy (also for the Wii) which features new galaxies. Although similar in gameplay, each game has some objects which do not appear in the other. This reduction is very similar to that in Section 5.2.2 with the Lily Pad and sludge.

The Leaf Raft is a raft that floats on water and that can be moved by standing on its edge. Lava is an environmental hazard which damages Mario. Unrealistically, we can have a thin layer of water on top of a layer of lava. Finally, an electric fence is another environmental hazard which damages Mario but will allow the Leaf Raft to pass through it.

Theorem 127. *Collecting a Power Star in a Super Mario Galaxy 2 galaxy is* PSPACE*-hard assuming no galaxy size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 77), where the target to collect is a Star. The simulation of a symmetric self-closing door is shown in Figure 5-14. The Star is under a Daredevil Comet, making Mario have only 1 HP, so he cannot afford to bounce in the lava or shock-boost through an electric fence.

Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by crossing the 1-way wall and riding the Leaf Raft across, then carefully making the Leaf Raft partially cross the electric fence. He cannot move to the section between port 3 and port 4 because of the electric fences. The lava is too long to simply jump to the other side, so the Leaf Raft is needed. Mario cannot do anything from port 2 because the 1-way wall blocks him from going to port 1. Mario goes from port 3 to port 4 in a similar manner. □

5.2.5 Super Mario 3D Land and Super Mario 3D World

Super Mario 3D Land and Super Mario 3D World are 3D Mario games for the 3DS and Wii U, respectively, that are based on the New Super Mario Bros. series instead of earlier 3D Mario games. Instead of collecting Shine Sprites or Stars, the player traverses a level to reach the flagpole at the end. In addition, the player does not have a health bar but loses their powerup or dies when taking damage. Levels have time limits.

The Switchboard is a platform that rides on tracks and contains two arrows. If Mario steps on an arrow, the Switchboard goes in the direction of said arrow. Otherwise, the Switchboard does not move. In Super Mario 3D World, the Switchboard can be controlled by using the Wii U gamepad, but only if the Switchboard is visible. We also make use of a pit deep enough that Mario cannot jump out.

Theorem 128. *Reaching the flagpole at the end of a Super Mario 3D Land/World level is* PSPACE*-hard assuming no level size limits and no time limit.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 77), where the target to reach is the flagpole. The simulation of a symmetric self-closing door is shown in Figures 5-15 and 5-16.

In the setup below, Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by going through the tunnel, then riding the Switchboard to the other side making sure it goes through the wall, then going through the tunnel on the other side. Mario cannot move the Switchboard to the other side and then leave via port 1 because the pit is too wide and the Switchboard cannot be moved without going through the (1-way) tunnel because it is blocked by a wall. If the Switchboard is on the wrong side, it cannot be moved either backward (because the path stops) or forward (because a wall then blocks the way). This ensures that Mario can go from port 1 to port 2 if and only if the Switchboard is already at port 1, and then the Switchboard must stay at port 2/port 3. Mario goes from port 3 to port 4 in a similar manner. □

5.2.6 Super Mario Odyssey

Super Mario Odyssey is a 3D Mario game for the Switch where Mario travels to different kingdoms collecting Power Moons and eventually goes to the Moon. Mario has the ability (via his hat Cappy) to capture certain enemies and objects to use their powers, but such objects tend to reset position after being uncaptured, so we will not be using them here.

We make use of a Jaxi, poison, and timed platforms. A Jaxi is a statue lion that can be ridden safely across poison, which is a hazard that kills Mario.

A timed switch makes some event happen for a specific amount of time. In our reduction, timed switch X makes platform X appear for just long enough for Mario to make a traversal.

Theorem 129. *Collecting a Power Moon in a Super Mario Odyssey kingdom is* PSPACE*-hard assuming no kingdom size limit.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 77), where the target to reach is a Power Moon. The simulation of a symmetric

self-closing door is shown in Figure 5-17.

Mario goes from port 1 to port 2 by pressing timed switch A, riding the Jaxi to the right, and traversing platform A. This opens the port 3 to port 4 traversal while closing the port 1 to port 2 traversal. Mario cannot go to port 3 because of the wide gap, or to port 4 because platform B is gone. The Jaxi is required because the poison it is on is very wide. Mario cannot do anything useful if he tries to enter from port 2 or port 4 because the platforms would be gone. Mario goes from port 3 to port 4 in a similar manner. □

5.2.7 Captain Toad: Treasure Tracker

Captain Toad: Treasure Tracker is a 3D puzzle platformer in the Mario universe, originally appearing as a type of level in Super Mario 3D World, and then released as a stand-alone game on the Wii U and ported to the 3DS and Switch. Notably, Toad can fall but not jump. The game contains rotating platforms controlled by a wheel which Toad must be adjacent to to move. The platforms move in 90° increments. We show PSPACE-hardness by constructing an antiparallel symmetric self-closing door (Theorem 77).

Theorem 130. *Collecting Stars in a Captain Toad: Treasure Tracker is* PSPACE*-hard assuming no level size limit.*

Proof. Figure 5-18 gives a top-down view of the construction. There is a U-shaped rotating platform at a height slightly below the high ground and far above the low ground. The U-shaped platform rotates counterclockwise and can be reached from the nearby high ground; however, the gap between the back of the U and the other side is too far for Toad to jump. Further, the dividing wall sits slightly above the rotating platform, preventing Toad from crossing. Toad is able to go onto the U platform from the high ground, activate the gear twice, and jump off of the U platform onto the low ground across the gap. The U platform is now facing the other way, allowing Toad to enter from the high ground on the other side, but preventing other traversals. □

