On the Parameterized Complexity of Finding Short Winning Strategies in Combinatorial Games

by

Allan Scott
B.Sc., University of Victoria, 2002
M.Sc., University of Victoria, 2004

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

© Allan Scott, 2009
University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by photocopying
or other means, without the permission of the author.

On the Parameterized Complexity of Finding Short Winning Strategies in Combinatorial Games

by

Allan Scott
B.Sc., University of Victoria, 2002
M.Sc., University of Victoria, 2004

Supervisory Committee

Dr. Ulrike Stege, Supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Departmental Member
(Department of Computer Science)

Dr. John Ellis, Departmental Member
(Department of Computer Science)

Dr. Gary MacGillivray, Outside Member
(Department of Mathematics)

Supervisory Committee

Dr. Ulrike Stege, Supervisor
(Department of Computer Science)

Dr. Venkatesh Srinivasan, Departmental Member
(Department of Computer Science)

Dr. John Ellis, Departmental Member
(Department of Computer Science)

Dr. Gary MacGillivray, Outside Member
(Department of Mathematics)

ABSTRACT

A combinatorial game is a game in which all players have perfect information and there is no element of chance; some well-known examples include othello, checkers, and chess. When people play combinatorial games they develop strategies, which can be viewed as a function which takes as input a game position and returns a move to make from that position. A strategy is winning if it guarantees the player victory despite whatever legal moves any opponent may make in response. The classical complexity of deciding whether a winning strategy exists for a given position in some combinatorial game has been well-studied both in general and for many specific combinatorial games. The vast majority of these problems are, depending on the specific properties of the game or class of games being studied, complete for either PSPACE or EXP.

In the parameterized complexity setting, Downey and Fellows initiated a study of "short" (or k-move) winning strategy problems. This can be seen as a generalization of "mate-in-k" chess problems, in which the goal is to find a strategy which checkmates your opponent within k moves regardless of how he responds. In their monograph on parameterized complexity, Downey and Fellows suggested that AW[*] was the "natural home" of short winning strategy problems, but there has been little work in this field since then.

In this thesis, we study the parameterized complexity of finding short winning strategies in combinatorial games. We consider both the general and several specific cases. In the general case we show that many short games are as hard classically as their original variants, and that finding a short winning strategy is hard for AW[P] when the rules are implemented as succinct circuits. For specific short games, we show that endgame problems for checkers and othello are in FPT, that alternating hitting set, hex, and the non-endgame problem for othello are in AW[*], and that short chess is AW[*]-complete. We also consider pursuit-evasion parameterized by the number of cops. We show that two variants of pursuit-evasion are AW[*]-hard, and that the short versions of these problems are AW[*]-complete.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents v

List of Tables x

List of Figures xi

Acknowledgements xiv

1 Introduction 1

2 Complexity Theory Primer 7

2.1 Classical Complexity . . . 7

2.1.1 Known Classes and Containments . . . 8

2.1.2 Hardness, Completeness, and Reductions . . . 10

2.2 Parameterized Complexity . . . 12

2.2.1 Fixed-Parameter Tractability . . . 13

2.2.2 Parameterized Intractability . . . 13

2.3 Summary . . . 20

3 Combinatorial Games 21

3.1 Introduction . . . 21

3.1.1 Positions, Moves, and Players . . . 22

3.1.2 Games, Graphs, and Strategies . . . 24

3.2 The Minimax Algorithm . . . 27

3.2.1 Dealing with Cycles . . . 28

3.2.2 Minimax and Practical Use . . . 29

3.3 Solving Brainstones 30
3.4 Other Solved Games 31
3.5 Summary 33

4 Classical Complexity of Games 34
4.1 Type 1 Games – Polynomial Graph Traversal 35
4.2 Type 3 Games – Unbounded 39
4.3 Type 2 Games – Polynomial Termination 43
4.4 Predictable Opponent 47
4.5 Complexity of Specific Games 50
4.6 Summary 52

5 Short Games 54
5.1 Short Succinct Winning Strategy 55
5.2 Techniques for Proving Complexity of Short Games 58
5.2.1 Classical Hardness for Short Games 59
5.2.2 Reduction into AW[*] 59
5.2.3 FPT by Bounded Game Graph 60
5.3 Short Alternating Hitting Set 61
5.4 Hex 64
5.5 Geography 67
5.6 Checkers 68
5.7 Othello 68
5.7.1 Rules of Othello 69
5.7.2 Endgame Othello 69
5.7.3 Short Generalized Othello 70
5.7.4 Variables 71
5.7.5 Formula F 77
5.8 Summary 84

6 Short Chess 86
6.1 The Game of Short Generalized Chess 86
6.2 Parameterized Membership of Short Generalized Chess 88
6.2.1 Encoding Positions 89
6.2.2 The Winning Condition 91
6.2.3 Formula F 92

6.2.4 Testing for Broken Rules . . . 92
6.2.5 Correctness of the Reduction . . . 99
6.3 Hardness of Short Generalized Chess . . . 100
6.4 Summary . . . 115

7 Parameterized Pursuit **117**
7.1 Pursuit-Evasion Background . . . 117
7.2 Hardness of Seeded Pursuit-Evasion . . . 119
7.2.1 Reduction . . . 120
7.2.2 Sequence of Play . . . 123
7.2.3 Correctness . . . 129
7.3 Pursuit in a Hypercube . . . 129
7.4 Short Seeded Pursuit-Evasion . . . 131
7.4.1 Hardness of Short Seeded Pursuit-Evasion . . . 132
7.4.2 Membership of Short Seeded Pursuit-Evasion . . . 132
7.5 Directed Pursuit-Evasion . . . 137
7.5.1 Short Directed Pursuit-Evasion . . . 141
7.6 Summary . . . 142

8 Conclusions **144**
8.1 Contributions . . . 144
8.2 Open Problems . . . 145
8.2.1 Brainstones . . . 146
8.2.2 Short Games . . . 146
8.2.3 Is AW[*] the Natural Home of Short Games? . . . 148

Bibliography **151**

A Problem Definitions **158**
A.1 Alternating Reachability . . . 158
A.2 r-Alternating Weighted CNF Satisfiability . . . 159
A.3 r-Alternating Weighted t-Normalized Satisfiability . . . 159
A.4 Circuit Value . . . 159
A.5 Clique . . . 160
A.6 Directed Pursuit-Evasion . . . 160
A.7 Directed Reachability . . . 161

A.8 Dominating Set 161
A.9 Endgame Generalized Checkers 161
A.10 Endgame Generalized Othello 162
A.11 Game 162
A.12 Generalized Geography 162
A.13 Generalized Othello 163
A.14 Hitting Set 163
A.15 Independent Set 163
A.16 Parameterized Quantified Boolean Formula Satisfiability 164
A.17 Parameterized Quantified Boolean t-Normalized Formula Satisfiability 164
A.18 Parameterized Quantified Circuit Satisfiability 165
A.19 Quantified Boolean Formula 165
A.20 Restricted Alternating Hitting Set 165
A.21 Satisfiability 166
A.22 Seeded Pursuit-Evasion 166
A.23 Short Alternating Hitting Set 167
A.24 Short Directed Pursuit-Evasion 168
A.25 Short Generalized Checkers 169
A.26 Short Generalized Chess 169
A.27 Short Generalized Geography 169
A.28 Short Generalized Hex 170
A.29 Short Generalized Othello 170
A.30 Short Directed Pursuit-Evasion 170
A.31 Short Seeded Pursuit-Evasion 171
A.32 Short Succinct Winning Strategy 171
A.33 Succinct Alternating Reachability 172
A.34 Succinct Circuit Value 172
A.35 Succinct Directed Reachability 173
A.36 Succinct Winning Strategy 173
A.37 Traveling Salesman Problem 173
A.38 Undirected Reachability 174
A.39 Unitary Monotone Parameterized QBFSAT_2 174
A.40 Vertex Cover 174
A.41 Weighted Circuit Satisfiability 175

A.42 Weighted t-Normalized Satisfiability 175
A.43 Weighted Satisfiability 175
A.44 Winning Strategy 176

B Games **177**
B.1 Tic-Tac-Toe 177
B.2 Go-moku 177
B.3 Connect Four 178
B.4 Geography 178
B.5 Othello 178
B.6 Hex 179
B.7 Checkers 180
B.8 Chess 180
B.9 Go 182

C Brainstones **184**

List of Tables

Table 2.1 Definitions for common classical complexity classes 9
Table 2.2 Complete Problems for Various Complexity Classes 12

Table 3.1 Upper bounds on the sizes of game graphs for some well-known games. 32

Table 4.1 Resource bounds for determining the existence of winning strategies in various types of games [73]. 35
Table 4.2 Existing Complexity Results for Generalized Games 52

Table 7.1 New Complexity Results for Pursuit Games 142

Table 8.1 New results presented for specific problems in this thesis. 145
Table 8.2 Open Problems 146

List of Figures

Figure 1.1 A game board for tic-tac-toe. 1
Figure 1.2 Every tic-tac-toe opening where X plays on a side can be obtained as a rotation of the opening where X plays on the left side. 2

Figure 2.1 The A-matrix. 15
Figure 2.2 The upper reaches of the parameterized complexity hierarchy. . 19

Figure 3.1 A partial game graph for tic-tac-toe. 25

Figure 6.1 An overview of the chessboard as built by the reduction. 102
Figure 6.2 Deadlocked pawns. 103
Figure 6.3 Deadlocked pawns laid out in a checkerboard arrangement. . . . 103
Figure 6.4 Pawns deadlocked in a checkerboard arrangement with a column segment removed. 104
Figure 6.5 Pawns deadlocked in a checkerboard arrangement with a diagonal removed. 105
Figure 6.6 Pawns deadlocked in a checkerboard arrangement with battlement cuts. 106
Figure 6.7 The layout of the assignment gadget. This shows only two segments. Additional segments are added to the end as necessary. 107
Figure 6.8 Interaction in the variable assignment gadget. Black has responded to white's attack on the column. Arrows indicate possible moves by black to capture the queen if white attacks the gadget. 108
Figure 6.9 Black has moved the bishop out of the queen's path and set the value of an existential variable in the process. 109

