Further Hardness Results for Stephen's Sausage Roll

by

Jason Liu

S.B. Mathematics and Computer Science and Engineering, MIT, 2024

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

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Jason Liu Department of Electrical Engineering and Computer Science May 17, 2024
Certified by:	Erik D. Demaine Professor of Computer Science, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Further Hardness Results for Stephen's Sausage Roll

by

Jason Liu

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Stephen's Sausage Roll is a relatively unstudied puzzle game with a fascinating set of mechanics for computational hardness problems. The only past results are from a class project in MIT's 6.5440 class of Fall 2023, which only dealt with two specific subsets of the mechanics restricted to two-dimensional forms of the game [1]. This project presents a more complete characterization of problems based off of Stephen's Sausage Roll, and provides solutions for a significant portion. In particular, both variants of Stephen's Sausage Roll considered in prior work can be solved by one of these results.

Thesis supervisor: Erik D. Demaine
Title: Professor of Computer Science

Acknowledgments

I would like to acknowledge Nicole Lu, Chengyuan Ma, Richard Sollee, and Kenny Zhang for our work together on the earlier results in Stephen's Sausage Roll that much of this project is built on. I'd also like to thank Lily Chung for proposing the initial problem with Stephen's Sausage Roll, and Prof. Erik Demaine for advising me for this thesis. Finally, I'd like to thank my undergraduate advisor Bjorn Poonen for the advice he's given me over the years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computational hardness is the study of difficult problems and classifying exactly how challenging they are. While the field was created to study more practical algorithms and problems such as the Travelling Salesman Problem, significant work is also spent on more recreational problems: games. Much work has gone into proving the hardness of games across many genres: board games such as Konane or Amazons, platform games like Mario and Celeste, puzzle games such as Baba is You, and many many more [2]–[5]. While many questions could be asked about games including, but not limited to, ability to force a win, ability to force a loss (in two-player games), or perhaps constructing complicated states within a game, the primary question of interest is usually that of whether a player can guarantee a win. Despite the significant effort put into studying games, and even when victory is the only question of interest, the ever increasing number of games available to study means that there are always many more available to be studied and many cases within each game to consider.

In this thesis, we study a relatively unstudied game: Stephen's Sausage Roll, a puzzle game made in 2016 by Increpere games [6]. The game itself starts with relatively simple mechanics, but creates complex mechanics by taking advantage of every corner case that appears when you combine the behavior of multiple objects while trying to respect basic physics. As such, it makes for a fascinating game to study in the context of computational hardness, as even including just a few mechanics can create strange interactions that lead to perhaps harder problems than expected. The hardness of the game has, as of yet, no published results. The only results I know of are two unpublished results from Prof. Erik Demaine's 6.5440 class in Fall 2023, which prove PSPACE-completeness of the game for two subsets of the mechanics [1]. Nonetheless, there is much available to be studied, and this project aims to learn as much as possible about the game.

In a bit more detail, Stephen is approximately a $2 \times 1 \times 1$ (height 1) object in a three-dimensional world pushing $2 \times 1 \times 1$ sausages around the world to cook them over grills. While the world is three-dimensional, we can mostly think of it as stacked 2-dimensional layers as Stephen's movement is mostly in two dimensions. There are also two types of immovable objects of interest: walls and grills. Walls and grills both block movement, but grills will cook any sausages moving over them while burning Stephen's feet. We also consider empty spaces to be their own type of "block": layers without empty spaces are fully filled with walls, grills, and possibly also immobile sausages and players. Finally, the main exception to Stephen's two-dimensional movement is his ability to move on ladders. Ladders are attached

Layers containing						
Player	Sausages	Walls	Grills	Empty	Difficulty	Source
2	2	1	1	1, 2	PSPACE	[1, Theorem 3]
2	2	1, 2	1	2	PSPACE	[1, Theorem 4]
2	2	1	1	2	PSPACE	Theorem 6
3	2	1, 2	1	2, 3	PSPACE	Theorem 7
2	2, 3, 4	2, 3, 4	1, 2, 3, 4	1, 2, 3, 4	P	Lemma 8, Case 1
2	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4	1, 3, 4	P	Lemma 8, Case 2
2	4	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4	P	Lemma 8, Case 3
3	1, 2	1, 2, 3, 4	1, 2, 3, 4	1, 3, 4	P	Lemma 8, Case 4
3	1, 2, 3, 4	1, 2, 3, 4	3, 4	1, 2, 3, 4	P	Lemma 8, Case 5
2	1, 2, 3, 4	1, 2, 3, 4	2, 3, 4	2, 3, 4	P	Lemma 8, Case 6
3	1, 2	1, 2, 3, 4	2, 3, 4	1, 2, 3, 4	P	Lemma 8, Case 7
3	1, 2, 3, 4	1, 3, 4	2, 3, 4	1, 2, 3, 4	P	Lemma 8, Case 8
2	3, 4	1, 2, 4	1, 2, 4	1, 2, 3, 4	P	Lemma 8, Case 9

Table 1.1: PSPACE-hard and P cases of Stephen's Sausage Roll. For PSPACE-hard results we may add more options to each of the sausage, wall, grill, and empty columns to get more results. For P results we may remove options from each column to get more results. We assume that all options are available in Levels 5 and higher.

to walls and allow Stephen to climb to the top or bottom of ladders as appropriate.

1.1 Results

In this thesis, we propose a categorization of Stephen's Sausage Roll based on whether each type of block or object is present in each layer of a level. In particular, this thesis focuses on the case the level has no ladders, thereby constraining Stephen to mostly interact with a few neighboring layers.[1] As such, most results can be summarized with simply the parameterization of a few neighboring layers to whichever layer Stephen is in (the "player layer").[2]. See Table 1.1 for a summary of past results and results from this thesis. In this thesis, we prove several cases in which Stephen's Sausage Roll is PSPACE-hard, along with a number of results of cases where Stephen's Sausage Roll turns out to be in P due, in almost all cases, to the level being unsolvable if any sausages at all are present.

1.2 Related Work

In general, proving some simple computational upper bounds for many games isn't too challenging. For example, deciding whether victory is possible for single-player games that are guaranteed to end within a polynomial number of moves (such as Minesweeper) is clearly

[1]Stephen is technically still able to interact with faraway layers using stacked sausages.

[2]If ladders are included then in many cases determining hardness might need information about much more than just a few neighboring layers

an NP problem: a winning sequence of moves would clearly show that victory is possible. Similarly, single-player games with a potentially exponential numbers of moves but with a polynomial amount of game state (such as Rush Hour) belong in PSPACE. Indeed, such games belong in NPSPACE because a nondeterministic machine can verify the possibility of victory simply by guessing one move at a time and verifying that the sequence of moves leads to a victory condition, and Savitch's theorem [7] shows that PSPACE equals NPSPACE.

Whereas generic upper bounds are easy to find, generic lower bounds tend to be somewhat more challenging to prove due to the variety in different game mechanics. One important framework for proving lower bounds used by many modern PSPACE-hardness proofs is ***planar motion planning***. At a high level, (single-player) planar motion planning consists of a robot moving around through a collection of gadgets whose states can change as the robot interacts with them. Problems of interest include reachability, where the robot attempts to get from a start location to some target location, and reconfiguration, where the robot attempts to get each gadget into some target state. Early examples by Viglietta [8], although not explicitly called planar motion planning, proved PSPACE hardness when given a door that could only be opened and closed by stepping on specific pressure plates that could be arbitrarily far away or doors that were operated using buttons that changed the state of 3 doors at once.

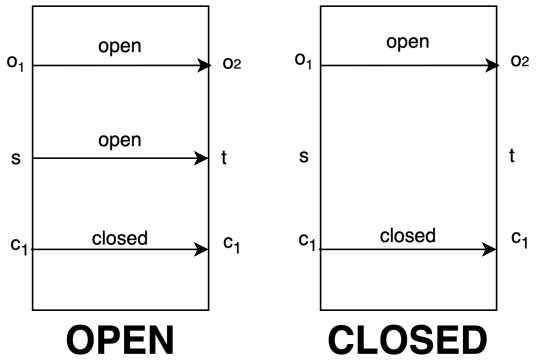
1.3 Gadget Theory

A ***gadget*** $G = (Q, L, T)$ is a finite collection of ***states*** Q, ***locations*** L, and some ***transitions*** $T \subset Q \times L \times Q \times L$ representing the ability to go from a given location in some initial state to some other location in some final state [9]. Transitions are usually denoted $(q_1, l_1) \to (q_2, l_2)$ to represent a transition starting from state q_1 in location l_1 and ending in state q_2 and location l_2. Gadgets can be summarized with ***gadget diagrams*** with one diagram per state, with transitions marked by arrows on the diagram corresponding to the starting state and labelled with their ending state. See Figure 1.1a for an example.

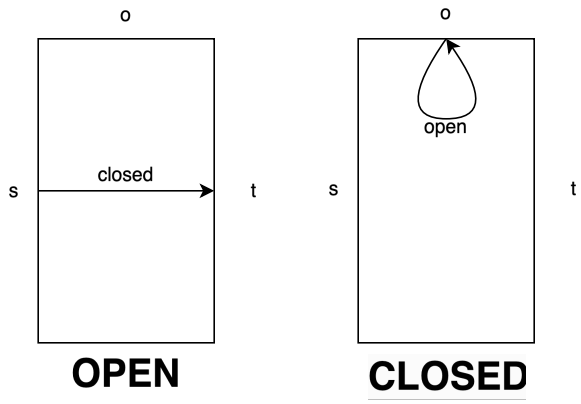
The entire problem is then modeled as a collection of gadgets, with connections between locations on (possibly different, possibly the same) gadgets that allows full traversal of a robot in either direction. Planar motion planning (for reachability) therefore is the question of whether a robot can reach an end location, given a starting location on a planar collection of gadgets and connections and starting states of each gadget [9].

While some gadgets were studied individually in prior works, gadget theory for motion planning was first studied in depth by Demaine, Hendrickson, and Lynch [9]. Of particular interest to this thesis however, is later work by Ani et al. [3] focusing specifically on proving hardness using only door gadgets.