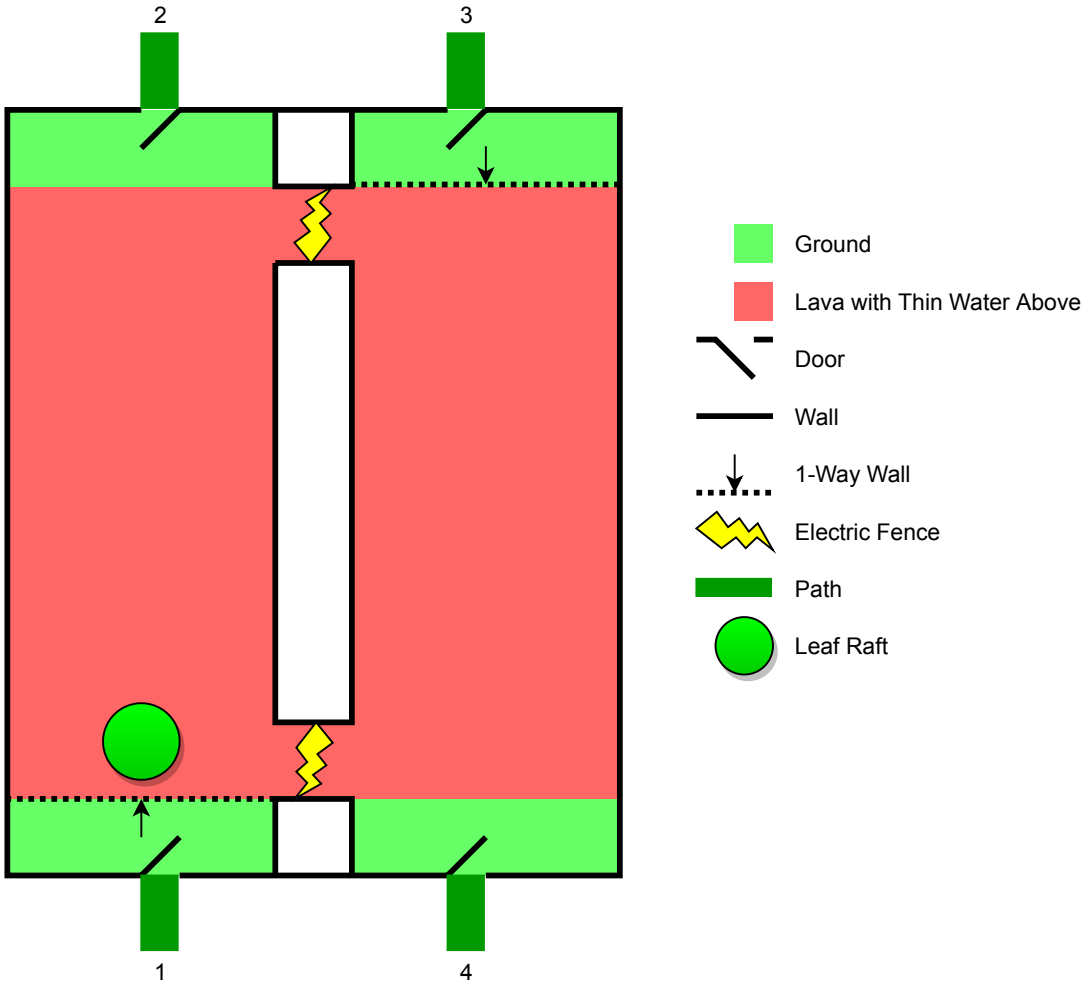


Figure 5-14: Simulation of a symmetric self-closing door in Super Mario Galaxy 2. The thin water above the lava allows the Leaf Raft to float but does not allow Mario to swim in it without getting his butt fire-hot. The walls cannot be wall jumped on (Super Mario Galaxy 2 allows vertical walls that cannot be wall jumped on).

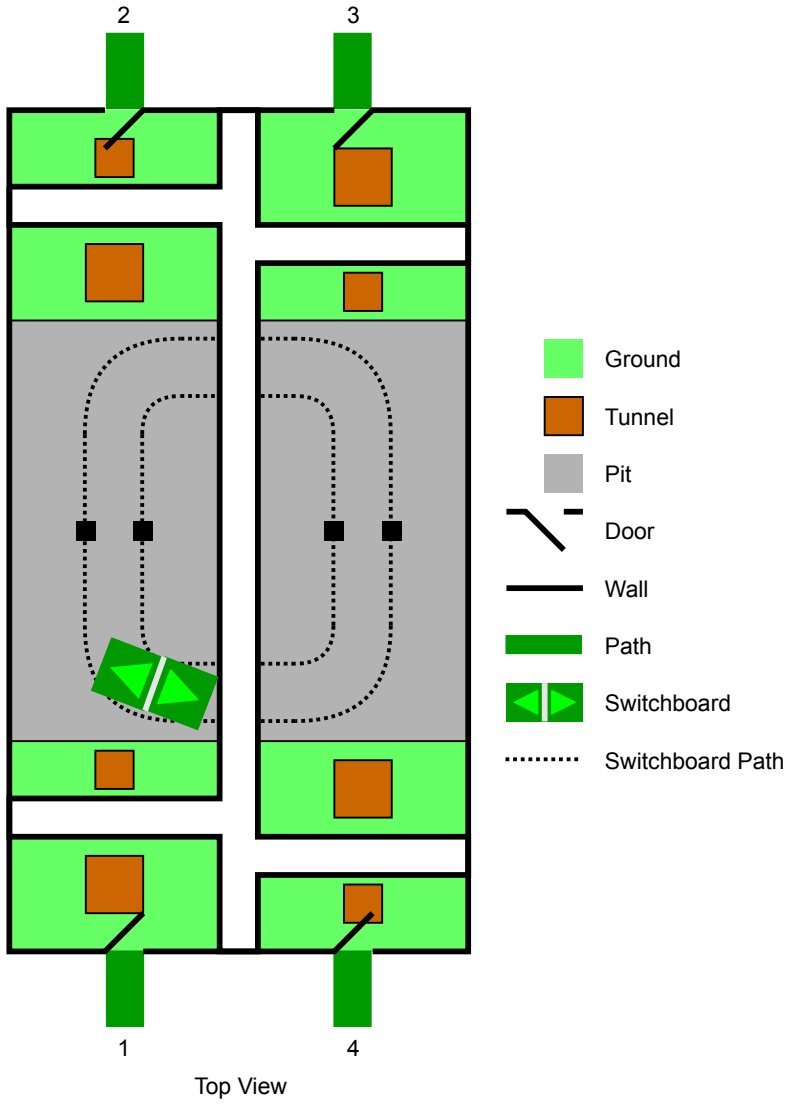


Figure 5-15: Top view of the simulation of a symmetric self-closing door in Super Mario 3D Land/World. The pit is long enough for the player to not be able to jump from the ground to anywhere near the center of the pit. The wider sides of the tunnels are wide enough to not allow wall jumping, making the tunnels 1-way. The hallways are also too wide to wall jump across.

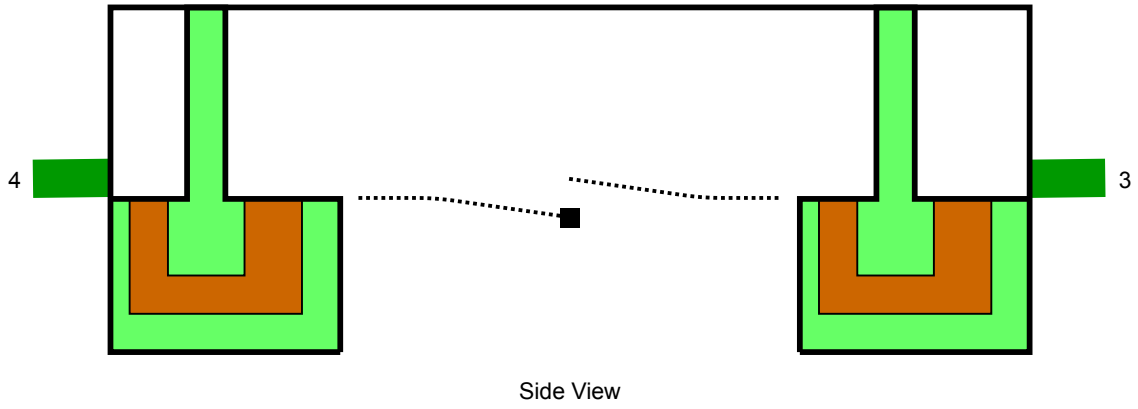


Figure 5-16: Side view of the simulation of a symmetric self-closing door in Super Mario 3D Land/World