Figure 6.10 A gadget to allow black to block the alternate capture column. Once the rook is moved next to the two black pawns, the diagonal is blocked and all the black pieces blocking it are protected. . . 110
Figure 6.11 Interaction in the one-way gadget. The queen is about to move in and take the black pawn. Black responds by taking the white pawn below with the rook, guarding the capture column. 111
Figure 6.12 The parallelogram gadget. 112
Figure 6.13 Parallelograms to leave the bishop vulnerable to the queen. . . 113
Figure 6.14 Parallelograms to protect the bishop. 114
Figure 6.15 The king's position in the king's diagonal. Note that a white piece can attack any square the king can move to. 115

Figure 7.1 Runway gadget construction for the ith quantifier. The cop (C) is shown in his starting position. Note that there is a one-to-one correspondence between the forks and the variables of S_i. . . . 122
Figure 7.2 An example of the assignment gadget for $r = 5$. The cop (C) and robber (R) are shown in their starting positions. 123
Figure 7.3 A Seeded Pursuit-Evasion instance constructed by our reduction from the Unitary Monotone Parameterized QBF-SAT$_2$ instance $S_1 = \{a, b, c\}, S_2 = \{d, e, f\}, F = (a \wedge \overline{d}) \vee (b \wedge \overline{f}) \vee (c \wedge \overline{e})$. The cops (C) and robber (R) are shown in their starting positions. 124
Figure 7.4 Distances from the clause vertices in the assignment gadget and runways. 126
Figure 7.5 Connections in G'. Solid arrows indicate a directed one-to-one mapping from the vertices in the origin set to the vertices in the destination set. Dashed arrows indicate that every vertex in the origin set has an edge to every vertex in the destination set. Dotted arrows indicate that each vertex in the origin set has one edge to each escape subgraph in the destination set. Arrows that alternate between dashes and dots indicate that each vertex in the origin set has an edge to every vertex of its unique escape subgraph in the destination set. 140

Figure 8.1 It is unclear which of the available paths the white checker took to reach the question mark. 147

Figure B.1 The starting position for othello. 179
Figure B.2 An 11x11 hex board. 180
Figure B.3 The starting position for checkers. 181
Figure B.4 The starting position for chess. 182

ACKNOWLEDGEMENTS

I would like to thank:

Ulrike Stege, for instruction and patience throughout, in addition to the many other supervisory things, and especially for the fun with algorithms and complexity theory that provided the motivation to start this whole process in the first place.

My parents, for immense encouragement and support. The ridiculous pile of left-over turkey, too.

Chapter 1

Introduction

We begin by asking a seemingly basic question: how difficult is it to play a game?

One can argue that if you were to analyze every single possible position in a game, then you could determine whether there is some way to win that game with absolute certainty. In this sense, the number of positions attainable in a game gives an upper bound on the difficulty of that game.

That said, one might wonder whether it is possible to cut out some of the cumbersome work of checking every position. For example, a tic-tac-toe board (Figure 1.1 – the dashed lines indicate the borders of the outer squares) is filled with readily exploited symmetries. While it may seem as though there are nine opening moves, you need analyze only three: the center, a corner, or a side. Given those three, every other opening move is simply a rotation of the board (Figure 1.2). One can also simplify the second move by using both rotations and reflections, and so on.

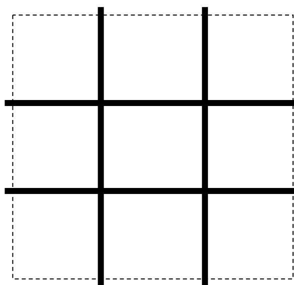

Figure 1.1: A game board for tic-tac-toe.

However, you can reduce the amount of work necessary even further if you are player two. You can rule out the possibility of finding a strategy for guaranteed

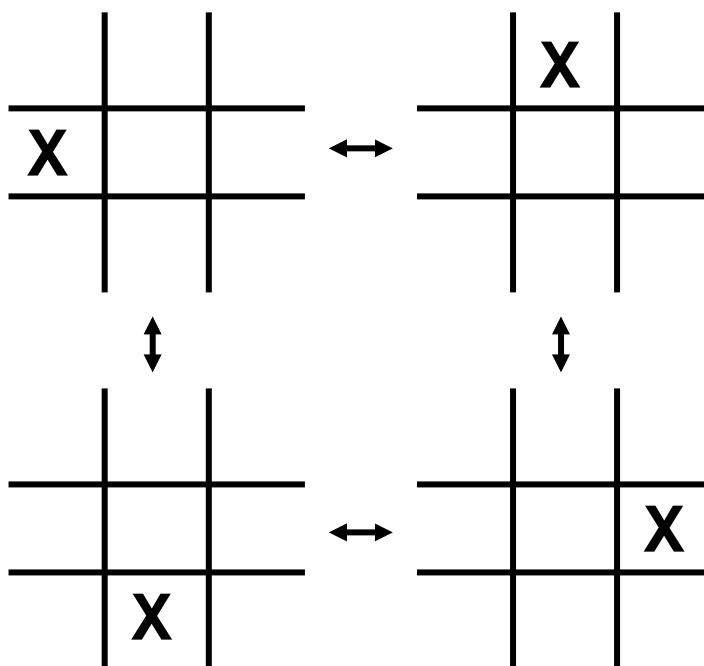

Figure 1.2: Every tic-tac-toe opening where X plays on a side can be obtained as a rotation of the opening where X plays on the left side.

victory because tic-tac-toe admits what is known as a *strategy-stealing argument*. In a strategy-stealing argument, if player two has a strategy which guarantees victory then player one can "steal" it by making a meaningless move on his first turn and then playing as if he were player two for the rest of the game. Player one is able to do this because both players' winning conditions are symmetric and having taken an extra square at the beginning doesn't hurt player one – if his strategy would ever compel him to play in the square he took at the beginning of the game, he already has it and can throw away the turn by moving randomly. Thus, in tic-tac-toe, if a strategy which guarantees victory exists for player two then such a strategy must also exist for player one. However, both existing at the same time is a contradiction since only one player can win a given game, and thus there is no winning strategy for player two. On the flip side, player one knows that if he plays perfectly he is guaranteed at least a draw – though this argument doesn't tell him what moves he would actually have to make to achieve this.

This strategy-stealing argument is a far cry from our initial measure of hardness. Where our initial approach would have us analyzing thousands of tic-tac-toe positions, we now require only some elaboration upon a clever observation. Thus, it is natural

to ask what is the least amount of work necessary to play a game optimally?

This sort of problem is very familiar to computer scientists. Let us set aside games for a moment and consider another example: the Traveling Salesman Problem (A.37). In this problem, a salesman is given a list of cities which he must visit, and a travel budget. He must start from his home city, visit each city on the list at least once, and then return home without exceeding his travel budget. In a fortuitous turn of events his employer provides food, accommodations, and local travel free of charge, so the salesman only needs to concern himself with making sure that his total airfare comes in under the given budget. Now, as we did before, let us ask: what is the least amount of work necessary to solve this problem?

Unfortunately, this problem has confounded computer scientists for decades. In fact, they have been unable to establish whether or not there is an algorithm which solves the problem within an amount of time which is a polynomial function of the input size – the input size being in this case proportional to the number of cities. In order to circumvent this problem, they have instead created an entire class of problems which are "as difficult as" the traveling salesman problem. This is the class of so-called *NP-complete* problems, and it is constructed in such a way that if anyone finds a polynomial-time algorithm to solve any specific NP-complete problem, that same algorithm can be used to solve every NP-complete algorithm in polynomial time at which point the class will collapse into P (the class of problems which can be solved in time polynomial in the size of the input).

Given this apparently fragile situation, one might compare the class of NP-complete problems to a balloon – as you stuff more and more problems in, it becomes increasing likely that one of them will finally tear a hole and let everything escape. Yet, Cook created the class of NP-complete problems over thirty years ago [16] and in the decades since it has been filled with literally hundreds of problems [33], but despite this there is no indication that the class is in any danger of collapsing. This suggests very strongly that the class of NP-complete problems is not a balloon ready to burst and release its problems into P, but rather an entirely distinct entity full of problems for which no deterministic polynomial-time algorithm exists.

As computer scientists now strongly suspect that NP-complete problems cannot be solved with polynomial-time algorithms, the question naturally turns to how to prove that this is indeed the case. Unfortunately, such a proof has proven just as elusive as a polynomial-time algorithm for solving an NP-complete problem, though not for lack of effort. The question of whether NP-complete problems can be solved

with polynomial-time algorithms is considered so significant that at the turn of the millennium the Clay Mathematics Institute included it in their list of "Millennium Problems" and offered a million-dollar prize to anyone who can solve it.

So where does this leave us and our discussion of how difficult it is to solve games? Well, when we asked about the least amount of work necessary to solve a game, we stepped into a quagmire of computational complexity issues. As it turns out, most games are generally considered to be much harder than NP-complete problems. As we discover through the course of this thesis, the problem of determining whether a winning strategy even exists is typically complete for either PSPACE or EXP, classes which are suspected to contain problems that are even harder than NP-complete problems like the traveling salesman.

Given this, one might be tempted to assume that this is the final word so far as the complexity of games is concerned. This assumption would, however, be premature. Certainly, it is ideal to solve problems with polynomial-time algorithms, but when a problem is shown to be NP-hard that is meant as justification for looking for reasonable algorithms which lie outside of P, not for dropping the problem entirely. To that end, in this thesis we consider games within a relatively new framework: fixed-parameter tractability.

When a problem is shown to be NP-hard, it means that we must make a compromise in order to solve the problem. We can compromise either in solution quality (which is the approach taken by approximation algorithms) or, as fixed-parameter tractability does, in speed. The idea of fixed-parameter tractability is to identify parameters which stay quite small relative to the input size. With such small parameters, we can allow the running time to grow exponentially (or worse) in the parameters so long as the running time grows just polynomially in the size of the input.

Abrahamson, Downey, and Fellows [3] initiated the study of parameterized games, introducing the concept of "short" or k-move games. This concept of k-move games is a generalization of the mate-in-k-moves problem from chess, and creates a parameter which can be applied to every combinatorial game. Short games are also very much in line with the standard heuristic approach to playing combinatorial games, which involves the evaluation of future moves to some upper limit.