An early door gadget is a door with opening and closing paths by Viglietta et al. [10], as seen in Figure 1.1a. This gadget has two states open and closed, six locations o_1, o_2, c_1, c_2, s, t, and transitions $(o_1, \cdot) \to (o_2, \text{open}), (c_1, \cdot) \to (c_2, \text{closed})$, and $(s, \text{open}) \to (t, \text{open})$ where each $\cdot$ can be either open or closed [10]. Viglietta et al. [10] showed that the existence of such doors along with crossovers is sufficient to prove PSPACE hardness. Later door gadgets expand on these doors either by changing some tunnels to be bidirectional or connecting some locations of the gadget.



(a) Directed door with opening and closing paths



(b) Gadget diagram for directed self-closing door

Of particular interest is the following result by Ani et al. [3] regarding open-optional directed self-closing doors.

Theorem 1 ([3, Corollary 3.4])**.** *1-player motion planning with any self-closing door is PSPACE-hard.*

Importantly, this theorem removes the need for crossover gadgets. Note also that while motion planning with self-closing doors is PSPACE-hard, both initially open and initially closed self-closing doors are needed to achieve PSPACE-hardness.

An ***open-optional directed self-closing door*** is, in essence, a door where locations t and c_1 are connected and locations o_1 and o_2 are connected. This gadget only has three locations o, s, t and has transitions $(o, \text{closed}) \to (o, \text{open})$ and $(s, \text{open}) \to (t, \text{closed})$. In other words, traversing through the door will automatically close the door which can only be opened via the opening location. While many other variants exist such as undirected and open-mandatory doors, this open-optional directed self-closing door is what we will need for our proof of PSPACE-hardness of Stephen's Sausage Roll. See Figure 1.1b for an example of a gadget diagram for a directed self-closing door.

Chapter 2

Stephen's Sausage Roll

Stephen's Sausage Roll is a puzzle game developed by Increpere games in 2016 [11]. In Stephen's Sausage Roll, the player controls Stephen in his quest to cook all sausages. The game starts with relatively simple mechanics, but creates complex mechanics by taking advantage of every corner case that appears when you combine the behavior of multiple objects while trying to respect basic physics. As such, it makes for a fascinating game to study in the context of computational hardness, as even including just a few mechanics can create strange interactions that lead to perhaps harder problems than expected.

2.1 Why Stephen's Sausage Roll?

While there are many games that can be studied using planar motion planning, Stephen's Sausage Roll still provides interesting new challenges via its mechanics. We first cover some of these challenges to defend the interest in this game before explaining the mechanics of the game.

First, each Stephen's Sausage Roll level involves a fixed number of sausages that must be cooked but not overcooked. In order to create reusable gadgets, cooking should presumably not be part of the main gadget use case and therefore the gadgets must either use fully uncooked sausages that are cooked afterwards in some sort of correction-checking procedure, fully cooked sausages that are pre-cooked in an initialization procedure, or a combination of some sort of partial pre-cooking and partial post-cooking. There exist prior results about adding a set of checking traversals to each gadget that could for example be used to post-cook every uncooked sausage [12]. Nonetheless, earlier work in MIT's 6.5440 class concluded that it was easier to reason about gadgets with fully cooked sausages and therefore this thesis, like prior work, will prove hardness while using gadgets that require initialization [1]. One part of the interest in Stephen's Sausage Roll is therefore the analysis and usage of previously unstudied gadgets that require initialization.

Secondly, Stephen's Sausage Roll levels can involve multiple layers, with players able to potentially affect objects many layers above or below the player. As such, even when player movement is restricted to a plane, Stephen's Sausage Roll allows for very interesting analysis of much more complex gadgets than many purely two-dimensional games.

Furthermore, Stephen's Sausage Roll includes ladders that simply allow the player itself

to move between layers. While our current constructions make minimal use of this mechanic, there are many cases remaining where it may be necessary to use this mechanic to produce even more complicated gadgets.

2.2 Game Mechanics [13]

Note that while we give an overview of most mechanics of the game for ease of reference for future work, the key mechanics for this thesis are sausage pushing and cooking as well as walking on a sausage. Sausage spearing is used in prior work, and ladders, as well as sausages on top of bodies or sausages are included because they are very interesting options for future exploration.

Stephen's Sausage Roll is, approximately, split into ***levels***, each consisting of a ***map*** and a set of movable ***objects*** fully within the level. The map is, in full generality, an $M \times M \times N$ lattice of cells with $M = \text{poly}(N)$, which we will often think of as N horizontal layers of size $M \times M$ each. Objects are orthogonally connected rigid sets of single-cell ***parts***, where rigidity means that the parts of an object will all translate in parallel when pushed, although some special objects have the additional ability to roll (sausages) or rotate (sausages and forks) in some form. In other words, taking C to be our set of all cell types, the state of a level can be described as an element of $C^{M \times M \times N}$ together with a list of objects, each with position, orientation, and for sausages, cooking status.

The most common objects are ***player*** and ***sausage*** objects. The player can be further subdivided into a $1 \times 1 \times 1$ ***body*** and a $1 \times 1 \times 1$ fork. The fork has a ***handle*** end and a ***spear*** end: under normal gameplay, the body and fork are always connected, with the handle side of the fork being the closer side of the fork to the body. Nonetheless, the fork and body may disconnect during gameplay under certain special circumstances. Like the body-fork combination, the sausage is also a $2 \times 1 \times 1$ object. Each of the two cells of the sausage has a two ***faces***: top and bottom. Aside from these standard objects, there are several other objects in the game, most notably the serpent and tree-island. I do not anticipate using these additional items, as the mechanics get significantly more complicated when implementing these.

The set C of cells contains empty space, walls (possibly with ladders), and grills. In particular, "land" as described in prior work is, in essence, walls that are one level below the player. Similarly, "water" is really just empty space that happens to be at the bottom of the map: in particular, the body, fork, and sausages are not allowed to fall out of the level. For completeness, we assume that everything outside of the level is empty space.

- Walls prevent a block from being travelled through by any object. Standing on them prevents falling, and trying to walk/move into them prevents movement.
- Ladders attached to the same side of a vertical stack of walls allows the player to climb the ladders up or down. Once on the ladder, the player has no control of their movement: they will attempt to go all the way along the ladder and exit it, and if movement is blocked they will return all the way to the starting point. Note that ladders may be attached to walls but not grills.

- Grills prevent objects from passing through them, and cook any sausages above them on the face of the sausage directly in contact with the grill. Due to their high temperature, if the body moves onto a grill it will immediately move back in the opposite direction from its initial movement.

- Empty space can be passed through freely by all objects.

While the blocks are relatively simple, the objects are somewhat less so. The player can move forwards and backwards and rotate ninety degrees around the fork. When holding the fork, the fork will always point in the forwards direction. Rotation requires a $2 \times 2 \times 1$ area to be empty: the current location of the body, the current and future locations of the fork, and the fourth square of the $2 \times 2 \times 1$ box outlined by these three squares. Whenever the player tries to move (other than falling) in such a way that the fork cannot stay attached, the move will not happen. When not holding the fork, the player can reattach the fork by moving the body to the handle and facing the fork. Otherwise, the fork can be pushed by any object attempting to move into its block.

When holding the fork, ladders can be climbed from the side: that is, the fork must be pointing neither towards nor away from the ladder. Without the fork, ladders are climbed from the front: the player must face the ladder when on the ladder.

The body may not attempt to move onto empty space. As such, the only non-ladder way to change the height of the player is to stand on a falling object. A disconnected fork may not fall below the level.

See Figure 2.1 for an example of how the objects and blocks look in game.

Sausages move away, if possible, from other objects that try to move into one of the two blocks of the sausage. If the incoming object approaches along the axis of the sausage or from below, it will slide out of the way. If the incoming object approaches from the two sides of the sausage, it will roll one block: the top face now becomes the bottom face and vice versa. If the sausage is unable to move, the object trying to move onto it will be blocked from moving. However, if said object is a fork approaching spear-first, then the fork will be embedded in the sausage. If the fork is not attached to a body, the sausage will still behave as before, but if the fork is attached to a body the sausage will move with the fork-body combination. The player will no longer be able to turn and the sausage can no longer roll, but the player gains the ability to walk directly to the left and right. If the sausage is blocked from movement, the player will also be blocked unless the player is moving backwards. If the player is moving backwards, the sausage will instead be detached from the fork. See Figure 2.2 for examples of how sausage rolling works.

No sausage may fall below the level.

The sausages can be cooked by grills. Each of the two faces (top, bottom) of each of the two blocks of the sausage can be cooked independently, and cooking the same face twice causes burning. The goal of the level is to cook all four faces of every sausage exactly one time each before returning to the exact position and orientation where the level started.

If an body on top of a sausage attempts to move perpendicular to the axis of the sausage, the sausage will attempt to roll in the opposite direction of the object's movement. As a result, a body on top of a sausage can only move off the sausage in a perpendicular direction to the axis if the sausage is prevented from rolling in either direction, and there is something