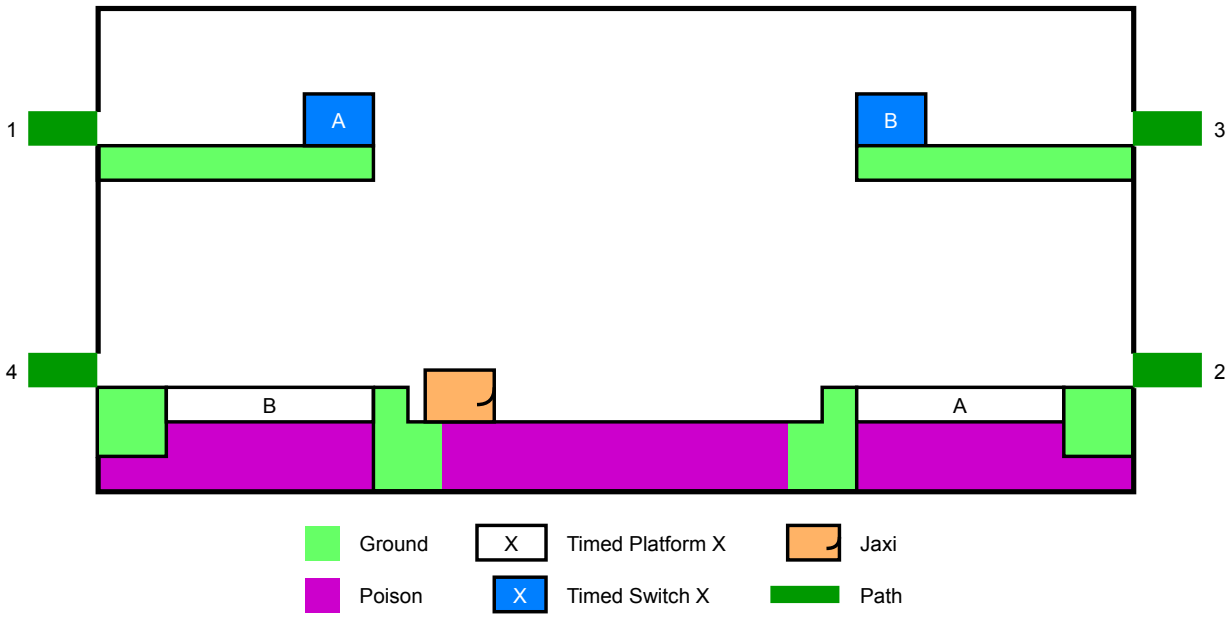


Figure 5-17: Simulation of a symmetric self-closing door in Super Mario Odyssey. This is a side view and is essentially 2-dimensional. All strips of poison are way too wide for Mario to cross with his various aerial skills, and the platforms with timed switches are too high to get to from below.

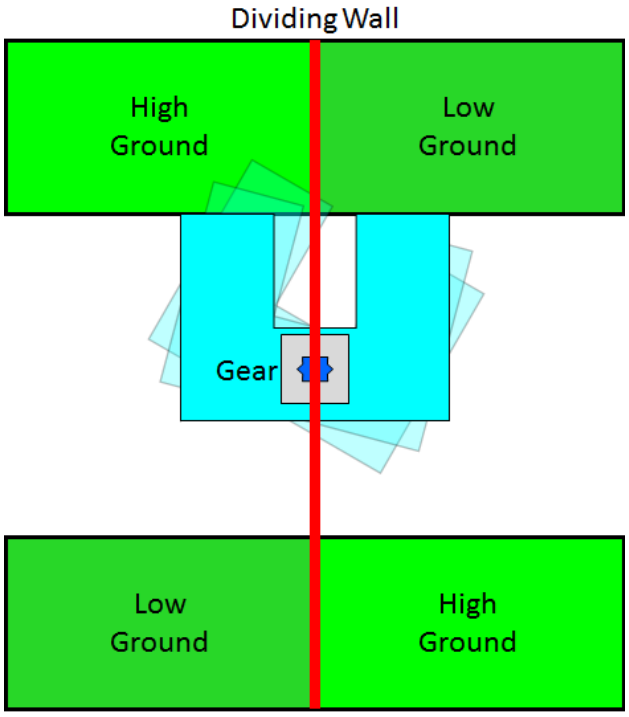


Figure 5-18: Top view of a simulation of a symmetric self-closing door.

5.3 Zelda

Zelda is a hugely popular series of action/adventure video games starting in 1986 with The Legend of Zelda which has sold over 6.5 million copies and currently consisting of 19 games, with the most recent, Zelda: Breadth of the Wild, selling more than 19 million copies [59]. A large number of game mechanics have been introduced over its history and the computational complexity of several of these was studied in [5] which showed that Zelda with push-only blocks is NP-hard; Zelda with hookshot, push-and-pull blocks, chests, pits, and tunnels is NP-hard; Zelda with Small Keys, doors, and ledges is NP-hard; Zelda with ice and sliding push-only blocks is PSPACE-complete; and Zelda with buttons, doors, teleporter tiles, pits, and Crystal Switches that activate raised barriers is PSPACE-complete. This section adds several results to that list, showing they also suffice for PSPACE-hardness. We use the planar Door result of Section 2.9.3, the planar self-closing door result of Section 2.8, and the 2-toggle of [25].

This work comes from [12] and [25] written in collaboration with Jeffrey Bosboom, Michael Coulombe, Erik D. Demaine, Isaac Grossof, Dylan Hendrickson, Lorenzo Najt, and Mikhail Rudoy.

5.3.1 Spinners

In The Legend of Zelda: Oracle of Seasons Link encounters a device with four entrances which, when entered, rotates in one direction and changes color. If entered again, it rotates in the other direction and changes color back. A picture is shown in Figure 5-19. This behavor fits perfectly into the 1-player motion planning framework. Define a k-spinner to be a two state deterministic reversible gadget on k locations. In one state, each location is connected to its neighbor by a directed edge in a clockwise direction. In the other state, all locations are likewise connected in a counterclockwise direction. We show that for any $k \geq 4$, path-planning problems with k-spinners and branching hallways is PSPACE-complete.

First, we can take a k spinner and have all but three consecutive locations lead to dead ends. The remaining three locations form a gadget that we call a deterministic fork. A

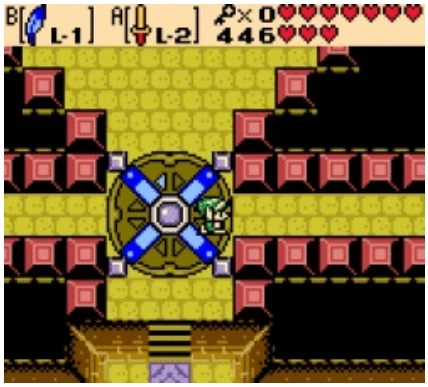


Figure 5-19: Example of a 4-spinner in The Legend of Zelda: Oracle of Seasons.

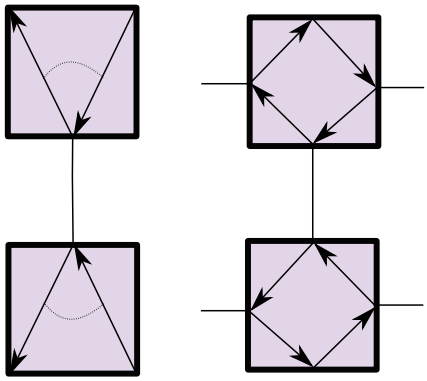


Figure 5-20: 4-spinners simulate deterministic forks which simulate crossing 2-toggles

deterministic fork is a reversible, deterministic gadget on three locations. In one state, it allows the robot to go from the center to the right location and return from the left to the center location. In the other state these directions are reversed. Figure 5-20 shows the construction of a crossing 2-toggle from two 4-spinners or equivalently two deterministic forks.