Unfortunately, the study of short games has advanced little since its inception. Abrahamson, Downey, and Fellows considered two short game problems: Restricted Alternating Hitting Set (A.20) and Short Generalized Geography

(A.27), which they showed to be in FPT and AW[*]-complete respectively. Downey and Fellows made a number of conjectures regarding short games, including one that AW[*] is "the natural home" for short games [21], but no follow up has been made.

Thus, the focus of this thesis is to advance the study of short games. In the process, we gain some additional insight into Downey and Fellow's "natural home" conjecture, and into several of the classes which lie in the upper reaches of the parameterized complexity hierarchy.

Thesis Overview and Contributions

We begin with a review of basics from computational complexity theory in Chapter 2 and some fundamentals of combinatorial game theory in Chapter 3 and include an algorithmic solution to the game Brainstonz^{TM}(which we refer to hereafter as brainstones).

In Chapter 4 we survey the complexity of combinatorial games in general, and prove completeness for several problems where a game is itself given as input. We also briefly mention some existing complexity results for specific combinatorial games.

The bulk of the new work presented in this thesis begins in Chapter 5. We formally review and study the concept of short combinatorial games in general. We introduce SHORT SUCCINCT WINNING STRATEGY (A.32), which models the problem of determining whether a k-move winning strategy exists from a given position in a given game. We show that this problem is hard for AW[P] (Lemma 8) and in XP when we restrict ourselves to considering games in which the number of moves available from any position is bounded from above by a polynomial function (Lemma 7). We also provide a generic technique for proving that games which are guaranteed to terminate in a polynomial number of moves and are known to be hard for some class X in the classical setting remain hard for X when transformed into k-move problems (Lemma 9).

After that, we develop some techniques for proving the parameterized complexity of short games, which we demonstrate on several short winning strategy problems through the rest of the chapter. We show: SHORT ALTERNATING HITTING SET (A.23) is in AW[*] (Theorem 5), SHORT GENERALIZED HEX (A.28) is in AW[*] (Theorem 6), ENDGAME GENERALIZED CHECKERS (A.9) is in FPT (Lemma 14), ENDGAME GENERALIZED OTHELLO (A.10) is in FPT (Lemma 15), and SHORT GENERALIZED OTHELLO (A.29) is in AW[*] (Theorem 7).

In the two following chapters we explore specific games in more depth. In Chapter 6 we specifically study the mate-in-k-moves chess problem and show that it is AW[*]-complete. Then, in Chapter 7 we consider pursuit-evasion games, and for the first time in our thesis consider parameterizations of games which are not related to the the number of turns remaining. We show that two short game problems, Short Seeded Pursuit Evasion (A.31) and Short Directed Pursuit-Evasion (A.30) are AW[*]-complete, and two other game problems, Seeded Pursuit-Evasion (A.22) and Directed Pursuit-Evasion (A.6) are hard for AW[*].

Finally, we wrap up the thesis with a summary of our contributions and open problems in Chapter 8.

We also have three appendices. In Appendix A we list the definitions of computational problems used in this thesis. In Appendix B we give informal rule sets for many of the games mentioned in this thesis (including tic-tac-toe, connect four, othello, hex, checkers, chess, and go). Lastly, Appendix C contains pseudocode for solving the game brainstones.

Chapter 2

Complexity Theory Primer

In this chapter we survey background material from classical and parameterized complexity theory which is relevant to this thesis. Those who require a more in-depth treatment of these topics are referred to [33] for classical complexity, and to [21] and [25] for parameterized complexity.

2.1 Classical Complexity

Classical complexity theory emerged from the idea that polynomial-time algorithms were desirable [14, 22], but that some problems did not appear to admit such algorithms. This quickly led to the question of whether there was a formal way to prove that these problems absolutely required more than polynomial time to solve. This question remains unsolved, but it has been circumvented to some extent. In [16], Cook showed that every problem in NP could be transformed into an instance of SATISFIABILITY (A.21) in polynomial time, and thus any algorithm capable of solving SATISFIABILITY in polynomial time could solve every other problem in NP in polynomial time as well. Thus, it could be said that SATISFIABILITY is among the hardest problems in NP. Soon after, Karp extended this property to a host of other problems [43]. This new class continued to grow quickly, leading Garey and Johnson to produce their famous text on the subject [33], complete with an expansive appendix of NP-complete problems.

2.1.1 Known Classes and Containments

Definition 1. DTIME($f(n)$) *is the set of decision problems which can be decided by a deterministic Turing machine using* $O(f(n))$ *time.* NTIME($f(n)$) *and* ATIME($f(n)$) *are defined in the same manner, but for nondeterministic and alternating Turing machines respectively.*

Definition 2. DSPACE($f(n)$) *is the set of decision problems which can be decided by a deterministic Turing machine using* $O(f(n))$ *space. We define* NSPACE($f(n)$) *and* ASPACE($f(n)$) *in the same manner, but for nondeterministic and alternating Turing machines respectively.*

Deterministic and nondeterministic Turing machines are defined and explained in many computational complexity texts, including [33]. One may view nondeterministic Turing machines as coming in two flavours: existential and universal. A nondeterministic Turing machine is stuck permanently in one of these two states. An existential nondeterministic Turing machine is what we normally consider to be a nondeterministic Turing machine – one which accepts an input if there is an accepting computation path. Meanwhile, a universal nondeterministic Turing machine accepts an input if all computation paths are accepting. An alternating Turing machine is a Turing machine which is capable of flipping back and forth between these existential and universal states freely. Alternating Turing machines are defined and explored at much greater length in [12].

Note that DTIME(t) $\subseteq$ NTIME(t) $\subseteq$ ATIME(t), since nondeterministic machines are capable of executing deterministic operations and alternating machines are likewise capable of executing nondeterministic operations.

We outline the classical complexity classes which are relevant (if only tangentially) to our work in this thesis in Table 2.1. The key classes are L (logarithmic space), P (polynomial time), PSPACE (polynomial space), and EXP (exponential time). These basic classes are defined for deterministic machines, while classes for nondeterministic and alternating machines are denoted by N and A prefixes respectively.

A key component of complexity theory is establishing the relationships between classes of problems. We already know that L $\subseteq$ NL $\subseteq$ AL, P $\subseteq$ NP $\subseteq$ AP, and PSPACE $\subseteq$ NPSPACE $\subseteq$ APSPACE. The following theorems add a few more containment results:

Theorem 1 (Time Hierarchy Theorem [38])**.** *If f(n) is time-constructible, then:* $\mathrm{DTIME}\left(o\left(\frac{f(n)}{\log f(n)}\right)\right) \subsetneq \mathrm{DTIME}(f(n))$

L	=	DSPACE(log(n))
NL	=	NSPACE(log(n))
AL	=	ASPACE(log(n))
P	=	DTIME($n^{O(1)}$)
NP	=	NTIME($n^{O(1)}$)
AP	=	ATIME($n^{O(1)}$)
PSPACE	=	DSPACE($n^{O(1)}$)
NPSPACE	=	NSPACE($n^{O(1)}$)
APSPACE	=	ASPACE($n^{O(1)}$)
EXP	=	DTIME($2^{n^{O(1)}}$)

Table 2.1: Definitions for common classical complexity classes

A function $f(n)$ is said to be *time-constructible* if there exists a deterministic Turing machine which, given an input string of n ones, outputs the binary representation of $f(n)$ in $O(f(n))$ time.

An analogous result holds for nondeterministic machines. Similar results also hold for space:

Theorem 2 (Space Hierarchy Theorem [71]). *For every space-constructible function* $f : \mathbb{N} \longrightarrow \mathbb{N}$*, there exists a language* ℓ *that is decidable in space* $O(f(n))$ *but not in space* $o(f(n))$*.*

A *space-constructible* function is defined as a time-constructible function, except that the Turing machine is limited to using $f(n)$ space, rather than time.

Theorem 3 (Savitch's Theorem [63]). *For any function* $f(n) \geq \log(n)$: $\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$*.*

Theorem 4. [12] *The following two statements are true:*

1. *Let* $t : \mathbb{N}_0 \to \mathbb{N}_0$ *such that* $t(n) \geq n$ *for all* $n \in \mathbb{N}_0$*. Then* $\text{ATIME}(t(n)^{O(1)}) = \text{DSPACE}(t(n)^{O(1)})$*.*

2. *Let* $s : \mathbb{N}_0 \to \mathbb{N}_0$ *such that* $s(n) \geq log(n)$ *for all* $n \in \mathbb{N}$*. Then* $\text{ASPACE}(s(n)) = \text{DTIME}(2^{O(s(n))})$*.*

From the time and space hierarchy theorems we know that L $\subset$ PSPACE and P $\subset$ EXP (note that we use $\subset$ to indicate a proper subset, which excludes the possibility that the two classes are equal). As a consequence of Savitch's Theorem, PSPACE

= NPSPACE. Similarly, taking Savitch's Theorem in conjunction with the space hierarchy theorem we get the result that NL $\subset$ PSPACE since NL $\subset$ NPSPACE and PSPACE = NPSPACE. Finally, the last theorem regarding the relation between alternating and deterministic classes tells us that AL = P, AP = PSPACE, and APSPACE = EXP. The currently established relationships between these classes are summarized as follows:

$$
\begin{gathered}
\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{AL} = \mathrm{P} \\
\mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{AP} = \mathrm{PSPACE} \\
\mathrm{PSPACE} = \mathrm{NPSPACE} \subseteq \mathrm{APSPACE} = \mathrm{EXP} \\
\mathrm{NL} \subset \mathrm{PSPACE} \\
\mathrm{P} \subset \mathrm{EXP}
\end{gathered}
$$

Before concluding this section, we note that some more restricted forms of alternation have been considered. For $t \geq 0$, the classes Σ_t and Π_t are restrictions of a polynomial-time alternating Turing machine where the machine switches between the existential and universal states at most t times and the starting state is given by the symbol; Σ starts in the existential state and Π starts in the universal state. Σ_0 is equal to NP, as the alternating Turing machine starts in and cannot leave the existential state. Π_0 is equal to co-NP, the complement of NP, since the machine is always in the universal state. The union of all these classes is PH, which is contained within AP [72]. To state this more formally:

$$
\bigcup_{t \geq 0} (\Sigma_t \cup \Pi_t) \quad = \quad \mathrm{PH} \subseteq \mathrm{AP}
$$

2.1.2 Hardness, Completeness, and Reductions

While the theorems presented in the previous section do shed some light upon the relationships between various complexity classes, they by no means offer a complete picture. In many cases the boundaries are fuzzy; by Theorem 1 we know that P $\neq$ EXP and by Theorem 2 we know that L $\neq$ PSPACE. Thus, while we know that P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXP, at least one of these containments must be proper. In fact, all of them are conjectured to be, but computer scientists have been unable to prove this. The question of whether or not P = NP, in particular, remains one of the most famous open problems in computer science, and is one of the problems for which the

Clay Mathematics Institute is offering one million dollars for a solution[1].