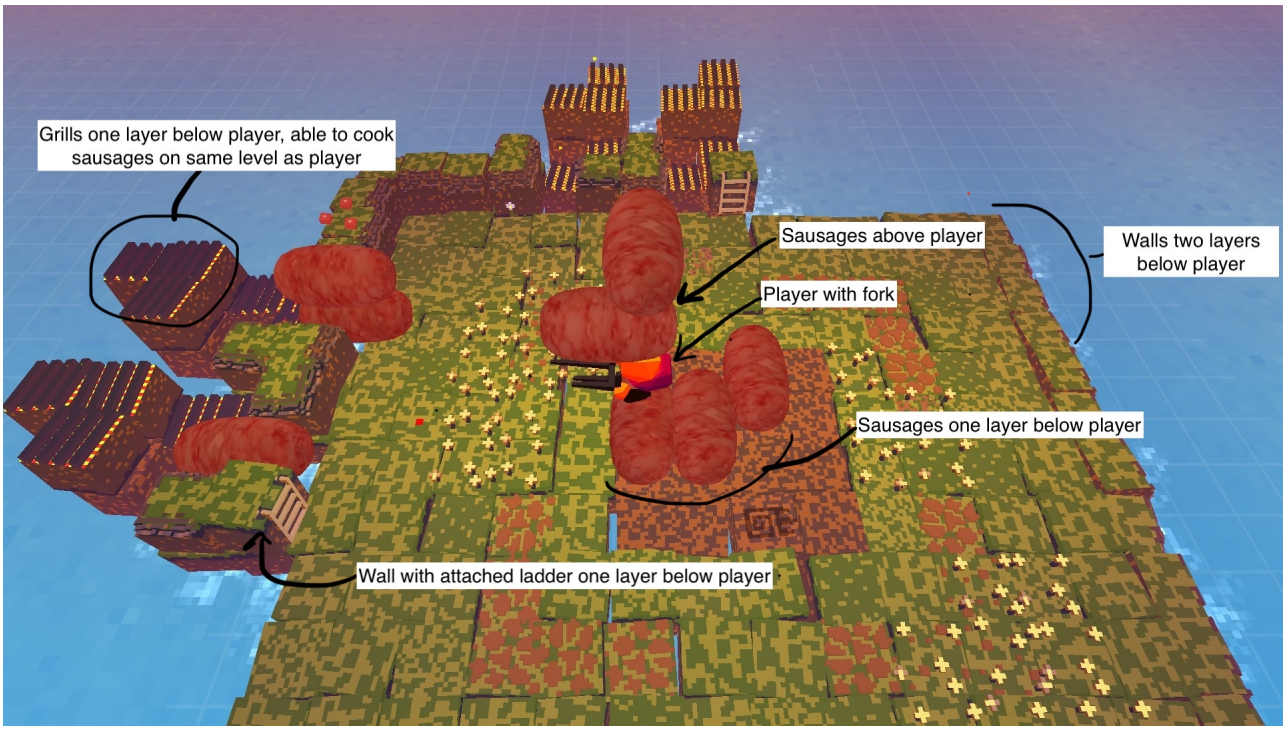


Figure 2.1: Sausages, player, and many blocks in a Stephen's Sausage Roll level
Screenshot of the game from the Steam page as the author does not currently have access to a copy of the game [11]

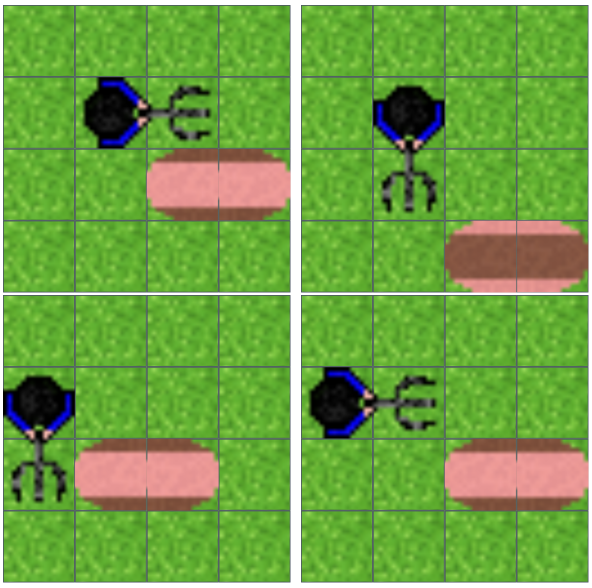


Figure 2.2: Pushing a half-cooked sausage along the long and short axes

for the body to step on. A body on top of a sausage can move off a sausage along the axis of the sausage as long as there is something for the body to step on.

Sausages on top of blocks or objects will not fall as long as one of the two blocks below the sausage is nonempty. If there is a non-moving block supporting the sausage (either an object not in motion or a wall/grill), the sausage will not be affected by any movement in the other supporting block. If the only block supporting the sausage is a fork object, then the sausage will stay on the fork when it makes translational moves (either by being pushed or when the player walks forwards and backwards). If the fork instead rotates around the player, the sausage will fall off without moving in the x, y directions. If the only blocks supporting the sausage are the body and optionally an attached fork, then the sausage will stay on the body's head and rotate, if possible, when the player rotates. However, blocking the sausage's movement will not block the player's movement rotationally or translationally, although it can stop the player from climbing up a ladder. In essence, the sausage moves with the player but is not attached: the only reason climbing can be blocked is because the sausage is a physical object in the way of climbing up further. If a sausage is supported by moving sausages, then if either lower sausage is moving along its axis the upper sausage will stay at the same position relative to the lower sausages. Otherwise, if the upper sausage is parallel to the lower sausages supporting it, it will also stay at the same relative position. However, if the upper sausage is perpendicular to the lower sausages, and the lower sausages are rolling instead of sliding, then the upper sausage will move two blocks for every one block that the lower sausages move.

At the start of each level, each object must be resting on some object or nonempty block. Furthermore, since there are no examples in the original game of a sausage starting on top of a grill, we further require that no sausages start on top of a grill since the behavior of such sausages is unclear.

The goal of a level is simply to cook all sausages without dropping them into the water and return the player and fork to the original position in the original orientation. In particular, if the original position is on top of a sausage then the player must return to standing on the same side of that same sausage, but the sausage itself need not be in its original position or orientation.

To explore some of these mechanics in a two-dimensional version of the game without "walls," there exists a fan-made demake of the game [14].

2.3 Existing results [1]

As a single-player game with unbounded moves but polynomial state, Stephen's Sausage Roll levels can be solved nondeterministically using polynomial space. Thus determining solvability is an NPSPACE problem and hence by Savitch's theorem it is a PSPACE problem [7].

Theorem 2 ([1, Theorem 1])**.** *Solvability of a general Stephen's Sausage Roll level is in PSPACE.*

Prior work on proving PSPACE-completeness of subsets of the mechanics rely on the

difficulty of planar motion planning given certain kinds of self-closing door gadgets [1], [3]. However, the game was only considered when restricted to essentially a two-dimensional level. The two specific cases considered were when a $M \times M \times 2$ level has only empty blocks on the top half, with possible empty blocks in the lower half as well, and a $M \times M \times 2$ level with no empty blocks in the bottom half and possible wall blocks in the top half. These were described as "water, grills, and land" and "walls, grills, and land," respectively [1]. Prior work has also created a level editor for the planar version of the game, including gadget-checking under the assumptions that sausages are never left partially cooked when leaving a gadget [15]. Unfortunately it turns out that this is an invalid assumption, so gadgets in this thesis are checked manually without the assistance of a gadget-checker.

2.4 Problem Framework

We consider a variety of Stephen's Sausage Roll problems, paramaterized by the following:

- An integer $2 \leq s \leq N$ denoting the layer where the player starts
- Whether we allow ladders in the level
- N subsets $S_1, S_2, \ldots, S_N \subseteq C \cup \{\text{sausages}\}$ denoting the types of blocks and objects (aside from possibly the player) that are initially found in each layer

Note that when all other parameters are fixed, PSPACE-hardness is preserved when turning allowing ladders in a level where they were previously banned simply by adding unusable ladders to the construction. Likewise PSPACE-hardness is preserved when adding more elements to any S_i (as these elements don't need to be used). In contrast, solvability in P is not preserved under these operations.

Note that banning ladders and selecting parameters $s = 2$, $S_1 = \{\text{walls, grills, empty}\}$, $S_2 = \{\text{empty, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$ reproduces the "water, grills, and land" problem. Similarly, banning ladders and setting $s = 2$, $S_1 = \{\text{walls, grills}\}$, $S_2 = \{\text{empty, walls, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$ reproduces the "walls, grills, and land" problem. Finally, banning ladders and setting $s = 2$, $S_1 = \{\text{walls, grills}\}$, $S_2 = \{\text{empty, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$ creates a "land and grills" problem, which is the probably most general case previously unsolved.

Chapter 3

Results

3.1 Initialization-required gadgets

While prior work has a theorem for initialization-required gadgets, we also include the proof of this theorem here for completeness [1]. Furthermore, this thesis uses a relaxed definition of initialization-required gadgets: prior work required that the only traversal starting from an uninitialized state would go from uninitialized to initialized, whereas in this thesis we only require that the only transition from a certain uninitialized entry point to a certain other location will initialize the gadget. In particular, transitions entering an uninitialized gadget from other locations may be allowed to produce arbitrary pathological behavior. We define four gadgets: an initialization-first directed self-closing door, an initialization-needed directed self-closing door and two types of initialization-needed opening door. For comparison, we provide gadget diagrams in Figures 3.1 and 3.2 of some versions of the initialization-needed gadgets for comparison with Figure 1.1b. We have two types of each gadget because both forms are useful in different proofs: the close-first, for example, will be used in the "Land and Grills" proof while the no-close was used in both the "Land, Grills, and Water" and "Land, Grills, and Walls" proofs. The key commonality between the two is that a visit to s followed by a visit to o can transform the gadget into a freely traversable $s - t$ path and that there is never any way to reach o from s or t or vice versa.

Definition 1. An initialization-first directed self-closing door is a gadget with states including uninitialized, open, and closed, and locations s, t, i, and o. We require the transitions $(i, \text{uninitialized}) \to (i, \text{open}), (s, \text{open}) \to (t, \text{closed}), (o, \text{closed}) \to (o, \text{open})$. We also require there are no location-changing transitions starting or ending at i, no other transitions starting from an uninitialized state, no other location-changing transitions starting from either an open or closed state, no transitions from either open or closed to any state that is not open or closed, and no transitions starting at s or t from closed to open.[1]Essentially, the gadget behaves as a standard directed self-closing door after being initialized from an external path.

Definition 2. An initialization-needed directed self-closing door is a gadget with states including uninitialized, open, and closed, and locations s, t, and o. We require the transitions $(s, \text{uninitalized}) \to (t, \text{closed}), (s, \text{open}) \to (t, \text{closed}), (o, \text{closed}) \to (o, \text{open})$. We also require there are no other transitions starting from s in the uninitialized state to either o or t, no

other location-changing transitions starting from either an open or closed state, no transitions from either open or closed to any state that is not open or closed, and no other transitions from closed to open.[1] Essentially, the gadget behaves as a standard directed self-closing door after being initialized via an $s - t$ traversal.

Definition 3. A close-first initialization-needed opening door is a gadget with states including uninitialized, open, and closed, and locations s, t, and o. We require transitions $(s, \text{open}) \to (t, \text{open})$, $(t, \text{open}) \to (s, \text{open})$, $(s, \text{uninitialized}) \to (s, \text{closed})$, and $(o, \text{closed}) \to (o, \text{open})$. We also require that there are no other transitions starting with either an open or closed state, no transitions starting from s in the uninitialized state going to t, and no other transitions either starting or ending at o. In particular, we allow the possibility of further transitions from s to itself that lead to other unspecified states.