Theorem 131. *For any $k \geq 4$, the path-planning problem with k-spinners and branching hallways is* PSPACE*-complete.*

Proof. We construct a deterministic fork by ignoring $k - 3$ of the edges in the spinner. Two deterministic forks together simulate a crossing 2-toggle as shown in Figure 5-20. By Corollary 9, the motion planning problem with crossing 2-toggles is PSPACE-complete. □

Corollary 132. *Determining if a player can beat a level in generalized The Legend of Zelda: Oracle of Seasons is* PSPACE*-complete.*

Proof. The Legend of Zelda: Oracle of Seasons contains 4-spinners and requires the player to navigate from one location to a target location in a grid. Since planar graphs can be laid out in a grid with only quadratic blowup [19], we can reduce from motion planning problems with 4-spinners which are PSPACE-complete by Theorem 5.3.1. □

Corollary 133. *The Legend of Zelda: Oracle of Seasons is* PSPACE*-hard.*

5.3.2 Magnetic Gloves is PSPACE-hard

The Magnetic Gloves are an item introduced in The Legend of Zelda: Oracle of Seasons, a 2D game, that projects a north or south magnetic force in any of the four cardinal directions. Among other interactions, they allow Link to remotely attract or repel metal "N" orbs, which are polarized north. Two important properties are the fact that multiple metal objects in range of the force are affected simultaneously, and that metal orbs are affected at any distance, even when off-screen. Since there are no rooms in the game larger than 15×11 tiles or containing more than one metal orb, we make the assumptions that the force would affect multiple metal orbs simultaneously and that orbs cannot overlap other orbs, and consider the cases where it has an infinite range and when it has a finite range of at least 15 tiles.

Theorem 134. *Generalized 2D Zelda with infinite-range magnetic gloves, metal orbs, ledges, and jump platforms is* PSPACE*-hard.*

Proof. We show PSPACE-hardness via reduction from motion planning with door gadgets [5]. Figure 5-21 shows our construction of a door gadget. In the center of the gadget is a metal orb that always blocks the traverse path (when closed) or the close path (when open). To open the door from the closed state, Link must be in the open path and repel the central metal orb with north magnetic force while facing down. To use the close path while in the open state, Link must use north magnetic force to repel the central metal orb while facing up. If Link tries to attract the central metal orb with south magnetic force, then one of the two ledge orbs will fall and permanently block the traverse path.

In an effort to embed the graph into a single room, we must prevent Link from using the magnetic gloves to manipulate a metal orb inside a gadget from far away. This is solved by entirely surrounding the room with a path with metal orbs on ledges leading to the goal, as in Figure 5-22. By selectively removing orbs (that would otherwise be dropped to block this path) in rows or columns which we intend the magnetic gloves to be used with a certain polarity, and placing our gadgets on disjoint sets of rows and columns, any unintended magnetic manipulations will permanently block the outer path and prevent the goal from being reached. □

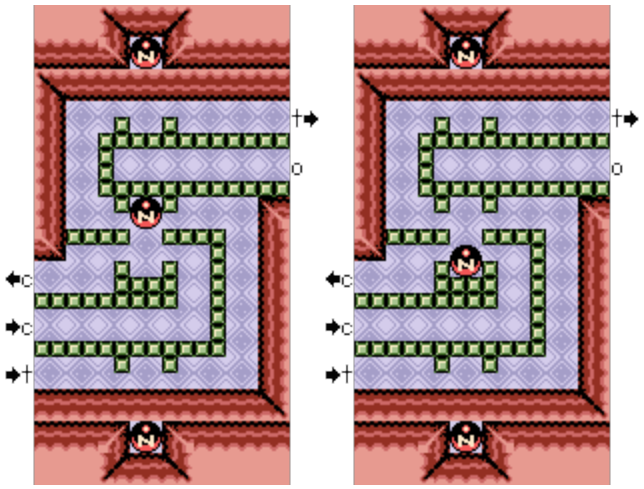


Figure 5-21: Construction of a door gadget using metal orbs, in the closed (left) and open (right) configuration. The open, traverse, and close paths are marked with directions.

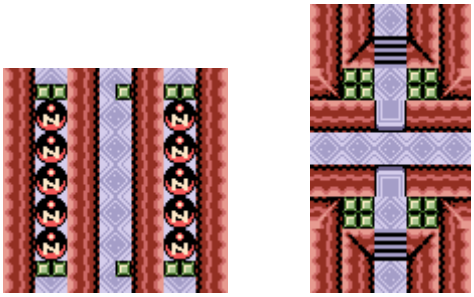


Figure 5-22: (left) Path lined with metal orbs to prevent Link from using the magnetic gloves while facing perpendicular into the path. (right) Crossover using jump platforms.

Theorem 135. *Generalized 2D Zelda with at least 15-tile range magnetic gloves, metal orbs, ledges, and jump platforms is* PSPACE*-hard.*

Proof. Compared to infinite range, having a maximum force distance permits black-box gadget constructions, as we prevent external interference by laying-out gadgets far apart in the dungeon. However, the construction in Theorem 134 is not self-sufficient because we protected the central metal orb from the left or right by using a single, distant hallway with orbs poised to block traversal to the goal.

We bring these two aspects together by compacting the door gadget enough to run blocking hallways on both sides, as shown in Figure 5-23. With this 11-tile-wide construction, the metal orbs above the hallways can be placed within the 15-tile limit to protect against

horizontal magnetic glove usage on the central metal orb. Rather than running the goal hallway around the outside of the room, we thread it past every gadget on both sides, completing the reduction. □

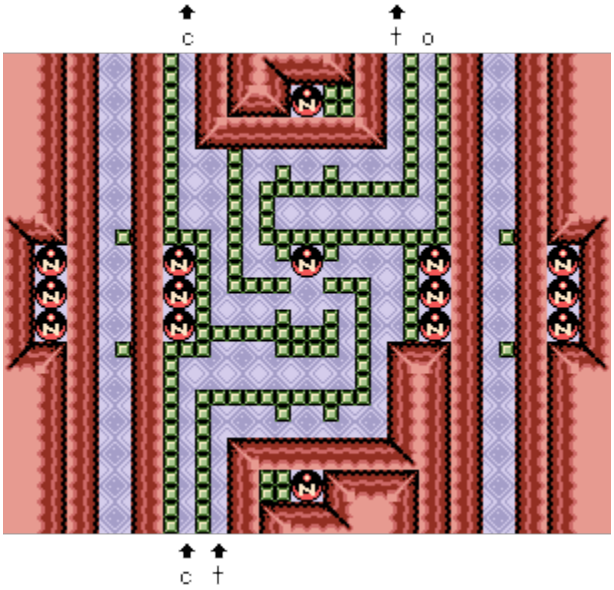


Figure 5-23: Compact construction of a door for 15-tile range magnetic gloves. Hallways on the left and right are traversed at the end to reach the goal.