In the meantime, *reductions* and the notion of *hardness* were developed to circumvent these fuzzy boundaries. We start with *Karp reductions* and *NP-hardness*:

Definition 3. *A* Karp reduction *[43] from problem X to Y is an algorithm A with polynomial running time, which takes as input any instance I of X and turns I into some instance J of Y such that J is a yes-instance for Y if and only if I is a yes-instance for X.*

A Karp reduction is a many-to-one reduction restricted to a polynomial running time. Nearly all of the reduction we use in the classical complexity setting are Karp reductions.

Definition 4. *A problem X is hard for* NP *(or* NP*-hard) if for every problem $Y \in$* NP*, there exists a polynomial-time reduction from Y to X.*

A problem X which is both NP-hard and in NP is said to be NP-complete. Such a problem can be said to be among the "hardest" problems in NP, since if you were to discover a polynomial-time algorithm to solve X you would have found a polynomial-time algorithm to solve every problem in NP which would in turn prove that P = NP.

Given a Karp reduction from problem X to problem Y, if Y is in P then the reduction yields a polynomial-time algorithm to solve X. Conversely, if X is NP-hard then Y is also NP-hard since every problem in NP can be transformed first into an instance of X (because X is NP-hard) and then into an instance of Y via the reduction.

The notions of hardness and completeness have been generalized as follows [33]:

Definition 5. *A problem X is said to be D-hard for a complexity class C if for every problem $Y \in C$, there is a reduction R where R is in D and R takes as input an instance A of Y and outputs an instance B of X such that B is a yes-instance of X if and only if A is a yes-instance of Y.*

Definition 6. *A problem X is said to be D* complete *for a complexity class C if X is both D-hard for C and in C.*

---

[1]The Clay Mathematics Institute has posted their list of millennium problems on the internet: `http://www.claymath.org/millennium/`.

For the concept of hardness in some class C to be meaningful, it must be with respect to some other class D so that we can construct a scenario where problems which are hard for C are not in D unless $C = D$. In the classical setting, P is typically used for D because the primary concern for most computer scientists is whether or not a given problem admits a polynomial-time algorithm. As such, hardness for NP, PSPACE, and EXP is implicitly proven with respect to P.

However, when proving hardness for P we cannot use polynomial-time reductions because such a reduction would be meaningless; there would be no threat of collapse into because P is already equal to itself. Thus, in this thesis, on the occasions where we prove P-hardness we prove hardness relative to L by using logspace reductions. When using a logspace reduction the space constraint applies only to intermediate processing, while the size of the input and output instances are ignored (otherwise a logspace algorithm would be incapable of reading the entire input).

In the parameterized complexity setting we also employ parameterized reductions, which are discussed in Section 2.2.

Complexity Class	Complete Problem	Reference
L	Undirected Reachability (A.38)	[57]
NL	Directed Reachability (A.7)	[40]
P	Alternating Reachability (A.1)	[40]
NP	Satisfiability (A.21)	[16]
PSPACE	Quantified Boolean Formula (A.19)	[74]
EXP	Succinct Circuit Value (A.34)	[54]

Table 2.2: Complete Problems for Various Complexity Classes

2.2 Parameterized Complexity

Unfortunately, an NP-hardness proof does not contribute directly to solving a problem; it only informs you that finding an algorithm which can solve all instances of a problem exactly and in polynomial time is most likely not possible. Instead, when a problem is NP-hard you must either compromise with respect to solution quality by accepting answers that may not always be optimal, or compromise with respect to running time by accepting algorithms that are not in P. The former choice can lead to approximation algorithms, heuristics, or randomized algorithms. The latter choice leads us to fixed-parameter tractability.

The basic idea of fixed-parameter tractability is to confine super-polynomial growth to a function which depends only on a relatively small parameter (or parameters), while growth with respect to input size remains polynomial. This field was developed by Downey and Fellows, who initially published a series of papers on the topic, eventually culminating in a monograph on the subject [21]. Since then there have been two additional books on the subject, one by Flum and Grohe [25] and one by Neidermeier [48], both in 2006.

2.2.1 Fixed-Parameter Tractability

To begin with, let us formalize the notion of fixed-parameter tractability.

Definition 7. *Consider a parameterized decision problem $< X, k >$ where X is a decision problem and k is a parameter for X. $< X, k >$ is in the class* FPT *if and only if there exists an algorithm A which computes X such that the running time of A is upper-bounded by a function of the form $c \cdot f(k) \cdot p(n)$ where c is an arbitrary (positive) constant, $p()$ is some polynomial function, and $f()$ is a function independent of n.*

The Vertex Cover (A.40) problem is an example of an NP-hard problem which is fixed-parameter tractable. One FPT algorithm for Vertex Cover (from [21]) branches on any edge $(u, v) \in E$. In one branch, delete u from G, decrease k by one, and recurse. In the other branch, delete v, decrement k, and recurse. If there are no edges to choose, return true. If the parameter k reaches zero, return true if $E = \emptyset$ and false otherwise. The size of the search tree for this algorithm is at most 2^k and we only do a polynomial amount of work at each step. Further refinements have resulted in an algorithm for which $f(k) = k^{1.2738}$ [13].

2.2.2 Parameterized Intractability

Naturally, an extended notion of tractability comes with its own notion of intractability. When a parameterized problem does not appear to be in FPT, it is natural to ask whether it is hard for some parameterized class which is conjectured to not be equal to FPT.

Unfortunately the approach of using more powerful computational models which we used in the classical setting does not get us very far in the parameterized setting. In other words, given that FPT = DTIME$(f(k) \cdot n^{O(1)})$, it would be tempting to start

an investigation of parameterized intractability with the class NTIME($f(k) \cdot n^{O(1)}$), known as para-NP [25]. However, para-NP seems too large to be of much use. Among other things, it can be shown that natural parameterizations of some basic problems which appear not to be in FPT, including CLIQUE (A.5), INDEPENDENT SET (A.15), DOMINATING SET (A.8), and HITTING SET (A.14), are not hard for para-NP [25].

Since the simple machine-based approach has quickly lead to an apparent dead-end in the parameterized setting, we instead use problems to define parameterized classes which appear to contain parameterized problems which are not in FPT. Let us first define a *parameterized reduction.*

Definition 8. *A* parameterized reduction *is a fixed-parameter tractable algorithm which transforms an instance* $< I, k >$ *of a parameterized problem* X *and transforms it into an instance* $< J, f(k) >$ *of another parameterized problem* Y *where* $f(k)$ *is indepent of the input size and* $< J, k' >$ *is a yes-instance for* Y *if and only if* $< I, k >$ *is a yes-instance of* X*.*

We can derive problems which are complete for nearly all the classes we need from variations of the following problem:

> r-ALTERNATING WEIGHTED t-NORMALIZED SATISFIABILITY (AWSAT$_{t,r}$)
> *Instance*: A sequence $S_1, \ldots, S_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $S_1 \cup \ldots \cup S_r$, where F consists of $t+1$ alternating layers of conjunctions and disjunctions with negations applied only to variables; integers $k_1, \ldots, k_r$.
> *Question*: Does there exist a size-k_1 subset s_1 of S_1 such that for every size-k_2 subset s_2 of S_2, there exists a size-k_3 subset s_3 of S_3 such that ... (alternating quantifiers) such that, when the variables in $s_1, \ldots, s_r$ are set to true and all other variables are set to false, formula F is true?
> *Parameters*: $k_1, \ldots, k_r$.

Once again we have limited alternation, as we did with the polynomial hierarchy, but these notions of normalization and layers are somewhat new and require some explaination. Consider a boolean formula in conjunctive normal form. It could be said that this formula consists of two layers of operators; the first layer consists of the conjunctions which connect the clauses together, and the second is the disjunctions within the clauses. Thus a CNF formula has two layers. Since the layers alternate between conjunctions and disjunctions precisely once, a CNF formula is 1-alternating.

If we were to modify the CNF formula by replacing some literals with bracketed conjunctions, then the resulting formula would have three layers and be two-normalized. In general, a formula is t-normalized if, when put into a canonical form where each "layer" contains only conjunctions or only disjunctions, there are at most t alternations between conjunctions and disjunctions along any path from the bottom layer to any of the top layers. The concept is derived from a notion in circuits called *weft* – the maximum number of gates which exceed a fixed fan-in (in-degree) along any path from input to output.

By restricting $\text{AWS}\textsc{at}_{t,r}$ we can define complete problems for the parameterized equivalents of both NP and the classes of the polynomial hierarchy. First, let us make a few observations:

An instance of $\text{AWS}\textsc{at}_{t,r}$ is also an instance of both $\text{AWS}\textsc{at}_{t+1,r}$ and $\text{AWS}\textsc{at}_{t,r+1}$ as we can buff up the instance with meaningless padding variables and quantifiers. Also, for $t > 1$ any instance of $\text{AWS}\textsc{at}_{t,r}$ can be transformed into an instance of $\text{AWS}\textsc{at}_{t-1,r+1}$ [25].

What we have here are two different sources of increasing complexity: increasing normalization depth (weft) from t and increasing alternation from r. The resulting set of problems is known as the A-matrix [25] and can be seen in Figure 2.1.

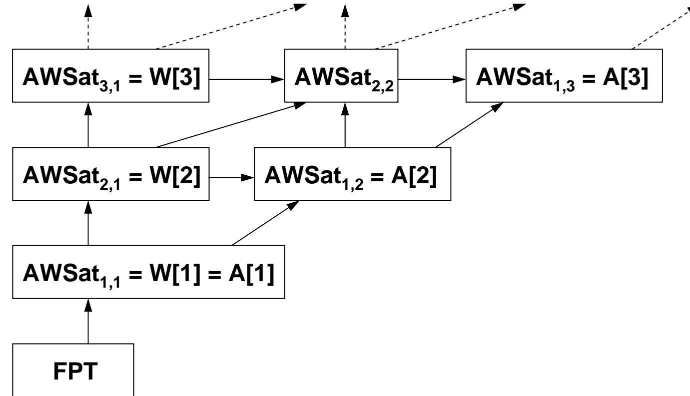

Figure 2.1: The A-matrix.