Definition 4. A no-close initialization-needed opening door is a gadget with states including uninitialized and open, and locations s, t, and o. We require transitions $(s, \text{open}) \to (t, \text{open})$, $(t, \text{open}) \to (s, \text{open})$, and $(o, \text{uninitialized}) \to (o, \text{open})$. We also require that there are no other transitions starting with an open state, no transitions starting from s in the uninitialized state going to t, and no other transitions either starting or ending at o. In particular, we allow the possibility of further transitions from s to itself that lead to other unspecified states.

These gadgets are sufficient to produce PSPACE-hardness, as shown in the following result analagous to Lemma 2 of the unpublished previous Stephen's Sausage Roll paper [1, Lemma 2].

Theorem 3. *Planar motion planning with initialization-needed directed self-closing doors and either form of initialization-needed opening doors is PSPACE-hard.*

Proof. We reduce from planar motion planning with directed self-closing doors which is PSPACE hard due to Theorem 1.

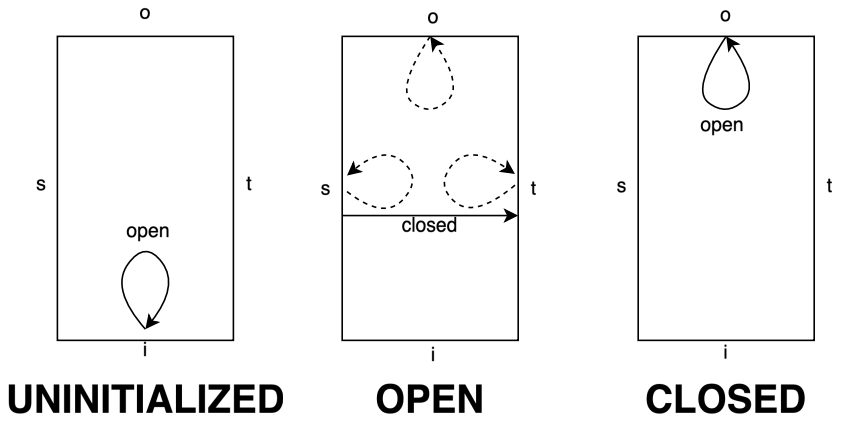
For any such planar motion planning problem with start S, target T, and some planar embedding G of directed self-closing doors and connections, we reduce to a problem using initialization-needed gadgets.

First, we add a new start point S' and an initialization path $S' \to S$ going through location s of every directed self-closing door and immediately exiting out of the corresponding location t of the self-closing door.

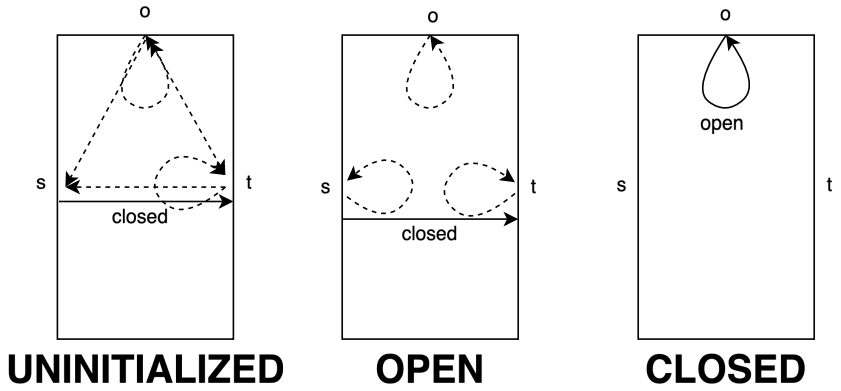
We then replace each crossing between the initialization path and the original problem G with an initialization-time crossover gadget and replace each directed self-closing door with an expanded door gadget, to be described below using shorthand from Figure 3.3.

The initialization-time crossover gadget has two single-use paths (namely, initialization-needed directed self-closing doors with no access to the o location) on the initialization path immediately before and after the crossing. There are also initialization-needed opening doors

[1] Note that in both self-closing door definitions we allow the possibilities of transitions that close the door from any location without going to another location. This changes nothing about the behavior because closed doors are strictly less useful than open doors, and therefore the robot would never try to close the door without going anywhere when solving reachability problems.

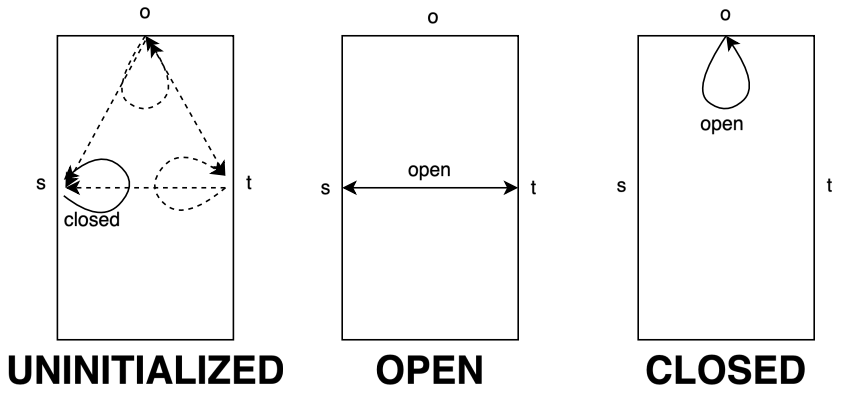


(a) An initialization-first directed self-closing door

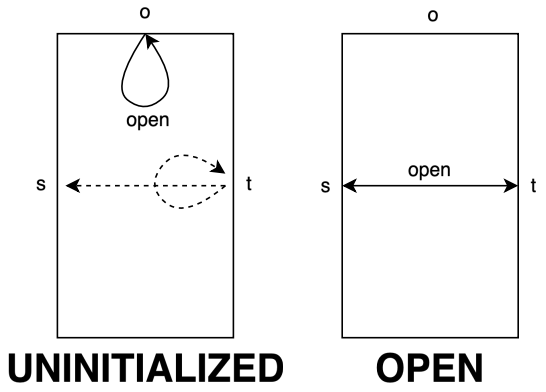


(b) An initialization-needed directed self-closing door

Figure 3.1: Gadget diagrams for initialization-needed self-closing doors. Note that the dashed arrows are transitions that may or may not exist, and may or may not send the gadget to other unspecified states with further transitions.

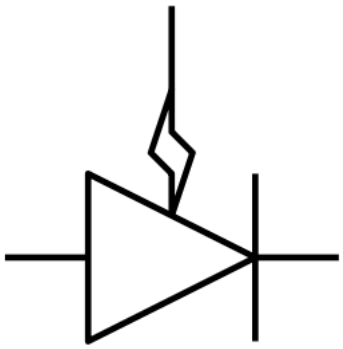


(a) A close-first initialization-needed opening door

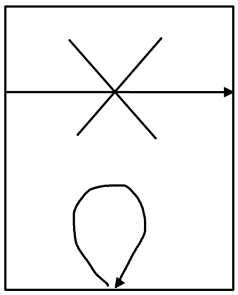


(b) A no-close initialization-needed opening door

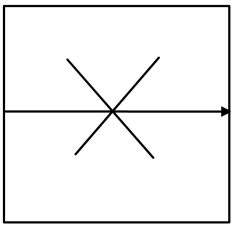
Figure 3.2: Gadget diagrams for initialization-needed opening doors. Note that the dashed arrows are transitions that may or may not exist, and may or may not send the gadget to other unspecified states with further transitions.



(a) Shorthand symbol for an initialization-needed opening door (s is on the right, t on the left)



(b) Shorthand symbol for an initialization-needed self-closing door (s is on the left, t on the right)



(c) Shorthand symbol for an initialization-needed directed single-use path (s is on the left, t on the right)

Figure 3.3: Shorthand for some initialization-needed gadgets

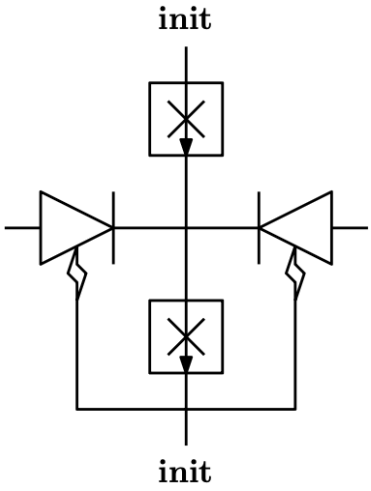
immediately before and after the crossing on the edge in G, with the gadgets oriented so that their s side is connected to the crossing point while their t side is not, and with their o side connected to the initialization path after the second single-use path. See Figure 3.4a for a diagram.

The expanded door gadget looks similar to the initialization-time crossover gadget, but with an initialization-needed self-closing door on the shared portion of the initialization path and the path from G instead of a single point of crossover. In particular, since there are external edges to most self-closing doors connected to the opening location o, there is an extra crossover needed between the initialization path and the edge to location o of the initialization needed opening path. In the case that the original self-closing door was meant to be open in G, we add a partial crossover gadget so as to allow the robot to toggle the initialization-needed directed self-closing door open between the two crossovers. See Figure 3.4b for a diagram.

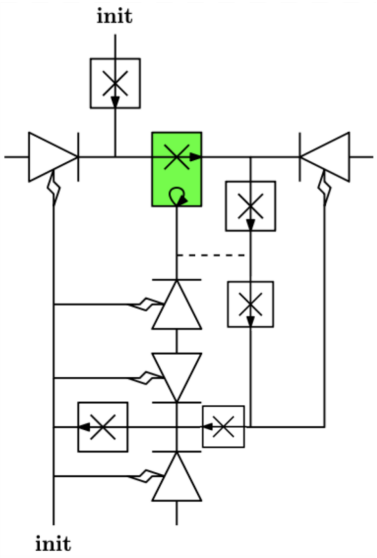
Our new problem is now to go from S' to T in this modified planar diagram G' where all gadgets start off in the uninitialized state.

Note that this reduction can be performed in polynomial time, using only polynomially more space, because G is polynomially sized and therefore we can find in polynomial time an appropriate $S' \to S$ path with polynomially many crossings with edges in G. We now prove that a robot can get from S to T in G if and only if a robot can get from S' to T in G'.