5.3.3 Cane of Pacci is PSPACE-hard

The Cane of Pacci is an item introduced in The Legend of Zelda: The Minish Cap, a 2D game, that shoots a bolt of magic that can enchant a circular hole tile, which will launch Link up an adjacent ledge if he enters the hole. As a pseudo-3D effect, the bolt ignores hole tiles that are not "vertically aligned" with Link's feet: if the bolt travels down a ledge, then the bolt will remember that it is now high above the floor. The bolt also ignores already-enchanted holes. In the game, the hole stays enchanted for a significant but limited time, so we consider both the finite- and infinite-duration generalizations.

Theorem 136. *Generalized 2D Zelda with fixed-duration Cane of Pacci, ground holes, ledges, and tunnels is fixed-parameter tractable with respect to cane duration.*

Proof. Let the Cane of Pacci enchant holes for t frames before they automatically unenchant, and let Link's running speed be at most $v \leq 1$ tiles per frame, which is slower than the bolt's travel speed u.

For Link to use an enchanted hole, he must be within a circle of radius vt tiles centered at the hole from the duration of the enchantment. Symmetrically, all holes that are beyond vt tiles from his location cannot be enchanted and used, so without loss of generality no strategy for beating the dungeon ever has more than $h = O(v^2t^2) = O(t^2)$ holes that are enchanted at any point.

Supposing that there are n square tiles in the world and Link moves at a speed of 1 pixel per frame, he can be at $O(n/v^2)$ possible positions. Link can fire at most one bolt per frame, and each bolt that enchants a reachable hole travels for at most $vt/u < t$ frames. Under efficient play, where bolts are only ever shot at reachable holes, the total number of game configurations would be $O(n/v^2 \times ht \times (t+1)^h) = n\,(t+1)^{O(t^2)}$.

Therefore, we can create a graph in linear time for fixed t, where each node is such a configuration of enchanted holes and to-be-enchanted holes around Link's location, connected by edges representing the effects of possible player inputs on the next frame: Link moving, Link shooting a bolt at a hole in view, or a bolt enchanting a hole. There will be a strategy to get to the end of the dungeon if and only if this graph has a path from the starting configuration node and an ending configuration node. □

Theorem 137. *Generalized 2D Zelda with infinite-duration Cane of Pacci, ground holes, ledges, and tunnels is* PSPACE*-hard.*

Proof. To show PSPACE-hardness, we reduce from motion planning with self-closing doors [7]. Figure 5-24 shows our design for a self-closing door gadget. Link opens the door by entering the open path and firing the Cane of Pacci over the stone barrier at the hole below the ledge. When open, Link can later traverse by hopping from hole to hole, and the last hole will launch Link up the ledge, disabling the enchantment and thus closing the door behind him. The walls surrounding the holes, and the fact that the cane's bolt does not travel down to lower height levels when shot from the top of a ledge, prevent Link from opening the door

anywhere except the open path. Because the enchantment does not have a finite duration, Link may be required to open a door but not return to use the door for an arbitrarily long time.

To lay out the graph of self-closing door gadgets in the game, we can make use of the crossover gadget, also shown in Figure 5-24, if the graph is not planar. Link can freely travel north or south on the upper level, and another path may run left and right by going down stairs and using a tunnel on the lower level. □

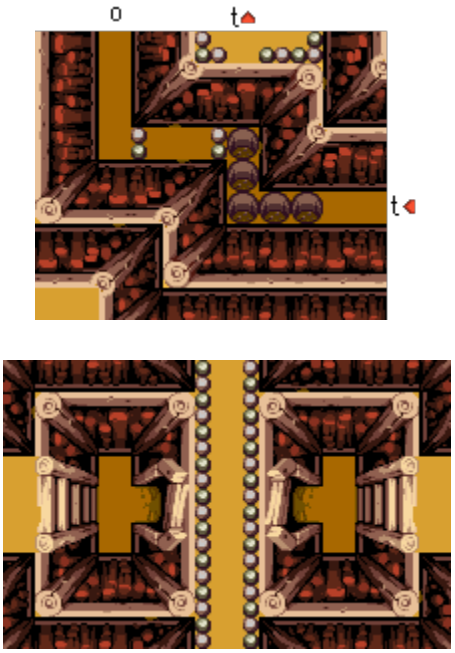


Figure 5-24: Gadgets in The Minish Cap: a self-closing door using holes for the Cane of Pacci (top) and a crossover using tunnels (bottom).

5.3.4 Magnesis Rune is PSPACE-Hard

In The Legend of Zelda: Breath of the Wild, a 3D game, Link obtains the multi-purpose Sheikah Slate, a tool that can be equipped with magical abilities called Runes. Among them is the Magnesis rune, which grants Link telekinetic power over metallic objects within a fixed distance. Compared to the magnetic gloves described in Section 5.3.2, Magnesis provides full 3D control of exactly one targeted metal object in a world with more-advanced simulated

physics, although Link cannot target objects that are out of his line-of-sight or that he is standing on.

Theorem 138. *Generalized 3D Zelda with the Magnesis rune and large metal plates is* PSPACE*-hard.*

Proof. We reduce from motion planning with self-closing doors [7], using the gadget illustrated in Figure 5-25. Within a closed room, we construct two paths of platforms over pits: the traverse line, with two gaps that can only be crossed by placing a large metal plate as a bridge, and the open line, raised above the first close enough to use Magnesis on the plate but too far to use it as a bridge to cross paths. Both paths connect to the outside with small exit doors to keep the large metal plate inside.

The self-closing door starts closed, where the large metal plate is not within Magnesis reach of the start of the traverse line. To open the door, Link must use Magnesis from the open line to relocate the plate so that when Link later enters the traverse line, he can use the plate as a bridge across both gaps. Carrying the plate from the first gap to the second gap puts it out of Magnesis range of the entrance of the traverse line, which closes the door upon traversal. □

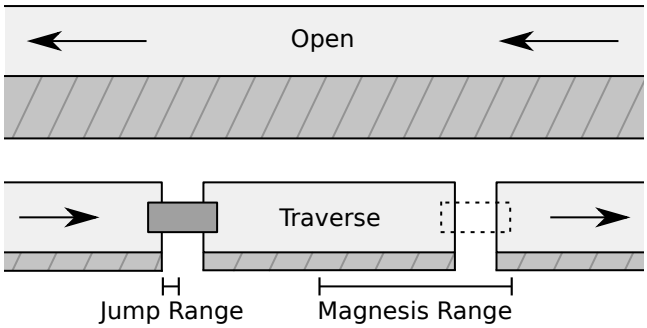


Figure 5-25: Construction of a door gadget using a large metal plate and platforms over pits, shown in the open state. The open line is raised above the traverse line. The layout was inspired by a puzzle in the Oman Au Shrine where the Magnesis rune is unlocked in The Legend of Zelda: Breath of the Wild.

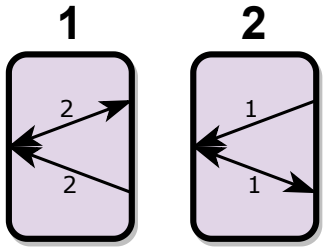


Figure 5-26: The Trainyard gadget.

5.3.5 Trainyard

The study of the complexity of Trainyard began with [3], which showed that finding a solution to a Trainyard level is NP-hard. Later, [2] showed that checking a solution to a Trainyard level is PSPACE-complete—verifying solutions may be harder than finding them. We improve on this result by showing that Trainyard is PSPACE-hard even with only one train, and with no color changes.