If the value of r is fixed at $r = 1$ the result is a hierarchy of satisfiability problems with increasing formula depth: the problem Weighted $t-$Normalized Satisfiability (A.42), which is complete for W[t] for all $t \geq 1$ [19].

Weighted t-Normalized Satisfiability
Input: A t-normalized boolean expression X, a positive integer k.

Parameter: k
Question: Does X have a satisfying truth assignment of weight k? That is, a truth assignment where precisely k of the variables which appear in X are true?

Unlike para-NP, the W-hierarchy appears to be a parameterized analogue for NP. CLIQUE and INDEPENDENT SET are complete for W[1] [20], while DOMINATING SET and HITTING SET are complete for W[2] ([18] and [21] respectively).

On the other hand, if the value of t is fixed at $t = 1$, the result is a hierarchy of CNF problems with an increasing number of alternating quantifiers: the problem r-ALTERNATING WEIGHTED CNF SATISFIABILITY (A.2), which is complete for A[r] for all $r \geq 1$ [25].

r-ALTERNATING WEIGHTED CNF SATISFIABILITY
Instance: A sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $s_1 \cup \ldots \cup s_r$, where F is in conjunctive normal form; integers $k_1, \ldots, k_r$.
Question: Does there exist a size-k_1 subset t_1 of s_1 such that for every size-k_2 subset t_2 of s_2, there exists a size-k_3 subset t_3 of s_3 such that ... (alternating quantifiers) such that, when the variables in $t_1, \ldots, t_r$ are set to true and all other variables are set to false, formula F is true?
Parameters: $k_1, \ldots, k_r$.

This hierarchy is the parameterized analogue of the polynomial hierarchy (specifically Σ_r). Note that A[1] = W[1], as $\text{AWSAT}_{1,1}$ is complete for both classes. This mirrors the fact that $\Sigma_0 = \text{NP}$.

Unfortunately, these classes alone are not sufficient for our purposes in this thesis. Games tend to inhabit the upper reaches of the parameterized complexity hierarchy, and thus we must broaden our scope to include W[SAT], W[P], AW[*], AW[SAT], AW[P], and XP. With the exception of XP, these classes can all be derived as restrictions (or generalizations) of the $\text{AWSAT}_{t,r}$ problem.

If we change $\text{AWSAT}_{t,r}$ to include r as an input and treat it as a parameter, the resulting problem is PARAMETERIZED QBFSAT_t (A.17), which is complete for the class AW[*] [21]. Abrahamson, Downey, and Fellows initially defined an AW hierarchy of classes of increasing formula depth (t) similar to the W-hierarchy, but

it collapsed immediately and is now a single class referred to as AW[*] [4, 3]. We further remark that the authors of [25] observed that $\text{AW}[*] = \bigcup_{t \geq 1} \text{A}[t]$, and thus the relationship between AW[*] and the A-hierarchy mirrors the relationship between PH and the polynomial hierarchy in classical complexity. That said, none of the AW classes (AW[*], AW[SAT], and AW[P]) are known or even conjectured to represent any notion of parameterized space.

PARAMETERIZED QUANTIFIED BOOLEAN t-NORMALIZED FORMULA SATISFIABILITY (PARAMETERIZED QBFSAT$_t$)
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $s_1 \cup \ldots \cup s_r$, where F consists of $t+1$ alternating layers of conjunctions and disjunctions with negations applied only to variables (t is a fixed constant); integers $k_1, \ldots, k_r$.
Question: Is it the case that there exists a size-k_1 subset t_1 of s_1 such that for every size-k_2 subset t_2 of s_2, there exists a size-k_3 subset t_3 of s_3 such that ... (alternating quantifiers) such that, when the variables in $t_1, \ldots, t_r$ are set to true and all other variables are set to false, formula F is true?
Parameter: r, $k_1, \ldots, k_r$.

If we instead remove the requirement that the formula be t-normalized from WEIGHTED t-NORMALIZED SATISFIABILITY, the resulting problem is complete for W[SAT] [3]. Analogously, removing the t-normalized requirement from PARAMETERIZED QBFSAT$_t$ yields a complete problem for AW[SAT] [21].

WEIGHTED SATISFIABILITY
Instance: A boolean formula F; a positive integer k.
Question: Does F have a satisfying assignment of Hamming weight k?
Parameter: k

PARAMETERIZED QUANTIFIED BOOLEAN FORMULA SATISFIABILITY (PARAMETERIZED QBFSAT)
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F involving the variables $s_1 \cup \ldots s_r$; integers $k_1, \ldots, k_r$.
Question: Is it the case that there exists a size k_1 subset t_1 of s_1 such that

for every size k_2 subset t_2 of s_2, there exists a size k_3 subset t_3 of s_3 such that... (alternating qualifiers) such that, when the variables in $t_1 \cup \ldots t_r$ are made true and all other variables are made false, formula F is true?
Parameter: $r, k_1, \ldots, k_r$

We can go one step further by giving these two problems more computing power with which to decide whether to accept the variable settings. If we change the problems to take circuits rather than boolean formulas, we get WEIGHTED CIRCUIT SATISFIABILITY (A.41) and PARAMETERIZED QUANTIFIED CIRCUIT SATISFIABILITY (A.18), problems which are complete for W[P] [3] and AW[P] [21] respectively.

WEIGHTED CIRCUIT SATISFIABILITY
Instance: A boolean circuit C; a positive integer k.
Question: Does C have a satisfying assignment of Hamming weight k?
Parameter: k

PARAMETERIZED QUANTIFIED CIRCUIT SATISFIABILITY (PARAMETERIZED QCSAT)
(PARAMETERIZED QCSAT)
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a decision circuit C with the variables $s_1 \cup \ldots s_r$ as inputs; integers $k_1, \ldots, k_r$.
Question: Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2, there exists a size k_3 subset t_3 of s_3 such that... (alternating qualifiers) such that, when the inputs in $t1 \cup \ldots tr$ are set to 1 and all other inputs are set to 0, circuit C outputs 1?
Parameter: $r, k_1, \ldots, k_r$

Finally, we have the class XP. A problem is in XP if it is in P when the parameter is treated as a constant.

Definition 9. *A parameterized decision problem $< X, k >$ is in the class* XP *if and only if there exists an algorithm A which computes X such that the running time of A is $O(n^{f(k)})$ where n is the input size and $f(k)$ is a function which grows independently of n.*

To extend our earlier analogy, XP can be seen as a parameterized EXP. Analogously to EXP and P, a problem which is hard for XP is not in FPT [21].

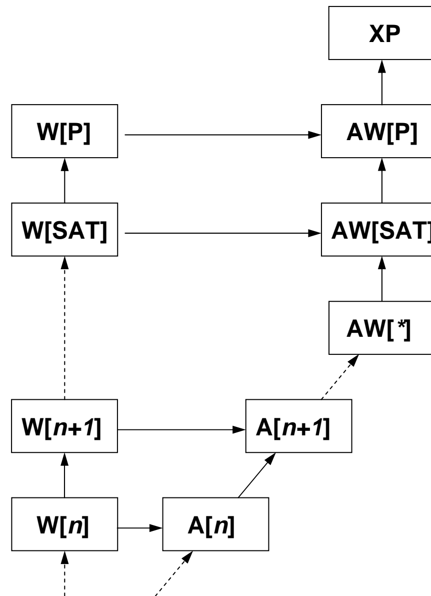

Figure 2.2: The upper reaches of the parameterized complexity hierarchy.

Parameterized Space

As there is no indication that AW[*] is a parameterized space class, one may be curious about what an actual parameterized space class looks like. In [25], the class para-PSPACE is defined to contain all problems which can be solved using at most $O(f(k) \cdot n^c)$ space. The same source observes that para-PSPACE contains AW[*]. This fact can easily be illustrated with the existence a short game which is complete for both PSPACE and AW[*] such as Short Seeded Pursuit-Evasion (A.31), which we discuss in Section 7.4. This means that for every problem in AW[*] there exists a parameterized reduction to a PSPACE-complete problem. In other words, every problem in AW[*] can be transformed in fixed-parameter tractable time into an instance of a problem which can be solved in polynomial space. The reduction may increase the size of the instance in some super-polynomial – though still fixed-parameter tractable – manner, so this does not show that AW[*] $\subseteq$ PSPACE. However, since the size of the new instance is at most some fixed-parameter tractable function of the size of the original instance, and a polynomial of a fixed-parameter tractable function remains a fixed-parameter tractable function, this does show that AW[*] $\subseteq$ para-PSPACE.

2.3 Summary

We have devoted this chapter entirely to reviewing concepts from computational complexity, as a firm understanding of these concepts is essential to the material which follows in later chapters. We require both the general concepts of hardness and membership, as well as an understanding of several specific classes: primarily PSPACE and EXP in classical complexity, and the AW classes from parameterized complexity. However, the presentation may be somewhat non-standard in places as we sought to highlight the concepts most prevalent in the latter parts of this thesis.

That said, we have omitted machine descriptions for most of the parameterized classes we discussed. Readers interested in this can find much information on this topic in [25]. In particular, they discuss using κ-restricted machines (both Turing machines and ARAM machines) to define many classes. For example, W[P] is the set of problems solvable by a κ-restricted nondeterministic Turing machine while AW[P] is the set of problems solvable by a κ-restricted ARAM program.

Chapter 3

Combinatorial Games

In this chapter we touch briefly on combinatorial game theory with a review of some definitions which are important to our complexity work throughout the rest of the thesis. We also discuss solving combinatorial games using the minimax algorithm. To finish the chapter, we survey a number of games which have been solved in the literature – primarily using this minimax approach – and add a new result for the game brainstones. The latter we use as an illustration of the algorithmic techniques presented in this chapter.

3.1 Introduction

Combinatorial game theory is a sizable field of study in its own right, and in this chapter we only scratch the surface. Much of the field originates with books by Conway, Berlekamp, and Guy [15, 10], which have seen recent revisions. A more recent book is [6]. A dynamic survey paper of the field is maintained online [28].