First we observe that a single-use path from s to t where location s is guaranteed to be visited before location t can be simulated with an initialization needed directed self-closing door simply by removing any access to location o: the only relevant transition remaining is the transition from s to t that permanently closes the gadget.



(a) Initialization time crossing.



(b) Initialization of the green self-closing door. The dashed line exists if the self-closing door should start in the open state.

Figure 3.4: Gadgets used for the initialization path

Next, note that the robot has no way to permanently leave the initialization path until it reaches S. Suppose for the sake of contradiction that the robot somehow leaves the initialization path before reaching S. The first such occurrence can only happen by the robot going through an initialization-needed opening door, or by the robot interacting somehow with the initialization-needed self-closing door. We consider the former case first. The robot's first access to each initialization-needed opening door is from the s side, which when the door starts uninitialized provides no way for the robot to leave location s. The robot is unable to reach any other location of the opening door until they go through a single-use path, after which the s location of the opening door becomes unreachable. After proceeding through this single-use path, the robot is only able to access the o location of the single-use path, which cannot reach either the s or t locations and therefore does not allow the robot to leave the initialization path. Thus the former case is impossible. Now consider the latter case. The first interaction of the robot with each initialization-needed self-closing door is to enter from s while uninitialized and to exit from t, which we know must put the door in the closed state. The only other possible interaction of the robot with each self-closing door (prior to reaching S) is with the opening location o, which by definition of initialization-needed self-closing doors is unable to access either s or t in either the open or closed states. Thus, the latter case is also impossible. Thus we see that the robot cannot leave the initialization path until reaching S.

Third, we show that we may assume without loss of generality that the robot's initial moves are to go from S' to S, and that once the robot reaches S every single-use path is blocked, every opening-door is open, and every initialization needed self-closing door of G' is open or closed depending on whether the corresponding self-closing door of G is initially

open or closed. Firstly, since the robot cannot leave the initialization path it may as well follow the path to the end and therefore reach S. Secondly, every single-use path is trivially blocked because the initialization path goes through all of them. Thirdly, every close-first initialization-needed opening door can be opened because we visit s first before visiting o for each opening door, which is sufficient to first initialize and close the door before opening it. Likewise, every no-close opening door can be opened because each o location of each opening door is visited. Furthermore, there is no reason not to open any opening door because opening the door allows for free traversal between s and t, which by construction is the maximum possible amount of freedom for traversing an opening door (location o is unreachable within the gadget from either s or t). Put another way, opening the door makes the reachability problem from S to T as easy as possible. As such, we may assume that every opening door is open. Finally, note that every self-closing door is traversed and therefore closed along the initialization path. Only those that are initially open in G are then given the option to be opened, so since opening a door always makes reachability no harder, we may assume without loss of generality that all doors are opened whenever possible. Thus, reachability in G' from S' to T is equivalent (after the robot moves from S' to S) to reachability from S to T in a variant of G' where all single-use paths are closed, all opening doors are open, and all self-closing doors are open or closed depending on their initial state in G.

But initialization-needed directed self-closing doors, once initialized, behave equivalently to standard directed self-closing doors. Thus reachability in G' under these conditions is exactly equivalent to reachability in G from S to T because all modifications to G have essentially been undone by our traversal through the initialization path: there is no longer any way to get onto the initialization path, all (partially) blocked edges of G are now once again unblocked, and the self-closing doors now operate properly. As such, the reduction preserves reachable instances and unreachable instances so the PSPACE-hardness of reachability with directed self-closing doors also proves PSPACE-hardness of reachability with initialization-needed directed self-closing doors. □

Remark 4. Note that by appropriately adjusting the definition of initialization-needed directed self-closing doors we can also produce initialization-needed forms of other self-closing doors defined by Ani et al. [3]. By essentially the same construction as in Theorem 3, we can prove that planar motion planning with either form of initialization-needed opening doors and any form of initialization-needed self-closing doors is also PSPACE-hard, although when open is mandatory there will be additional crossovers needed in order in the expanded door.

Corollary 5. *Planar motion planning with initialization-needed directed self-closing doors and either form of initialization-needed opening doors is PSPACE-hard.*

Proof. We can use the exact same construction, but with an additional visit to the i location on each initialization-first self-closing door before the initialization path visits the s location. We may need to add some additional crossovers in order to facilitate this visit. In particular, while it may be possible to reopen closed doors from location i, it is impossible to reach location i so that doesn't matter.

Alternatively, we can ignore the $s - t$ visit and corresponding crossovers when the door is allowed to start open. □

3.2 Land and Grills

In this section we prove that the Land and Grills case is PSPACE-hard. As a reminder, this is the ladder-less case with $s = 2$, $S_1 = \{\text{walls, grills}\}$, $S_2 = \{\text{empty, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$. This implies both the "Land, Grills, and Water" and "Land, Grills, and Walls" results from prior work.

Theorem 6. *Stephen's Sausage Roll is PSPACE-complete without ladders when $s = 2$, $S_1 = \{\text{walls, grills}\}$, $S_2 = \{\text{empty, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$.*

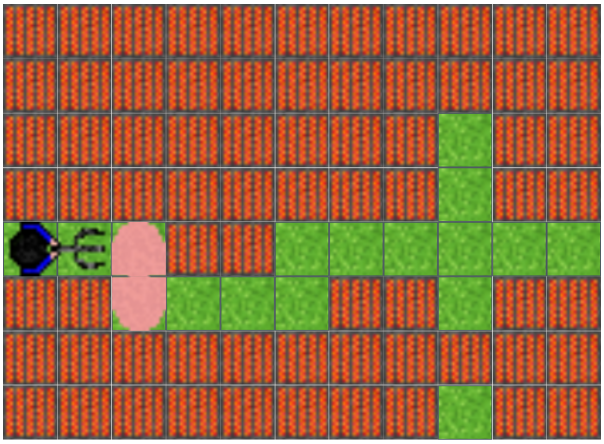
Proof. Note that from Theorem 2 we know Stephen's Sausage Roll is in PSPACE under these conditions. From Theorem 3 it suffices to construct a close-first initialization-needed opening door and an initialization-needed directed self-closing door. See Figure 3.5 for the opening doors and Figure 3.6 for the self-closing door. In these diagrams, cells with orange bars are grills one level below the player, green cells are walls one level below the player, and sausages are pink ovals that becomes brown when fully cooked. Note that these sausages are on the same level as the player, and the only things the player can walk on are green cells. We will justify that these constructions work as long as we ban all states that result in sausages being impossible to ever fully cook, since sausages that cannot be fully cooked will prevent victory.

We first show that the opening door construction satisfies Definition 3. Note that there is, indeed, no way to traverse from o to any other location, since there is no path of blocks available. Likewise, once the opening door is in the state corresponding to either of the lower two images, traversals in either direction between s and t are freely available: the sausage is completely nonblocking in Figure 3.5d and can be pushed up from either the left or the right in Figure 3.5c to reach the last state. Also, the path is fully blocked in Figure 3.5b because from the left and right the sausage can only be pushed down, left, or right, all of which would overcook the sausage. Also, starting from s in the uninitialized state, we can reach the closed state simply by pushing the sausage 6 times to the right.

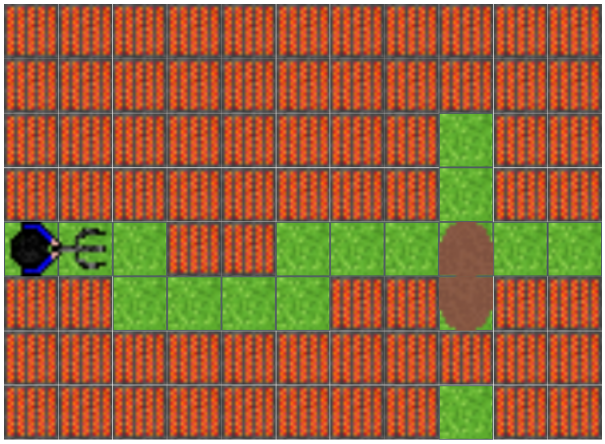
Finally, we justify that there is no transition from s in the uninitialized state to t without leaving the gadget. Entering the gadget from the left, we note that there is no way to push the sausage to the left (without leaving first) unless we push it up or down first. As such, any attempt to cross the gadget must start by pushing the sausage right some number of times and then pushing it either up or down. Pushing it down is possible only after $0, 1, 4, 5$, or 6 pushes to the right, and pushing it up is only possible after $1, 2, 3$, or 4 pushes to the right. We consider the cases one at a time.

- Pushing down after 0 pushes to the right: The sausage will no longer be able to be fully cooked, as there will never be any way to move the sausage back up (pushing it all the way to the right next to location o will at minimum overcook the lower half of the sausage), and so there is no other way to cook the upper half without cooking the lower half. As such, pushing down after 0 pushes to the right will create a state where the sausage can never be fully cooked, which is not allowed.

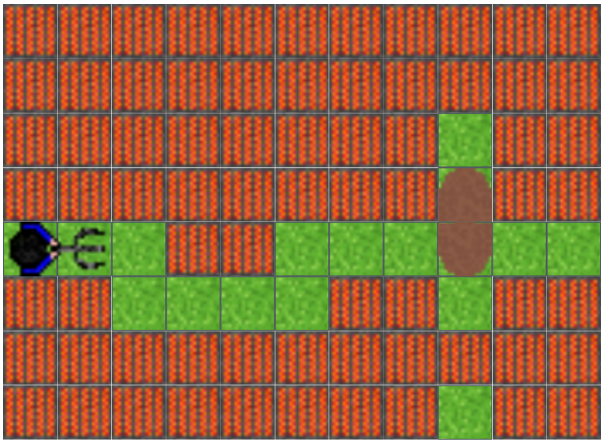
- Pushing down after 1 push to the right: The sausage is evenly cooked on its current bottom side, but the only way to move it at this point is to push it left or right in



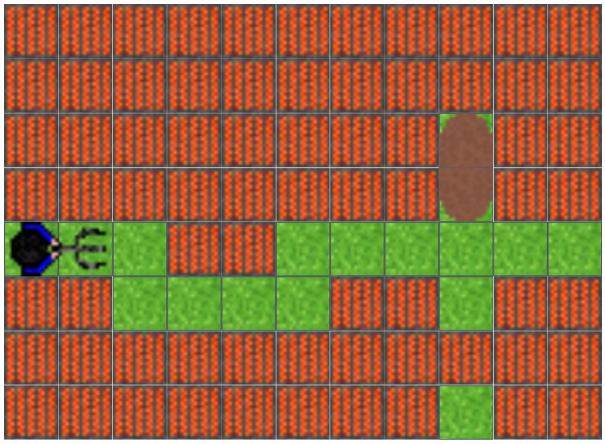
(a) An uninitialized opening door



(b) A closed opening door



(c) An opened opening door not yet traversed



(d) An opened opening door after traversal

Figure 3.5: Opening doors in a land and grills framework. Locations s, t, o are on the left, right, and bottom, respectively.

which case the lower half is once again more cooked than the upper half. There is no way to resolve this in this gadget without pushing the sausage back up, which again is impossible. As such, pushing down after 1 push to the right will create a state where the sausage can never be fully cooked, which is not allowed.