Trainyard is a puzzle game in which the goal is to build a system of rails so that trains of the correct colors reach certain stations. We consider one-train colorless Trainyard, where solutions consist of only rails, crossings, and ***switches***. There is a single train which moves forwards along the rails; it succeeds if it reaches a designated location, and crashes and fails if it the track it is on ends. Rails can be traversed in both directions.

The only nontrivial behavior comes from switches, which have two states. A switch changes state every time the train moves through it. It has three locations: two of them always route the train to the third, and the third routes the train to one of the first two depending on the state. We can model this as a toggle line/toggle line/toggle switch with some locations identified; we call this the ***Trainyard gadget***, which is shown in Figure 5-26. Since tracks can bend and cross each other, the planarity of a system of Trainyard gadgets does not matter. Now one-train colorless Trainyard is equivalent to one-player motion planning with the Trainyard gadget—except that the Trainyard gadget is not input/output, so we have not defined one-player motion planning with it.

Definition 139. ***One-player motion planning with the Trainyard gadget*** *takes place in a system of Trainyard gadgets where the connection graph is a partial matching. That is,*

each location is either paired with one other location or a ***dead end****.*

A robot moves through the system similarly to with input/output gadgets. When it enters a Trainyard gadget, it takes the unique available transition. When it exits a Trainyard gadget, it moves to the unique paired location, or stops if it is at a dead end.

Theorem 140. *One-player motion planning with the Trainyard gadget, or equivalently one-train colorless Trainyard, is* PSPACE*-hard.*

Proof. We will reduce from one-player motion planning with the toggle switch/toggle line. We can not quite directly simulate a toggle switch/toggle line, for a few reasons:

- The Trainyard gadget, and thus any gadget simulated by it, can be entered at any location, not just input locations. To account for this, we will denote some vertices in the simulation as input and output, and the arrangement of gadgets will ensure that the robot always enters simulated gadgets at input-denoted locations and exits and output-denoted locations. In particular, output-denoted locations always lead to input-denoted locations.

- One-player motion planning with the Trainyard gadget does not include fan-ins. However, we can easily simulate fan-in in the above sense by denoting two locations as input and one as output on the Trainyard gadget—the Trainyard gadget is a fan-in provided the robot never enters at one location.

- Even with the above caveats, we have not been able to simulate the toggle switch/toggle line (or any unbounded output-disjoint deterministic 2-state input/output gadget with multiple nontrivial inputs) with the Trainyard gadget. Instead, we simulate a toggle switch/toggle line for ***exponentially long***. Formally, we describe a network of Trainyard gadgets for each natural number k such that the kth network has the same behavior as the toggle switch/toggle line for at least 2^k transitions, and contains a number of Trainyard gadgets polynomial in k. Consider a system of n toggle switch/toggle lines from the reduction showing they are hard. The system has at most 2^n configurations and $5n$ locations for the robot; thus after at most $5n2^n$ transitions the robot

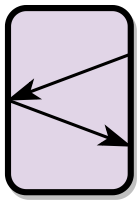


Figure 5-27: The reverse branch gadget.

reaches either the goal location or the dead end at False-Out on the first quantifier. If we pick polynomial k such that $2^k > 5n2^n$ (e.g., $k = 2n + 3$ suffices), then the network of Trainyard gadgets we obtain by replacing each toggle switch/toggle line with the kth simulation has the same behavior long enough for the robot to either reach the goal location or crash. Hence these exponentially long simulations suffice for PSPACE-hardness.

Thus it suffices to find an exponentially long simulation of the toggle switch/toggle line. Before describing this simulation, we present an exponentially long simulation of an intermediate gadget called the ***reverse branch***, shown in Figure 5-27. This has one state and three locations, of which we assume is never entered, one will never be exited, and one is both exited and entered.

Our exponentially long simulation of a reverse branch is shown in Figure 5-28. The k gadgets in the bottom row serve as fan-ins, since we assume the robot never enters at the bottom right. Consider the states of the top row of $k + 1$ gadgets as describing a number in binary: up (state 1) is 0, down (state 2) is 1, and the bits are read right to left. When the robot enters at the left, it increments this number (mod 2^{k+1}) and exits at the bottom right, unless the states are all up so the number is 0, in which case it exits the top right. When the robot enters at the top right, it flips the state of every gadget in the top row and exits at the left; this changes the number by $x \mapsto -x - 1$. In particular, the distance from 0 changes by at most 1 with each transition. By starting at 2^k as in Figure 5-28, it takes at least 2^k transitions to reach 0, so the simulation is correct for 2^k transitions.

Now we simulate a toggle switch/toggle line using a Trainyard gadget and two reverse branches, as shown in Figure 5-29. When the robot enters the top, it exits the top right,

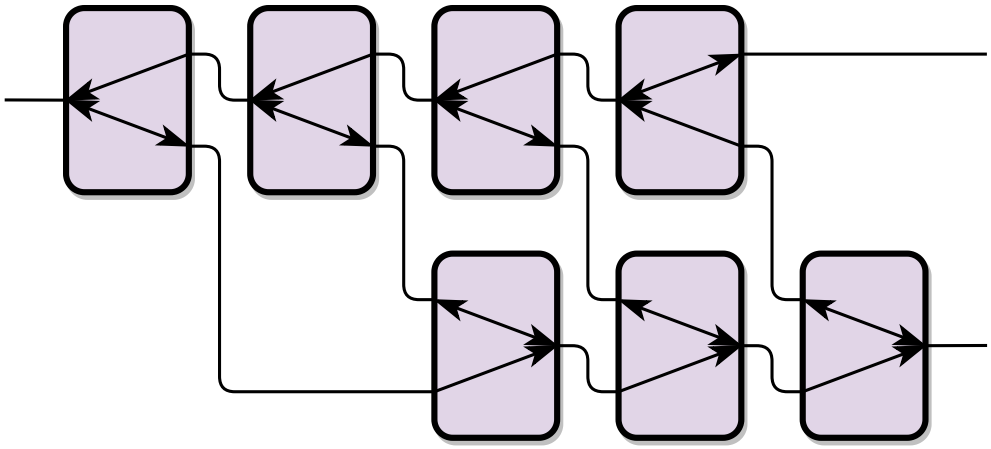


Figure 5-28: An exponentially long simulation of a reverse branch using Trainyard gadgets.

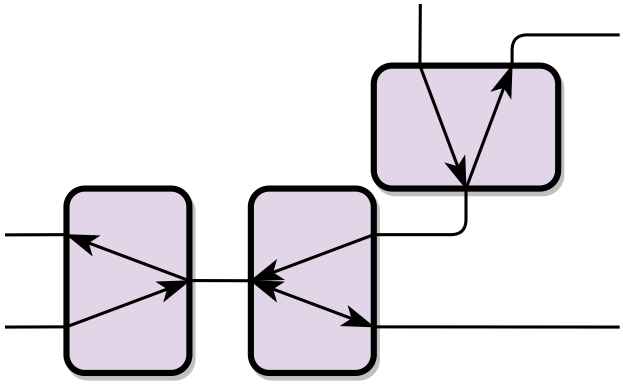


Figure 5-29: A simulation of a toggle switch/toggle line using a Trainyard gadget and reverse branches.

flipping the state of the Trainyard gadget (in the middle); this is the toggle line. When the robot enters the bottom right, it exits at the top left or bottom left depending on the state of the Trainyard gadget, and flips the state; this is the toggle switch. Each transition through the simulated gadget makes at most one transition through each reverse branch, so if the reverse branches are correct for 2^k transitions, so is the toggle switch/toggle line. □

5.4 Simplifying Prior Proofs

In this section we give examples where our motion planning problems can be used to simplify proofs of already known results. Using results from Section 3.3.1 the proof of PSPACE-

hardness for Mario Kart can be reduced to a single simple construction. Most applications of the Doors Framework required the construction of crossovers, which can now be eliminated. Many of the gadgets from past block pushing proofs can also be eliminated. Finally, as noted earlier, proofs using the Mario/Portal framework can be simplified by using door opening gadgets.