Definitions for what a combinatorial game is vary slightly, but all agree on three key points [10, 15, 50, 51, 28]: players alternate taking turns sequentially, there is no element of chance (that is, all moves are strictly deterministic), and it is a game of *perfect information* (there is no game information hidden from any player). Such games tend to be chess- or go-like in that they place emphasis on the strategic and tactical thinking of the players by removing any element of chance or misinformation. Eliminating any element of chance omits games such as backgammon and parcheesi, as the outcome of those games depends significantly on random die rolls. The prohibition on hidden information omits games such as stratego, where the strength of each

player's pieces are concealed from his opponent. Poker violates both requirements as the players receive random cards, and each player conceals his cards from his opponents.

In our case, nearly all the combinatorial games we consider in this thesis have additional game-theoretic properties worthy of mention:

1. *Two-player*: there are two actively competing agents. Many apparently multi-player scenarios can be turned into two-player situations by reducing the game to "me and my allies" versus "everyone else".

2. *Partisan*: the game is not impartial[1]. That is, there are positions for which moves which are available to one player and not another. In our case, we treat all our games as having two completely disjoint sets of moves, one for each player. This is clearly the case for games like chess, checkers, go, and othello, as the players are only permitted to move or place pieces of their own colour.

3. *Zero-sum*: the sum of the players' outcomes is zero. This terminology is as old as game theory, where a positive value is assigned for a victory and a negative value is assigned for a defeat. Zero-sum games may admit scenarios with varying degrees of victory and defeat; that is, one win can be preferable to another because its value is higher. However, in our case any win is as good as any other win, so we used fixed values for each outcome: +1 for a win, -1 for a loss, and 0 for a draw.

4. *Finite*: all the games we consider have a finite number of positions. We do not explicitly preclude the notion of a game looping in an infinite cycle, but we do discuss how such cycles are typically handled in Section 3.2.1.

3.1.1 Positions, Moves, and Players

We consider a position to be the most basic unit of combinatorial game theory. Formalizing the notion of a position in general is difficult due to the broad nature of games, but informally a position consists of all the information necessary for a player to make a move. This notion becomes easy to formalize given a specific game. Let us consider two examples:

[1]In an impartial game the set of moves available from a given position is always the same regardless of which player's turn it is to move. The only impartial game we consider is SHORT ALTERNATING HITTING SET (A.23), in Section 5.3.

In tic-tac-toe, a position need only consist of the board. The player to move is indicated implicitly by the number of symbols already played (X moves if the number is even, O if the number is odd) since one symbol is placed each turn and none are ever removed. There are no rules which would require any other information. Analogous arguments can be made for many other games, including go-moku, connect four, and hex.

Chess, on the other hand, requires more information to describe a position. This starts with the state of the board and an explicit indication of which player's turn it is to move. Knowledge of whether either king or any rook has moved previously is necessary in order to properly enforce castling rules. Then there is the threefold-repetition rule, which states that if a board position is repeated three times during a game either player may request a draw. Casual players may not enforce this rule, but it is part of the FIDE rule set which governs large tournaments [1]. Therefore, if we are seeking to enforce the rules of standard tournament play, a position must include a complete list of moves from the beginning of the game up to the current turn.

Note that while in both examples we required an indication of which player's turn it is to move, this is because both examples are partisan games. When dealing with impartial games no such indication is necessary.

We also note that all games define a *starting position*, which is the state of the game before the first move is made. For example, the starting position for tic-tac-toe is an empty board.

Once we are familiar with the concept of a position, a *move* can be defined as a transition from one position to another. We require that a move be "complete" in the sense that in the position being moved to it is the next player's turn to move. In other words, we do not allow moves which are "partial" in the sense that the player who made the move gets to move again immediately. As an example, consider checkers: one player may move the same piece many times in his turn if he is able to make consecutive captures. For our purposes, these intermediate steps are not proper moves – it is only a complete move once the player is no longer able to capture (or chooses not to continue making captures) and has ended his turn.

Finally, some terminology notes regarding our players. We refer to them generically as first player and second player, or as player one and player two. These names are assigned based on the position we are given, so if we were given a chess position in which it is black's turn to move we would call black player one in the context of that problem even though the rules of chess state that black is the second player to

move. In specific games they may be given other names based on the pieces they control, for example: X and O in tic-tac-toe, white and black in chess, or robber and cops in pursuit-evasion. We often use these game-specific names when available, as they tend to be more descriptive.

3.1.2 Games, Graphs, and Strategies

We begin by reviewing some properties of combinatorial games.

Definition 10. *If $\mathbb{G}$ is a combinatorial game, then:*

1. *$pos(\mathbb{G})$ is the set of all possible positions in $\mathbb{G}$.*
2. *$move(\mathbb{G}, p, q)$, where $p, q \in pos(\mathbb{G})$, is a boolean function which maps to true if moving from position p to position q is legal in $\mathbb{G}$, and false otherwise.*

Furthermore:

Definition 11. *If $\mathbb{G}$ is a two-player partisan combinatorial game for which the sets of moves available to either player are disjoint, then $pos_x(\mathbb{G})$, where $x \in \{1, 2\}$ is the set of all possible positions in $\mathbb{G}$ from which player x can move.*

Now we are ready to define the concept of a *game graph*.

Definition 12. *A directed graph $G = (V, E)$ is the* game graph *for a game $\mathbb{G}$ if and only if:*

1. *There is a one-to-one function $f\colon pos(\mathbb{G}) \mapsto V$.*
2. *For every pair of vertices $u, v \in V$, $(u, v) \in E \Leftrightarrow move(\mathbb{G}, f^{-1}(u), f^{-1}(v))$.*

Informally, to construct the game graph representation of a game $\mathbb{G}$, start by creating a node for every possible game position $p \in pos(\mathbb{G})$. Then, for each legal move $move(\mathbb{G}, p, q)$, add an edge from the node representing p to the node representing q. Once every legal position and move has been added to the game graph, it is complete. Figure 3.1 shows an example of a partial game graph from tic-tac-toe.

The idea of a game graph is strongly related to that of a *game tree*, sometimes referred to in game theory as an *extensive-form representation*. Game trees are an old concept [47], and we retain the term game tree for a very specific usage which we define along with the minimax algorithm in Section 3.2.

Next, we make an observation regarding the structure of these game graphs:

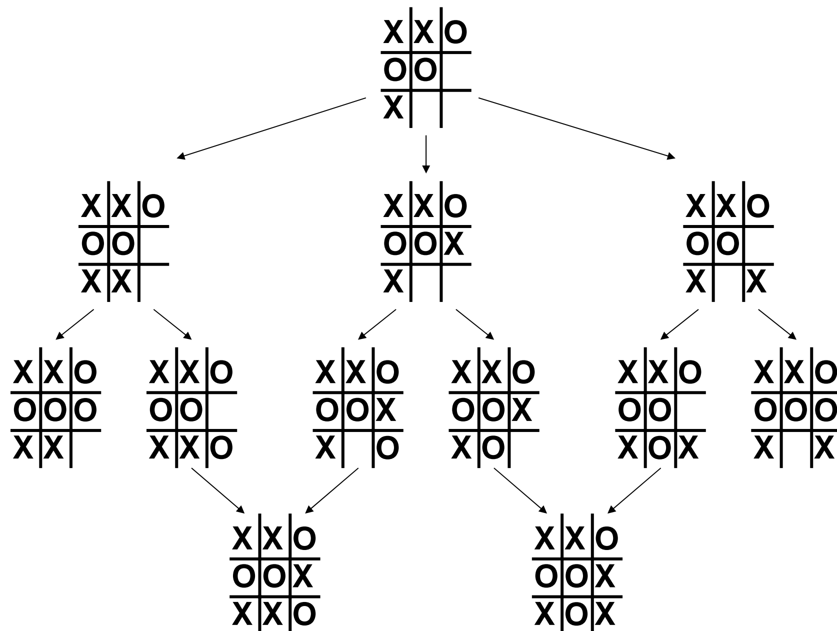

Figure 3.1: A partial game graph for tic-tac-toe.

Lemma 1. *The game graph of a two-player partisan game for which the sets of moves available to either player are disjoint is bipartite.*

Proof. The nodes of such a game graph can be broken up into two sets: the set of nodes V_1 in which it is player one's turn to move, and the set of nodes V_2 in which it is player two's turn to move. Because the sets of moves available to either player are disjoint, the intersection of these two sets is empty ($V_1 \cap V_2 = \emptyset$). A node in either set cannot have an edge to another node in the same set because we have defined moves to be "complete", thus these two sets are partite sets and the graph is bipartite. □

From this Lemma, we observe the following:

Corollary 1. *If $\mathbb{G}$ is a two-player partisan game for which the sets of moves available to either player are disjoint, then:*

1. $pos_1(\mathbb{G}) \cap pos_2(\mathbb{G}) = \emptyset$
2. $pos_1(\mathbb{G}) \cup pos_2(\mathbb{G}) = pos(\mathbb{G})$
3. *For any game graph G of $\mathbb{G}$, the set of vertices V can be partitioned into V_1 and V_2 such that there are two one-to-one functions $f_1 : pos_1(\mathbb{G}) \mapsto V_1$ and $f_2 : pos_2(\mathbb{G}) \mapsto V_2$.*

Next we review the definition of a *terminal* position. Informally, this is a position at which the game ends.

Definition 13. *Given some combinatorial game* $\mathbb{G}$ *and some position* $p \in pos(\mathbb{G})$*,* p *is a* terminal *position if and only if* $move(\mathbb{G}, p, q)$ *is false for all* $q \in pos(\mathbb{G})$*.*

Once the game is over we must declare a winner, or whether the game was a draw.

Definition 14. *Given a two-player combinatorial game* $\mathbb{G}$ *and a terminal position* $p \in pos(\mathbb{G})$*,* $win(\mathbb{G}, p)$ *returns the number of the player who won (i.e.* $win(\mathbb{G}, p) = 1$ *or* 2*), or zero if the game is a draw.*

Now we define what a *strategy* is. We are only concerned about the case of two-player games, as all the games we consider in this thesis are two-player.