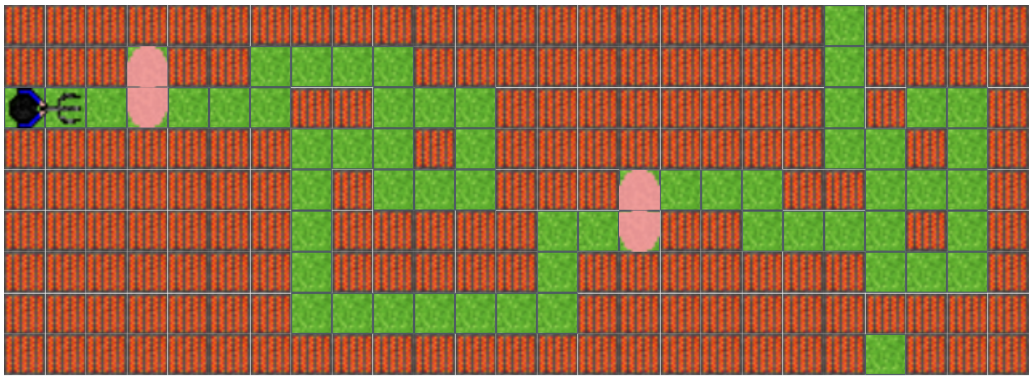
- Pushing down after $4, 5$, or 6 pushes to the right: This overcooks the sausage and is therefore not allowed.
- Pushing up after $1, 2, 3$, or 4 pushes to the right: This overcooks the sausage and is therefore not allowed.

As such, we see that the opening door construction satisfies Definition 3, with the slight difference that there are technically two open states that we merge together.

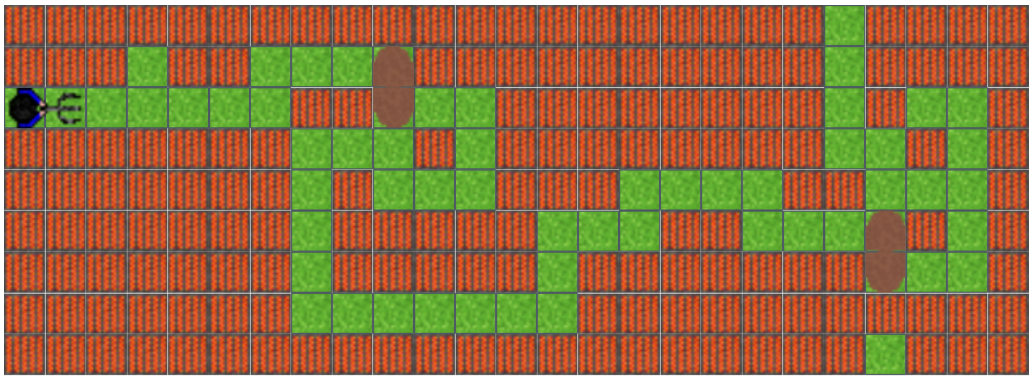
We now show that the self-closing door construction satisfies Definition 2. Note that the construction is essentially split into two portions: the left portion, which we call a *diode*, and the right portion, which we call the *gate*. We prove some properties of these gadgets separately.

First, note that the only way to traverse the diode when uninitialized from left to right must involve pushing the sausage either up or down after at most six pushes to the right (there is no way to push the sausage left until you push it up or down at least once). Due to the position of the path, pushing up is only possible after $0, 1, 2, 3$, or 4 pushes to the right, while pushing down is only possible after $1, 4, 5$, or 6 pushes. We consider the cases one at a time.

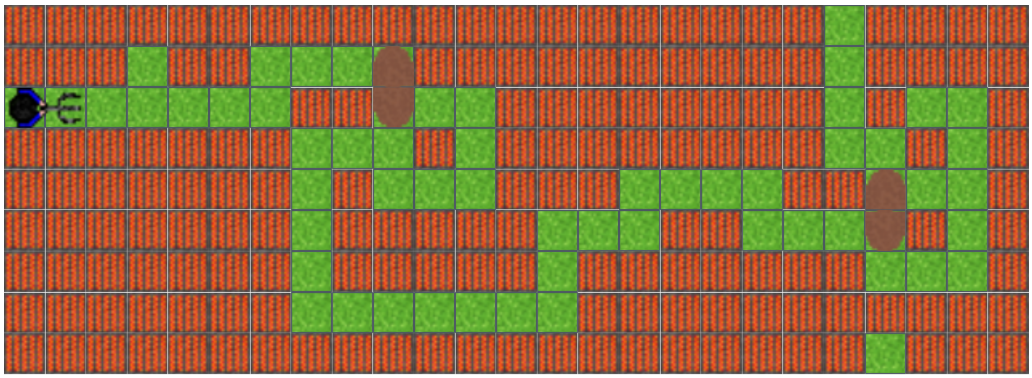
- Pushing up after 0 pushes to the right: The sausage will be more cooked in the upper than the lower half, which can only be corrected by pushing the sausage down at some point which is impossible. This sausage will therefore become impossible to properly cook, which is not allowed.
- Pushing up after $1, 2, 3$, or 4 pushes to the right: The sausage will be overcooked which is not allowed.
- Pushing down after 1 push to the right: The sausage will be unable to move up or down without overcooking, so traversing the diode will require the sausage to be pushed to the right again. Once that occurs, pushing down, left, or right would all overcook the sausage while pushing up is impossible since the fork cannot reach below the sausage, so we cannot traverse the diode. Thus pushing down after 1 push to the right does not work.
- Pushing down after 4 or 5 pushes to the right: The sausage will be overcooked which is not allowed.
- Pushing down after 6 pushes to the right: The sausage is now fully cooked and therefore constrained to lie within two of the four vertically-connected cells of the diode. Traversal is possible by pushing it two steps down, walking to the lower right of the diode, then pushing the sausage two steps up before exiting.



(a) An uninitialized self-closing door



(b) A closed self-closing door



(c) An open self-closing door

Figure 3.6: Self-closing doors in a land and grills framework. Locations s, t, o are on the left, top, and bottom, respectively.

Thus we see that an uninitialized diode can be traversed from left to right, but such traversal will leave the sausage in the top two cells of the four vertically-connected cells. Once the sausage is in that position, it can be pushed either one or two cells down from the left without traversing. Such diodes are considered to be *initialized*.

Suppose for the sake of contradiction that an initialized diode can be traversed from right to left. This must start by pushing the sausage to the upper two spots in order to reach the lower right of the diode, then by pushing the sausage to the lower two spots in order to exit on the left. However if the sausage is in the upper two spots then it is only reachable from the left, so this sequence of operations is impossible.

Thus a diode can be initialized from a left-to-right traversal, after which arbitrarily many left-to-right traversals are possible but right-to-left traversals are impossible.

Now consider the gate: suppose we try to traverse the gate from left to right starting from an uninitialized state. In order to do this, we must push the sausage either up or down after some number of pushes to the right, as we have no way to push it to the left. We can push it up after $0, 1, 4, 5$, or 6 pushes to the right, and we can push it down after $1, 2, 3$ or 4 pushes to the right. We consider cases one at a time.

- Pushing up after 0 pushes to the right: The upper half of the sausage will be more cooked than the lower half, which can only be resolved by pushing it back down at some point or by pushing it all the way to the right or left into either the gate or diode gadget. However it is impossible to reach the sausage from above and pushing the sausage into either gadget will overcook it and thus this sausage is uncookable, which is not allowed.

- Pushing up after 1 push to the right: The sausage is evenly cooked on its current bottom side, but the only way to move it at this point is to push it left or right in which case the lower half is once again more cooked than the upper half (pushing up overcooks the sausage). There is no way to resolve this without pushing the sausage back down, which as in the previous case is impossible. As such, pushing up after 1 push to the right will create a state where the sausage can never be fully cooked, which is not allowed.

- Pushing up after 4 or 5 pushes to the right: This overcooks the sausage and is therefore not allowed.

- Pushing up after 6 pushes to the right: The sausage is now fully cooked and trapped within the four vertically connected cells near the right of the gate. In particular it is in the top two cells, so the player can travel to the upper right of the gate and push the sausage down to the middle two cells. Then by travelling to the middle right of the gate the player can knock the sausage down another cell in order to exit the gadget, leaving the gate in the closed state as in Figure 3.6b.

- Pushing down after $1, 2, 3$, or 4 pushes to the right: This overcooks the sausage and is therefore not allowed.

Thus an uninitialized gate can be traversed from left to right, but such traversal will leave the gate in the closed state. Likewise an open gate can also be traversed from left to right, with essentially the same process but without the need to cook the sausage via 6 pushes to the right as that has already been done. Note that in the closed state, there is no way to traverse the gate from left to right because the player cannot reach below the sausage to push it up out of the way. There is also no way to open the gate from the right without traversing the gate from right to left because returning to the right/upper side of the gate will force the sausage to be pushed to the lower two cells in the process. Also note that the gate can be easily opened from the opening path simply by pushing the sausage up a cell. Finally, the opening path is evidently completely separated from any other location in the gadget.