5.4.1 Two Player Mario Kart

Mario Kart is a popular Nintendo racing game whose computational complexity was considered in [13] which showed NP-completeness for 1 player races and PSPACE-completeness for 2 player races with reductions from 3SAT and QSAT respectively. Using results from this thesis, the 2 player proof now only needs a single, simple gadget, reducing a several page proof to a paragraph.

This section comes from [29] written in collaboration with Erik Demaine and Dylan Hendrickson.

Theorem 141. *Deciding if a player can force a win in two player Mario Kart is* PSPACE*-hard.*

Proof. A single-use one-way gadget can be constructed from a ramp and Dash Mushroom in Mario Kart. We place a ramp before a gap in the track long enough that a racer going at the normal maximum speed will not be able to make the jump and will fall onto another track that will take a long time to reach the finish line, ensuring they lose. However, this gap is small enough that, if the player uses a Dash Mushroom before, the increase in speed will allow them to make the jump. We put a single Dash Mushroom power-up before each ramp, ensuring the first racer to arrive can pick up the item and use it to cross the gap. To ensure a racer does not pick up the item and then keep it for later use, we precede the mushroom and ramp with a one-way gadget implemented by a long-fall. Along with the trivial existence of crossovers and the finish line as a location based win condition, Mario Kart is PSPACE-hard by Theorem 96. □

5.4.2 Planar NAND

Some prior results use forced distant closings as part of their reduction from 3SAT. Here we observe that several of those construct NAND or door closing gadgets and under our new framework those parts of the past constructions alone would be sufficient for proving hardness. In each case this would significantly simplify the proofs needed.

In [22] Push-1 block pushing puzzles are shown to be NP-hard. Inspecting the reduction from 3SAT we see the construction of an "H-gadget" and a "no-reverse" gadget. We would now call the no-reverse gadget a diode and the H-gadget a parallel undirected door closing gadget. From Lemma 49 we now know that these two gadgets would suffice for NP-hardness. Similarly, to deal with planarity they give the "XOR Crossover" which we would call a crossing undirected door closing gadget, and once again we now know this gadget with the no-reverse gadget suffices for NP-hardness. This eliminates the need for the "one-way" and "fork" gadgets, as well as the most complicated construction, the "Locked Door" gadget.

Similarly, in [37] we can now eliminate the need to construct three of the four gadgets given in that proof, as the crossing NAND is now known to be the only one needed. In fairness [37] and other prior more complicated constructions provided the ideas and techniques that allowed us to prove our results.

Finally, examining the proof that Pokemon is NP-hard from [5], it is clear that their clause gadget is constructed from three undirected door closing gadgets. By combining that with their single-use path gadget, an anti-parallel NAND gadget can be constructed yielding NP-hardness by Lemma 48. This avoids the need for a crossover gadget, which is by far the most complicated part of that reduction.

5.4.3 Planar Doors

Our planar door results simplify prior uses of a door framework for 2D applications that previously needed to use crossover gadgets. Here is a list of prior uses of the door framework which could benefit from this result:

- The Lemmings door [16, Figure 4] has an internal crossing, so Theorem 83 applies.

- The Donkey Kong Country 1, 2, and 3 doors [5, Figures 21–23] are the Case 10: OtCcT door, Case 4: OTtcC door, and internal crossing door, respectively, so Theorems 85 and 83 applies.

- The Legend of Zelda: A Link to the Past door [5, Figure 30] has an internal crossing, so Theorem 83 applies.

- The Super Mario Bros. door [32, Figure 6] is the Case 4: OTtcC door, so Theorem 85 applies.

- The Witness door [1, Figure 50] is undirected with an optional open port, so Theorem 85 applies.

- The Fire Emblem door [39, Figure 5] is the Case 3: OCcTt door so Theorem 85 applies. are not in fact needed to prove PSPACE-hardness of these games.

Bibliography

[1] Zachary Abel, Jeffrey Bosboom, Michael Coulombe, Erik D Demaine, Linus Hamilton, Adam Hesterberg, Justin Kopinsky, Jayson Lynch, Mikhail Rudoy, and Clemens Thielen. Who witnesses The Witness? Finding witnesses in The Witness is hard and sometimes impossible. *Theoretical Computer Science*, 2020.

[2] Matteo Almanza, Stefano Leucci, and Alessandro Panconesi. Tracks from hell – when finding a proof may be easier than checking it. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *LIPIcs*, pages 4:1–4:13, La Maddalena, Italy, June 2018.

[3] Matteo Almanza, Stefano Leucci, and Alessandro Panconesi. Trainyard is NP-hard. *Theoretical Computer Science*, 748:66–76, 2018.

[4] Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (NP-)hard. In *Proceedings of the 7th International Conference on Fun with Algorithms (FUN 2014)*, Lipari Island, Italy, July 2014.

[5] Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[6] Joshua Ani, Sualeh Asif, Erik D Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. PSPACE-completeness of pulling blocks to reach a goal. *arXiv preprint arXiv:2006.04337*, 2020.

[7] Joshua Ani, Jeffrey Bosboom, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, and Jayson Lynch. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020)*, Favignana, Italy, September 2020.

[8] Joshua Ani, Erik D. Demaine, Yevhenii Diomidov, Dylan H. Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. *preprint*, 2020.

[9] Joshua Ani, Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. *CoRR*, abs/2005.03192, 2020.

[10] Jose Balanza-Martinez, Timothy Gomez, David Caballero, Austin Luchsinger, Angel A Cantu, Rene Reyes, Mauricio Flores, Robert Schweller, and Tim Wylie. Hierarchical shape construction and complexity for slidable polyominoes under uniform external forces. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2625–2641. SIAM, 2020.

[11] Jose Balanza-Martinez, Austin Luchsinger, David Caballero, Rene Reyes, Angel A Cantu, Robert Schweller, Luis Angel Garcia, and Tim Wylie. Full tilt: Universal constructors for general shapes with uniform external forces. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2689–2708. SIAM, 2019.

[12] Jeffrey Bosboom, Michael Coulombe, Erik D Demaine, Dylan Hendrickson, Jayson Lynch, and Lorenzo Najt. The Legend of Zelda: The Complexity of Mechanics. *In preparation*, 2020.

[13] Jeffrey Bosboom, Erik D Demaine, Adam Hesterberg, Jayson Lynch, and Erik Waingarten. Mario Kart is hard. In *Japanese Conference on Discrete and Computational Geometry and Graphs*, pages 49–59. Springer, 2015.