Definition 15. *Given a two-player game* $\mathbb{G}$ *with players* x *and* y*, a* strategy *for player* x *in* $\mathbb{G}$ *is a function* $f : pos_x(\mathbb{G}) \mapsto pos_y(\mathbb{G})$ *where for all* $p \in pos(\mathbb{G})$*, either* $move(\mathbb{G}, p, f(p))$ *is true or* p *is a terminal position and* $f(p) = p$*.*

Given a strategy for both players in a two-player game, we can determine the outcome of the game. We define a function for this value:

Definition 16. *Given a two-player combinatorial game* $\mathbb{G}$*, a position* $p \in pos_1(\mathbb{G})$*, and strategies* s_1 *and* s_2 *for players 1 and 2 respectively,* $outcome(\mathbb{G}, p, s_1, s_2)$ *returns one of the three following values:*

1. *0, if the outcome of the game is a draw.*

2. *1, if the outcome of the game is a win for player one.*

3. *-1, if the outcome of the game is a win for player two.*

Given a game graph G of two-player partisan combinatorial game $\mathbb{G}$ for which the sets of moves available to either player are disjoint, we can solve $outcome(\mathbb{G}, p, s_1, s_2)$ for any p, s_1, and s_2 as follows: for all $v \in V_1$, remove all out-edges except for $(v, s_1(v))$. Do likewise with s_2 and V_2. At this point, every vertex in the graph has exactly one out-edge except for the terminal vertices, which have none. This means that from p there is either a path which either reaches a terminal vertex and ends, or else loops back upon itself. In the latter case, $outcome(\mathbb{G}, p, s_1, s_2) = 0$ since the game is effectively a draw. In the former case, the outcome of the game is decided by the terminal node at the end of the path.

Finally, we are ready to define what a *winning strategy* is.

Definition 17. *Given some two-player game $\mathbb{G}$ and some position $p \in pos(\mathbb{G})$, a strategy s_1 is said to be* winning *(or a* winning strategy*) if and only if for all strategies s_2, $outcome(\mathbb{G}, p, s_1, s_2) = 1$.*

3.2 The Minimax Algorithm

In this section we introduce the minimax algorithm. The minimax algorithm can be traced back to a 1928 paper by von Neumann [49], though it has been noted that the idea had already been used for a specific game two centuries prior [53]. This algorithm has been used determine for many games whether the game admits a winning strategy, and is also an underlying concept of many of the heuristics used by computers to play combinatorial games. We are interested in it primarily as a tool for solving games by brute force, which enables us to prove membership results for some games.

```
algorithm minimax(GameGraph G, Position p)
   result := −∞
   for each q in G.outedges(p) do
      f := -minimax(q)
      if f > result then
         result := f
   if result = −∞ then
      return evaluate(p)
   return result
end algorithm
```

The `evaluate` function returns a numeric value based on the state of the game in regards to the person who would move next were the game not over. If the game is a win, it returns a positive value. If the game is a loss, it returns a negative value. In the case of a draw, it returns zero. We use the values $+1$ and -1 for a win and a loss (respectively) as they are the values Conway used in the development of combinatorial game theory [15], though other values can be made to work as well.

The name minimax hints at the way the algorithm works; each player seeks to minimize his opponent's maximum-valued options. In the algorithm presented this is accomplished by negating values so that each player views the negation of what his opponent sees and thus both players are simply maximizing from their perspective.

At its core, this minimax algorithm is a tree traversal. Given a graph which is acyclic but not a tree, it will revisit nodes and repeat the work it has already done at the node. Thus, the number of nodes in a minimax traversal may well be substantially larger than the number of nodes in the actual game graph. We use the term *game tree* specifically to refer to the tree traversal of a graph performed by the minimax algorithm.

3.2.1 Dealing with Cycles

Note that the minimax algorithm presented in the previous subsection does not terminate if the game graph contains a cycle. Here we present a different algorithm for solving game graphs. Instead of traversing the graph we check every vertex of the graph and determine its value by taking the maximum value over the negations of the current values of the child nodes. We repeat this until we complete a pass in which no vertices changed values. When this happens we know that it is safe to terminate, since additional passes will continue to yield the same result.

```
algorithm solve(GameGraph g, position p)
   do
      change := false
      for each node in g
         if (g is unlabeled)
            if |node.children| = 0 then
               node.label := evaluate(node)
               change := true
            else
               max := -∞
               for each child in node.children do
                  if -child.label > max then
                     max := -child.label
               if max != node.label then
                  node.label := max
                  change = true
   while (change)
   return p.label
```

```
end algorithm
```

This revision of the algorithm terminates on graphs with no odd-length cycles. As mentioned earlier, due to how positions and moves are defined the game graph for any two-player game is bipartite.

At this junction we note that many official bodies have gone out of their way to remove cycles from their games. For example, the FIDE laws of chess [1] contain two separate rules that prevent cycles. The first, rule 9.2, is known as *threefold repetition*, and effectively specifies that the player to move may request a draw from the arbiter if, on his turn, he is in a position that he has played from at least twice before. The second, rule 9.3, is known as the *fifty-move rule* and specifies that a player may request a draw from the arbiter on his turn if in the last fifty turns there have been no captures and no pawns moved, or if he intends to make a move that would induce such a scenario. The fifty-move rule also forces polynomial game termination, since there are only $4n$ pieces – each of which can only be captured once – and $2n$ pawns – each of which can be moved at most $n - 2$ times before it must be promoted to another piece.

Similarly, in go the "ko" rule preempts all two-cycles by disallowing them outright. In some rule sets (including those used for official competition in China and the United States) ko is superseded by the "super-ko" rule, which disallows recreating any previous position at all and thus explicitly eliminates all loops from the game graph.

As to why organizations implement rules to remove cycles from game graphs, we suspect that is is for practical purposes; a game which cycles endlessly is effectively a draw, and these rules simply make this explicit while enabling the game to terminate.

3.2.2 Minimax and Practical Use

While the variants of the minimax algorithm described above are theoretically capable of solving games, with currently available hardware there are several games for which the sheer size of the game graph makes finding a solution with this algorithm highly unlikely. For example, checkers, which for which an upper bound of 10^{18} positions has been shown [7], is the largest game (measured by graph size) to have been solved [65]. This was accomplished by means of an incomplete analysis of the game tree – positions that are not reachable under optimal play have not been analyzed – after

many years of computation. Game graphs for similar games such as othello, chess, and go are many orders larger, as we explain in greater detail in Section 3.4.

When the goal is to get a computer to play a combinatorial game which remains as-of-yet unsolved, the basic approach is to terminate the minimax algorithm when it reaches any node at a certain depth and substitute a heuristic evaluation of the current node for an exact win/loss evaluation of the branch of the game tree rooted at that node. This method dates all the way back to a 1950 paper by Claude Shannon about computer chess [70]. Since then, additional heuristics such as alpha-beta pruning [60] have been adopted to cull branches of the game tree and improve performance. The alpha-beta pruning scheme is among the oldest and most basic, but more sophisticated methods have since been developed. However, a proper treatment of this topic would require much more space and falls outside the scope of this thesis.

3.3 Solving Brainstones

Brainstonz™is a new game developed by Philippe Trudel for McWiz games[2], first published in 2007 and marketed under the name "BrainStonz™" (also under the name "BrainPucks™", with a set of hockey-themed pieces). We refer to the game here as brainstones.

As this is a relatively new game, we are not aware of anyone having performed any analysis of it. In this section, we explain how to solve the game as an example of solving a game with the minimax technique. The rules of the game are as follows:

1. The game is played on a four by four board. Each square on the board is marked with one of eight different symbols, and each symbol is used exactly twice.

2. There are two players, each of which has his own set of coloured stones. For sake of explanation, let us say that first player's stones are black and second player's are white.

3. On the first turn, the first player places one of his black stones on a square. Second player then moves and they alternate thereafter.

4. On every turn after the first, the player to move places two stones on unoccupied squares. If, during the course of his turn, he places a stone on a symbol for

[2]Website: www.mcwiz.com

which the matching symbol is already covered by one of his stones, he must immediately remove one of his opponent's stones. It is perfectly legal to place one stone to create such a match, remove an opposing stone, and then place your second stone on the square you just cleared.

5. The first player to place four of his stones in a line (horizontally, vertically, or diagonally) wins.

6. If a player is unable to play (because no free squares remain to place a stone on) the game is a draw.

We can quickly calculate an upper bound on the number of possible positions in this game as follows: given that there are 16 squares, each of which could be in one of three states, and it could be either black or white's turn to move, we get $3^{16} \cdot 2 = 86093442$. This is actually a relatively small number as far as combinatorial games are concerned. Connect Four has a much larger game graph (estimated at 10^{14}) and yet was solved by brute-force computer methods over two decades ago in 1988. That said, this problem still taxes the computing resources of average machines. In adjacency list format, the game graph would require several hundred gigabytes of storage space. As such, we elected to generate the tree on the fly. Our solution stores only a handful of flags for each node, which are necessary for the `solve` algorithm presented at the beginning of Section 3.2.1. We associate these flags with the positions in the game graph through a perfect hash function, which in essence transforms a 16-bit trinary string (an accurate board representation, since each square of the 4x4 board can be empty or occupied by one of two different-coloured types of piece) into a binary string.

Pseudocode for our solution can be found in Appendix C. The results state that under optimal play the game is a draw. That is, neither player has a winning strategy,

This concludes our investigation of brainstones for now. However, the techniques we review in Chapter 4 prompt us to ask new questions regarding the game in Section 4.6.

3.4 Other Solved Games

Of the games listed in Table 3.1, only two have been solved. Checkers, as we mentioned earlier, is weakly solved (positions which are not reachable under optimal play

have not all been analyzed) and the analysis took roughly ten years to complete [65]. Connect four was solved by James D. Allen in October of 1988 [7].

Connect Four	10^{14}	[7]
Checkers (8x8)	$5 \cdot 10^{20}$	[65]
Othello	10^{28}	[7]
Chess	10^{50}	[7]
Go	10^{171}	[76]

Table 3.1: Upper bounds on the sizes of game graphs for some well-known games.

Partial solutions exist for some games as well. Othello on a six by six board has been solved [24], but larger board sizes remain unsolved. Chess is nowhere near a complete analysis, though endgame databases for board positions including up to 6 pieces (excluding positions involving 5 pieces vs. one king) are available for download on the web [75].