Putting this information together we arrive at the following properties of a self-closing door. First, traversing from s to t in either an uninitialized or open door is possible and is guaranteed to leave the door in a closed state. Second, the door can be opened from the opening path. Therefore all the necessary transitions exist. Thirdly, location o is fully separated from s and t, and there is no way to transition starting from an open or closed door into a door that is neither open or closed: both sausages are trapped within 4 vertically-connected cells, and moving the diode up and down doesn't change its state (infinite traversal left-to-right is still possible while banning traversals right-to-left) and the gate portion is open in the upper two positions and closed in the bottom. There is no way to traverse from s to t in a closed door because a closed gate portion is untraversable from left to right, and there are never any traversals from t to s in an open or closed door because the diode will block the traversal. Finally there are no other transitions from closed to open: the gate can only be opened either from the opening path or by a right-to-left traversal, which would have to be immediately reversed because the diode blocks any further movement to the left. Therefore the combination of our diode and gate gives us a self-closing door.[2]

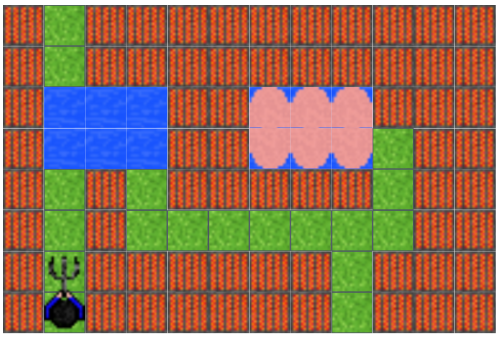
Since we have successfully constructed both an initialization-needed directed self-closing door and a close-first initialization-needed opening door, we know that Stephen's Sausage Roll is PSPACE-complete without ladders when $s = 2$, $S_1 = \{\text{walls, grills}\}$, $S_2 = \{\text{empty, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$. □

3.3 Underground sausages

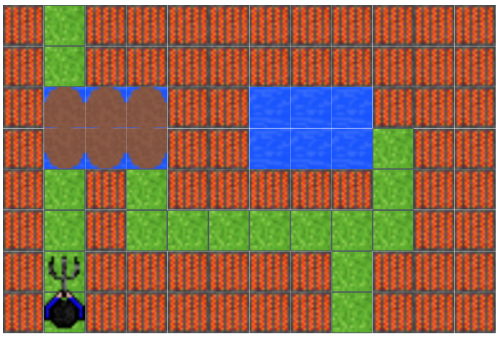
In this section we consider, essentially, the case where sausages are removed from the player's layer. Specifically, suppose that all sausages are in the layer below the player: cooking is still possible since sausages roll when the player walks over them. Alternatively, we can think of this problem as the land, grills, and walls case where the player is trapped in the layer above the grills.

Theorem 7. *Stephen's Sausage Roll is PSPACE-complete without ladders when* $s = 3$, $S_1 = \{\textit{walls, grills}\}$, $S_2 = \{\textit{empty, walls, sausages}\}$ *and* $S_i = \{\textit{empty}\}$ *for* $i > 2$.

[2]Technically we have multiple open states and multiple close states depending on the state of the diode. Note however that although the gate has multiple open states, the sausage cannot actually be in the topmost two cells when the player is outside of the gadget, because the sausage will only stay there if we somehow manage to reverse our way out of the diode.



(a) An uninitialized opening door



(b) An open opening door

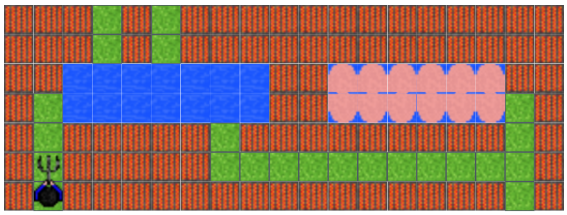
Figure 3.7: Opening doors in an underground framework. Locations s, t, o are on the bottom left left, top left, and bottom right respectively.

Proof. Note that from Theorem 2 we know Stephen's Sausage Roll is in PSPACE under these conditions. From Corollary 5 it suffices to construct an open-first initialization-needed opening door and an initialization-first directed self-closing door. See Figure 3.7 for the opening doors and Figure 3.8 for the self-closing door. In these diagrams, cells with orange bars are grills two levels below the player, blue cells are walls two levels below the player, green cells are walls one level below the player, and the sausages are pink ovals that becomes brown when fully cooked. Note that in this construction sausages are one level below the player, and the only things the player can move on are green cells and sausages. We will justify that these constructions work as long as we ban all states that result in sausages being impossible to ever fully cook, since sausages that cannot be fully cooked will prevent victory.

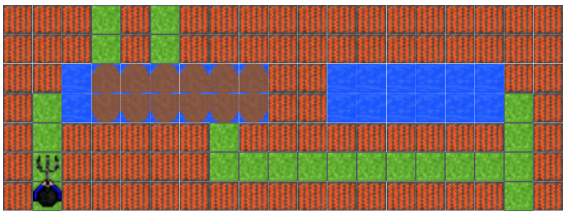
We first show that the opening door construction satisfies Definition 4. First we observe that the desired transitions exist: entering from o, moving onto the right-most sausage, and then rolling the sausage 5 times to the left will also push the other two sausages and thereby move the door into the open state, after which the player can exit again from o by walking down from the right-most sausage to exit. When open, the door can be traversed simply by walking across the left-most sausage. Also note that no other transitions exist: an uninitialized door cannot be affected from the s or t locations because those locations cannot reach any sausages or any other locations in an uninitialized state. Furthermore, from an uninitialized state the only way to change the state of the gadget is to walk onto the right-most sausage from o. At this point, there is no longer any way to move onto the right-most path because you can only step off a sausage perpendicular to its axis of rotation if the sausage is unable to roll, so you must roll the sausages five steps to the left before exiting downwards back to o. Also, in the open state no sausages are able to move without overcooking and we cannot step from one sausage to another because they are all able to roll and thus there are no other transitions starting from an open state.

As such, we see that the opening door construction satisfies Definition 4 and in fact does so with the minimal number of transitions and states.

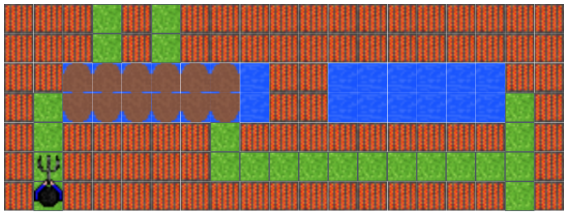
We now show that the self-closing door construction satisfies Definition 1. First note that the needed transitions do exist: an uninitialized gadget in state i will become open if the player steps onto the left-most sausage and rolls all of them left 9 times before stepping



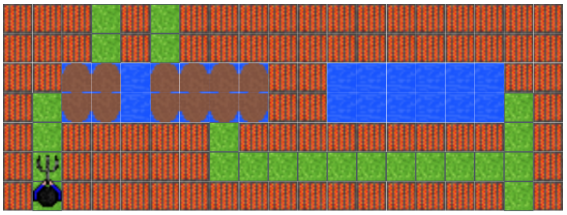
(a) An uninitialized self-closing door



(b) A closed self-closing door



(c) An open self-closing door prior to first traversal



(d) An open self-closing door after first traversal

Figure 3.8: Self-closing doors in an underground framework. Locations s, t, i, o are on the bottom left, top left, bottom right, and top right, respectively.

of the lower half to exit from i. An open door can be traversed from s by stepping onto the leftmost sausage, rolling right, and exiting from above, leaving the door in the closed state. Finally, after entering a closed door from o we can step onto the third sausage, roll left and then right, and step back up to leave the gadget in the state in Figure 3.8d.

We now show that no banned transitions exist: First, note that an uninitialized door cannot have any transitions starting from s, t, or o because there is nothing reachable from those locations. The only transition starting from i will be to step onto the rightmost sausage, after which the sausage can only be exited by rolling left 9 times and stepping down, so there is a unique transition that starts from an uninitialized state. Once all sausages are in the left blue portion, they can never leave that portion of the gadget, so the remaining possible states correspond to knowing which column is empty. Furthermore each column can only be made empty by moving a sausage out of that column and immediately stepping off said sausage, so the only possible empty columns are columns $1, 3, 5$, and 7. An empty column 1 is the closed state, in which case there is no way to go from s to t because there is nothing reachable from t. An empty column $3, 5$, or 7 leads to an open state, so there are no transitions starting from open or closed that end in a state that is neither open nor closed. In all states there is no way to go from t to s because there is no way to step left onto the green path. There is also no way to go from i to any other location or from o to any other location because the constrained space means that sausages can roll at most one cells to the left or right, and both i and o are two cells (horizontally) away from any other path. Finally, note that a closed door also cannot be opened from t because the only blank column is column 1, and so the only possible actions starting from t would be to step onto the sausage, roll the sausage to the left, roll back to the right because the sausage cannot be exited, and end up in the closed state again.

Thus, we see that the self-closing door construction satisfies Definition 1, albeit with multiple possible open states.

Since we have successfully constructed both an initialization-first directed self-closing door and a no-close initialization-needed opening door, we know that Stephen's Sausage Roll is PSPACE-complete without ladders when $s = 2$, $S_1 = \{\text{walls, grills}\}$, $S_2 = \{\text{empty, sausages}\}$ and $S_i = \{\text{empty}\}$ for $i > 2$. □

3.4 Cases without ladders

In this section we give a brief categorization of cases when ladders are banned, depending on the contents of S_{s-2}, S_{s-1}, S_s, and S_{s+1}. Since each S has at most 4 elements, there are 15^4 possibilities as no set can be empty. This is far too many for a table, but computer checks suggest that over 90% of cases are covered.

3.4.1 Solvable in P

Lemma 8. *In the following cases, Stephen's Sausage Roll is in P.*

1. *If S_{s-1} contains neither wall or sausage then Stephen's Sausage Roll is in P.*
2. *If S_s doesn't contain empty then Stephen's Sausage Roll is in P.*
3. *If sausage is not an element of $S_{s-1} \cup S_s \cup S_{s+1}$ and there are no ladders then determining solvability in Stephen's Sausage Roll is in P.*
4. *If the sets S_s, S_{s+1} do not contain sausage and S_{s-1} doesn't contain empty, then Stephen's Sausage Roll is in P.*
5. *If grill is not an element of $S_{s-1} \cup S_{s-2}$ and there are no ladders then determining solvability in Stephen's Sausage Roll is in P.*
6. *If grill and empty are not elements of S_{s-1} and there are no ladders then determining solvability in Stephen's Sausage Roll is in P.*
7. *If grill is not an element of S_{s-2} and sausage is an element of S_{s-1} but not S_s or S_{s+1} and there are no ladders then determining solvability in Stephen's Sausage Roll is in P.*
8. *If grill is not an element of S_{s-2} and sausage is an element of S_{s-1} but wall is not, and there are no ladders then determining solvability in Stephen's Sausage Roll is in P.*
9. *If sausage is not an element of $S_{s-1} \cup S_s$ and walls and grills are both not elements of S_{s+1} and there are no ladders then determining solvability in Stephen's Sausage Roll is in P.*

Proof. 1. In this case there are no valid levels as there is nothing for the player to stand on. As such this lemma is vacuously true.