[14] Kevin Buchin and Dirk HP Gerrits. Dynamic point labeling is strongly PSPACE-complete. *International Journal of Computational Geometry & Applications*, 24(04):373–395, 2014.

[15] David Caballero, Angel A. Cantu, Timothy Gomez, Austin Luchsinger, Robert Schweller, and Tim Wylie. Relocating units in robot swarms with uniform control signals is pspace-complete. In *Proceedings of the 32th Canadian Conference on Computational Geometry, 2020*, 2020.

[16] G. Cormode. The hardness of the Lemmings game, or oh no, more NP-completeness proofs. In *Proceedings of Third International Conference on Fun with Algorithms*, pages 65–76, 2004.

[17] Diogo M Costa. Computational complexity of games and puzzles. *arXiv preprint arXiv:1807.04724*, 2018.

[18] Joseph Culberson. Sokoban is PSPACE-complete. In *Proceedings of the International Conference on Fun with Algorithms*, pages 65–76, Elba, Italy, June 1998.

[19] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, Mar 1990.

[20] Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O'Rourke. Pushing blocks is hard. *Computational Geometry: Theory and Applications*, 26(1):21–36, August 2003.

[21] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Canadian Conference on Computational Geometry, Fredericton, New Brunswick, Canada, August 16-19, 2000*, 2000.

[22] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry (CCCG 2000)*, pages 211–219, Fredericton, New Brunswick, Canada, August 2000.

[23] Erik D. Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. In *Proceedings of the 10th International Conference on Algorithms and Complexity*, volume 10236 of *Lecture Notes in Computer Science*, pages 177–195, Athens, Greece, May 2017.

[24] Erik D. Demaine, Isaac Grosof, and Jayson Lynch. Push-pull block puzzles are hard. In *Proceedings of the 10th International Conference on Algorithms and Complexity*, pages 177–195, Athens, Greece, May 2017.

[25] Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *LIPIcs*, pages 18:1–18:21, La Maddalena, Italy, June 2018.

[26] Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 18:1–18:21, La Maddalena, Italy, June 2018.

[27] Erik D. Demaine and Robert A. Hearn. Constraint Logic: A uniform framework for modeling computation as games. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, pages 149–162, June 2008.

[28] Erik D. Demaine, Robert A. Hearn, and Michael Hoffmann. Push-2-F is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, pages 31–35, Lethbridge, Canada, August 2002.

[29] Erik D. Demaine, Dylan Hendrickson, and Jayson Lynch. Toward a general theory of motion planning complexity: Characterizing which gadgets make games hard. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science (ITCS 2020)*, pages 62:1–62:42, Seattle, Washington, January 2020.

[30] Erik D. Demaine, Michael Hoffmann, and Markus Holzer. PushPush-k is PSPACE-complete. In *Proceedings of the 3rd International Conference on Fun with Algorithms (FUN 2004)*, pages 159–170, Isola d'Elba, Italy, May 2004.

[31] Erik D Demaine, Joshua Lockhart, and Jayson Lynch. The computational complexity of portal and other 3d video games. In *9th International Conference on Fun with Algorithms (FUN 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[32] Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is harder/easier than we thought. In *Proceedings of the 8th International Conference on Fun with Algorithms*, pages 13:1–13:14, La Maddalena, Italy, June 8–10 2016.

[33] Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. Arrival: A zero-player graph game in NP ∩ coNP. In *A Journey Through Discrete Mathematics*, pages 367–374. Springer, 2017.

[34] John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *LIPIcs*, pages 124:1–124:14, Prague, Czech Republic, July 2018.

[35] Michal Forišek. Computational complexity of two-dimensional platform games. In *Proceedings International Conference on Fun with Algorithms (FUN 2010)*, pages 214–227, 2010.

[36] Michael P. Frank. Asynchronous ballistic reversible computing. In *Proceedings of the IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Washington, DC, November 2017.

[37] Erich Friedman. Pushing blocks in gravity is NP-hard. Unpublished manuscript, March 2002. https://www2.stetson.edu/~efriedma/papers/gravity.pdf.

[38] Jonathan Gabor and Aaron Williams. Switches are pspace-complete. In *CCCG*, pages 42–48, 2018.

[39] Jiawei Gao. The computational complexity of fire emblem series and similar tactical role-playing games. *arXiv preprint arXiv:1909.07816*, 2019.

[40] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[41] Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: next stop in CLS. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *LIPIcs*, pages 60:1–60:13, Prague, Czech Republic, July 2018.

[42] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. Limits to parallel computation: P-completeness theory, 1995.

[43] Alan Hazelden, Lee Shang Lun, and Allison Walker. Sokobond. https://www.sokobond.com/, 2014.

[44] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation.* A. K. Peters, Ltd., Natick, MA, USA, 2009.

[45] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation.* A K Peters/CRC Press, 2009.

[46] Robert A Hearn, Erik D Demaine, and Greg N Frederickson. Hinged dissection of polygons is hard. In *CCCG*, pages 98–102, 2003.

[47] Markus Holzer and Sebastian Jakobi. On the complexity of rolling block and alice mazes. In *International Conference on Fun with Algorithms*, pages 210–222. Springer, 2012.

[48] Dénes Kőnig. Über eine Schlussweise aus dem Endlichen ins Unendliche. *Acta Sci. Math. (Szeged)*, 3(2–3):121–130, 1927.

[49] Max Lindblad. How hard is Wings of Vi?: An analysis of the computational complexity of the game Wings of Vi, 2015.

[50] André G. Pereira, Marcus Ritt, and Luciana S. Buriol. Pull and PushPull are PSPACE-complete. *Theoretical Computer Science*, 628:50–61, 2016.

[51] Gary Peterson, John Reif, and Salman Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(7-8):957–992, 2001.

[52] Gary L Peterson and John H Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science*, pages 348–363. IEEE, 1979.

[53] Marcus Ritt. Motion planning with pull moves. arXiv:1008.2952, 2010. https://arXiv.org/abs/1008.2952.

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

[55] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC 1978, pages 216–226, New York, NY, USA, 1978. Association for Computing Machinery.

[56] Tommy Thompson. "With fate guiding my every move": The challenge of Spelunky. In *FDG*, 2015.

[57] Tatsuie Tsukiji and Takeo Hagiwara. Recognizing the repeatable configurations of time-reversible generalized langton's ant is pspace-hard. *Algorithms*, 4(1):1–15, 2011.

[58] Tom C Van Der Zanden and Hans L Bodlaender. Pspace-completeness of bloxorz and of games with 2-buttons. In *International Conference on Algorithms and Complexity*, pages 403–415. Springer, 2015.

[59] Video Game Sales Wiki. The Legend of Zelda. https://vgsales.fandom.com/wiki/The_Legend_of_Zelda, 2020.

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

[61] Giovanni Viglietta. Lemmings is PSPACE-complete. *Theoretical Computer Science*, 586:120–134, 2015.

[62] Wikipedia. Sokoban. https://en.wikipedia.org/wiki/Sokoban.

[63] Gordon Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3(1):131–150, 1991. Originally appeared at SoCG 1988.

[64] Zhujun Zhang. A Note on Computational Complexity of Dou Shou Qi. *arXiv preprint arXiv:1904.13205*, 2019.

[65] Zhujun Zhang. A note on hardness frameworks and computational complexity of Xiangqi and Janggi. *arXiv preprint arXiv:1904.00200*, 2019.