2. In this case the player cannot move (and arguably doesn't even have any place to spawn in) and therefore the level is unsolvable if there exist any sausages.

3. If sets S_{s-1}, S_s, S_{s+1} all have no sausages, then there are no sausages that can interact with the player, as there are no sausages they can stand on or push with a fork, and no sausages on top of their head that they can knock off by moving. As such, if there are any sausages at all then the level cannot be completed and otherwise the level is trivially solved. As such, solvability of Stephen's Sausage Roll in this case is equivalent to lack of existence of sausages, which can be checked in P.

4. The only directly accessible sausages must always be in layers $s-1, s$, or $s+1$, of which only $s-1$ has sausages. Such sausages cannot move and therefore cannot be cooked. Thus solvability of Stephen's Sausage Roll in this case is equivalent to lack of existence of sausages, which can be checked in P.

5. First note that since there are no ladders, there is no way to remove sausages at or above layer $s+1$ that are resting on one or more walls or grills. As such, any cooking must be performed when the sausage is at or below level s, using grills at or below level $s-1$, in order to fully cook every sausage.

   If sets S_{s-2} and S_{s-1} have no grills, then all cooking must occur when the sausage is at least two layers below the player, after which sausages cannot go up any layers. Thus the last sausage to be cooked will be impossible to flip, as by that point all sausages will be at or below level $s-2$ and thus inaccessible to the player. Thus if any sausages exist, then the level is unsolvable.

   As such, solvability of Stephen's Sausage Roll in this case is equivalent to lack of existence of sausages, which can be checked in P.

6. The last sausage in layers $s-1, s$, or $s+1$ must always be cooked on a grill in either layer $s-2$ or $s-1$, all of which are blocked since empty and grills do not exist in the layer $s-1$. Thus solvability of Stephen's Sausage Roll in this case is equivalent to lack of existence of sausages, which can be checked in P.

7. In this case the only directly accessible sausages are in layer $s-1$ or lower and can only be cooked once they drop to layer $s-2$ or lower, so analogous to the grill-less case the last sausage of layer $s-1$ will be uncookable. If there are no sausages in layer $s-1$ then no other sausages are reachable, so solvability of Stephen's Sausage Roll in this case is equivalent to lack of existence of sausages, which can be checked in P.

8. In this case the player must start off standing on a sausage that is in layer $s-1$. The same logic as the previous part implies that the last sausage in this layer cannot be cooked.

9. In this case the only directly accessible sausages are those in layer $s+1$ that are only accessible if they are on your head. These sausages cannot be knocked off of your head because there is nothing to knock them against.

□

3.4.2 PSPACE-hard cases

- If S_{s-1} contains both grills and walls and S_s contains both empty and sausages then by Theorem 6 Stephen's Sausage Roll is PSPACE-hard.[3]
- If layer S_{s-2} contains both grills and walls, S_{s-1} contains walls, sausages, and empty, and S_s contains empty, then by Theorem 7 Stephen's Sausage Roll is PSPACE-hard.[3] This covers 2^{10} cases.

3.4.3 Unsolved

We list some notable unsolvable cases that are tricky to consider and some comments as to why.

- $S_{s-2} = \{\text{grill, wall, empty, sausage}\}, S_{s-1} = \{\text{grill, sausage, empty}\}, S_s \supset \{\text{empty}\}$. In this case we can only stand on sausages, which makes path construction tricky. In particular it is possible that solutions involve intentionally dropping the levels of sausages since the player only needs to be at or above the level of the (possibly dropped) initial sausage they stood on.
- $S_{s-2} = \{\text{empty, sausage, grill}\}, S_{s-1} = \{\text{grill, sausage, wall, empty}\}$, and $S_s = S_{s+1} = \{\text{empty}\}$. In this case the sausages all start off resting on other sausages. While it is possible to roll them off of the sausages underneath onto grills for cooking, gadget initialization and design still becomes much more difficult. In particular the lower sausages can only be cooked by vertical stacks of sausages, which when perfectly aligned are completely irreversible.
- $S_{s-2} = \{\text{empty, grill, wall}\}, S_{s-1} = \{\text{wall, empty}\}, S_s = \{\text{empty, sausage}\}$. While intuitively it seems simple to push sausages off the player's level onto the level below into the gadgets introduced in Theorem 7, it seems highly nontrivial to prevent leakage in the gadgets: if only some sausages are pushed into the level below, it may be possible to escape the initialization path early and potentially return to fix the initialization issues afterwards.
- Sausage is not in either S_{s-1} or S_s. In this case many sausages are balanced on top of the player, with each sausage at least half-on either some sausage or the player but with no sausage resting on any grill or wall. These sausages can be knocked off the player's head, but it is unclear what constraints can be placed on the landing positions of these sausages.

3.5 Ladders

Adding ladders to PSPACE-hard cases does not affect PSPACE-hardness, and indeed allows for the player s to start at a level other than the ones immediately nearby the sausages. However, many of the P cases are no longer trivial, and we encounter the possibility of

[3]Note that we do allow the possibility of additional options in each layer in this case.

multi-level gadgets or even three-dimensional gadgets, the exploration of which we leave to future researchers.

Chapter 4

Conclusion

We have shown that two variants of Stephen's Sausage Roll are PSPACE-complete in Sections 3.2 and 3.3. Furthermore, we have shown that initialization-needed gadgets can be used to perform planar motion planning with various kinds of initialization-needed self-closing doors and opening doors.

Nonetheless, there is still a significant number of cases open for future research. Such cases include the ones listed in Section 3.4.3, as well as most cases that involve ladders. Other possibilities include finding difficulty bounds when given only a few sausages, and an analysis of any changes in difficulty (if any) introduced by the addition of movable objects such as the serpent and tree-island mentioned in Section 2.2. A less interesting but still useful problem would be to adapt the solver by Chengyuan Ma to work when sausages may be left partially cooked during a traversal to be fully cooked by later traversals [15].

References

[1] MIT Hardness Group, J. Liu, N. Lu, C. Ma, R. Sollee, and K. Zhang, *Stephen's Sausage Roll is PSPACE-complete*, Unpublished, 2023.

[2] R. A. Hearn, "Amazons, Konane, and Cross Purposes are PSPACE-complete," in *Games of No Chance 3* (Mathematical Sciences Research Institute Publications), M. H. Albert and R. J. Nowakowski, Eds., Mathematical Sciences Research Institute Publications. Cambridge University Press, 2009, pp. 287–306. DOI: 10.1017/CBO9780511807251.015.

[3] J. Ani, J. Bosboom, E. D. Demaine, J. Diomidova, D. Hendrickson, and J. Lynch, "Walking Through Doors Is Hard, Even Without Staircases: Proving PSPACE-Hardness via Planar Assemblies of Door Gadgets," in *10th International Conference on Fun with Algorithms (FUN 2021)*, M. Farach-Colton, G. Prencipe, and R. Uehara, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 157, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 3:1–3:23, ISBN: 978-3-95977-145-0. DOI: 10.4230/LIPIcs.FUN.2021.3.

[4] L. Chung and E. D. Demaine, "Celeste is pspace-hard: Discrete and computational geometry, graphs, and games," *Thai Journal of Mathematics*, vol. 21, no. 4, pp. 671–686, Dec. 2023. URL: https://thaijmath2.in.cmu.ac.th/index.php/thaijmath/article/view/1538.

[5] Z. Abel and D. Hendrickson, "Baba is Universal," in *Proceedings of the 12th International Conference on Fun with Algorithms*, 2024.

[6] S. Lavelle, *Stephen's sausage roll*, https://www.stephenssausageroll.com/about.html, Accessed: 2023-12-13.

[7] W. J. Savitch, "Relationships between nondeterministic and deterministic tape complexities," *Journal of Computer and System Sciences*, vol. 4, no. 2, pp. 177–192, 1970, ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(70)80006-X.

[8] G. Viglietta, "Gaming is a hard job, but someone has to do it!" *CoRR*, vol. abs/1201.4995, 2012. arXiv: 1201.4995. URL: http://arxiv.org/abs/1201.4995.

[9] E. D. Demaine, D. Hendrickson, and J. Lynch, "Toward a General Complexity Theory of Motion Planning: Characterizing Which Gadgets Make Games Hard," in *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, T. Vidick, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 151, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 62:1–62:42, ISBN: 978-3-95977-134-4. DOI: 10.4230/LIPIcs.ITCS.2020.62.

[10] G. Aloupis, E. D. Demaine, A. Guo, and G. Viglietta, "Classic Nintendo games are (computationally) hard," *Theoretical Computer Science*, vol. 586, pp. 135–160, 2015.

[11] Steam, *Stephen's Sausage Roll on Steam*, https://store.steampowered.com/app/353540/Stephens_Sausage_Roll, Accessed: 2024-5-15.

[12] J. Ani, L. Chung, E. D. Demaine, J. Diomidova, D. Hendrickson, and J. Lynch, "Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude," in *Proceedings of the 11th International Conference on Fun with Algorithms (FUN 2022)*, Favignana, Italy, May 2022, 2:1–2:30.

[13] Plant God, *SSR Spoiler-Free Hint Guide*, https://steamcommunity.com/sharedfiles/filedetails/?id=2019667067, Accessed: 2023-11-27.

[14] au voleur! *Stephen's Sausage Roll (Demake) by au voleur!* https://au-voleur.itch.io/stephens-sausage-roll-demake, Accessed: 2023-11-27.

[15] C. Ma, *Stephen's Sausage Roll - Level Editor*, https://danglingpointer.fun/ssr/, Accessed: 2023-12-13.