For comparison's sake, a rough estimate of the number of atoms in all the stars the observable universe come to about $4 \cdot 10^{79}$ [11]. This puts the number between the size of the game graphs for chess (10^{50}) and go (10^{171}), which suggests that a complete analysis of the game graphs for these games may never be feasible by conventional brute-force computational methods.

A more comprehensive list of solved and partially-solved games can be found in [77].

Strategy Stealing

Before concluding our investigation into solved and partially-solved games, there is one more technique for us to discuss. We mentioned the strategy stealing argument earlier, in our introduction. This argument was first use by Nash on the game of hex [32]. The general form of strategy stealing is as follows: should the rules of some game $\mathbb{G}$ meet certain conditions, then if someone were to supply a supposed winning strategy for player two in $\mathbb{G}$, player one could steal that strategy by making his initial move at random and then playing as if he were actually player two. Thus, if there exists a winning strategy for player two then there also exists a winning strategy for player one, which is a contradiction (only one player can win in a zero-sum game) and thus there is no winning strategy for player two. The certain conditions are that both players have symmetric winning conditions, and having made that extra initial move cannot cause first player problems later in the game.

As we mentioned, this argument was been applied to hex. There are no ties in hex, so the lack of a winning strategy for player two implies that there is one for player one. The strategy-stealing argument can also be applied to tic-tac-toe, as we did in the introduction of this thesis. More generally, it applies to any instance of the generalized (m, n, k) game (essentially a game of tic-tac-toe played on an m by n board with the goal of getting k symbols in a row).

3.5 Summary

We had two primary goals in this chapter. The first was to review concepts and definitions from combinatorial game theory which would enable us to speak about combinatorial games and game graphs in a mathematically precise manner, much as we reviewed computational complexity in Chapter 2. Being able to model games with graphs is an important step which we build upon in the next chapter, when we consider the problem of finding a winning strategy for a game which is given as an input in graph form.

The second goal of this chapter was to discuss games which are already solved and the techniques by which these solutions were found. In nearly all the cases we covered the technique used was the minimax algorithm, including our solution for the game brainstones.

Chapter 4

Classical Complexity of Games

In this chapter we begin to venture into territory which can be called Algorithmic Combinatorial Game Theory: studying the complexity of finding winning strategies. First, we deal with a simple, but important, observation: the rules of chess give a fixed board size and maximum number of pieces, therefore the game graph for chess is of constant size and any algorithm we run on this graph (such as the minimax algorithm) runs in constant time. However, this observation is neither interesting nor illuminating, but rather a bit misleading; we already observed in Section 3.4 that the size of the game graph in chess is well beyond our means to compute.

Ultimately, classical computational complexity is about analyzing the resources required to solve a problem in regards to the input size, and in chess a given board position would naturally be an input. Therefore, to put aside this misleading, degenerate constant-time complexity result, we treat the size of the board as the input size (say, an $n \times n$ board). To do this, we consider *generalized* versions of games where the size of the input board is not strictly stated as part of the rules, but is instead determined by the input. For example, games like chess and othello are generalized for play on an $n \times n$ board. In the case of some games, such as othello, the rules are already adapted to any board size. However, in the case of a game with more complicated rules additional adjustments may be necessary; for example, in chess it is not immediately apparent what the starting position on an $n \times n$ board looks like and thus it must be defined (as we do in Chapter 6).

Now we are ready to study complexity issues which arise in the process of determining whether generalized games admit winning strategies. Table 4.1, taken from [73], outlines some known memberships for such games. A similar table can be found in [17].

Type	Game	Algorithm
1	Logarithmic Space	Polynomial Time
2	Polynomial Time	Polynomial Space
3	Linear Space	Exponential Time

Table 4.1: Resource bounds for determining the existence of winning strategies in various types of games [73].

In Table 4.1, games of the first type are those in which the number of reachable positions is bounded by a polynomial. For example, the cop-and-robber game in which a single cop chases a single robber around a graph [55, 52] falls into this class: the cop and robber each occupy a vertex, thus there are $O(n^2)$ possible positions.

Games of the second type are those in which the game is guaranteed to end in at most a polynomial number of moves. We refer to these games as having *polynomial termination* (which we formally define below) while [39] and [17] refer to them simply as *bounded*. Tic-tac-toe, hex, and othello are natural examples of such games: each move involves a player putting a piece in an unoccupied square, pieces are never removed, and the game must terminate when all squares are occupied. For such games the size of the board is an upper bound on the length of the game.

In games of the third type (also called *unbounded* [39, 17]), the number of reachable positions can be bounded by an exponential function (in the size of the input). An exponential bound is always sufficient as a polynomial number of bits can describe at most an exponential number of positions. Checkers is an example of such a game [62].

We point out that in Table 4.1, the "Game" column describes the resources sufficient for an alternating Turing machine to determine whether a winning strategy exists, while the "Algorithm" column gives sufficient resources to solve the same problem with a deterministic Turing machine.

Our goal in this chapter is to present some intuition behind Table 4.1 and, more importantly, to present some hardness results which complement it.

4.1 Type 1 Games – Polynomial Graph Traversal

As listed in Table 4.1, games of the first type are those in which the number of reachable positions is bounded by a polynomial. An early result regarding such games can be found in [42], where a problem called Game (A.11) is shown to be P-complete.

GAME
Input: A game $\mathbb{G} = (P_1, P_2, W_0, s, M)$, where P_1 and P_2 are disjoint sets (of positions for player 1 and 2 respectively), $W_0 \subseteq P_1 \cup P_2$ (the winning positions), $s \in P_1$ (the starting position), and $M \subseteq P_1 \times P_2 \cup P_2 \times P_1$ (the set of allowable moves).
Question: Does player one have a winning strategy in $\mathbb{G}$?

GAME was shown to be in P using a retrograde version of the `solve` algorithm we presented in Section 3.2.1, and P-hard by reduction from GEN, the problem of determining whether a given element x of some given set X is in the smallest subset of X which is closed under a given binary operator [42].

We present an alternative formulation of the problem, complete with a different (and new) hardness reduction, which is more consistent with the terminology and techniques we use in this thesis. The reduction also enables us to highlight the relationship between finding winning strategies and solving ALTERNATING REACHABILITY (A.1).

WINNING STRATEGY
Input: A bipartite directed graph $G = (V_1 \cup V_2, E)$ where V_1 is the set of positions in which it is player one's turn to move and V_2 is the set of positions in which it is player two's turn to move, position $p \in V_1$, and sets $W \subseteq V$, $L \subseteq V$, $D \subseteq V$ of winning, lost, and drawn positions respectively.
Question: Does player one have a strategy which guarantees victory in the game described by G, W, L, and D from position p?

Note: We sometimes describe an instance of WINNING STRATEGY as a tuple $(V_1, V_2, E, p, W, L, D)$.

To show that WINNING STRATEGY (A.44) is P-complete, we use the following problem:

ALTERNATING REACHABILITY
Input: A directed graph $G = (V, E)$, a partition $V = X \cup Y$ of the vertices, and designated vertices s, t.
Question: Is $apath(s, t)$ true? *apath* is defined as follows. Vertices in X are "existential" while those in Y are "universal". Such a graph is called

an *alternating* graph or an AND/OR graph. The predicate $apath(s, t)$ holds if and only if

1. $s = t$, or
2. s is existential and there is a $z \in V$ with $(s, z) \in E$ and $apath(z, t)$, or
3. s is universal and for all $z \in V$ with $(s, z) \in E$, $apath(z, t)$ holds.

Note: We sometimes describe an instance of ALTERNATING REACHABILITY as a tuple (V, E, X, Y, s, t).

The ALTERNATING REACHABILITY (A.1) problem is P-complete [40]. Its variant, BIPARTITE ALTERNATING REACHABILITY, where the graph is bipartite with X and Y being the partite sets, is also P-complete [35].

Clearly, the problems WINNING STRATEGY and BIPARTITE ALTERNATING REACHABILITY have much in common. Both are "played" on directed graphs, and the concept of nodes being existential or universal is functionally equivalent to the game notion of either first player or his opponent being allowed to move from a node. The difference between the two problems lies in the target vertices; BIPARTITE ALTERNATING REACHABILITY has one target vertex t, while WINNING STRATEGY may have many target vertices in both W and L. Therefore, we embark upon the following lemma:

Lemma 2. WINNING STRATEGY *is P-complete.*

Proof. We prove this statement by showing that WINNING STRATEGY is equivalent BIPARTITE ALTERNATING REACHABILITY, which is known to be P-complete [35].

First, we reduce from WINNING STRATEGY to BIPARTITE ALTERNATING REACHABILITY, showing that WINNING STRATEGY is in P.

For the most part, our reduction leaves the original inputs of the BIPARTITE ALTERNATING REACHABILITY instance largely unchanged. However, we face one significant obstacle in the fact that an instance of WINNING STRATEGY may have many terminal positions in which player one wins, while BIPARTITE ALTERNATING REACHABILITY has only a single destination vertex. We contract those winning positions into a single vertex as follows.

First, let us start by observing that, given a game graph and sets W and L, the set of positions in which first player has won is $(W \cap V_1) \cup (L \cap V_2)$. That is, either

it is player one's turn to move and he has won, or it is second player's turn to move and he has lost. Now, we pick a vertex in $W \cap V_1$ and call it t. Similarly, we take any vertex in $L \cap V_2$ and call it t'. We let W' denote the set $(W \cap V_1) \cup (L \cap V_2)$.

The set of arcs E' is almost a complete copy of E, with two exceptions. The first exception is that we modify all arcs (u, v) where $v \in W'$; we replace such arcs with either (u, t) if $v \in V_1$ or (u, t') if $v \in V_2$. The second exception is that we add a single extra arc (t', t).

Thus, given an instance $(V_1, V_2, E, p, W, L, D)$ of Winning Strategy, we transform it into an instance $(V_1 \cup V_2, E', V_1, V_2, p, t)$ of Bipartite Alternating Reachability.

If player one has a winning strategy in G, then he can reach a vertex $v \in W'$ regardless of what out-edges his opponent chooses to follow from his nodes. Player one can use this same strategy to get from p to t in G', since the starting positions and partite sets are the same, and every vertex in W' was contracted into either t or t' (and t' has exactly one out-edge which leads to t).

Similarly, if there is an alternating path from p to t in $G' = (V_1 \cup V_2, E')$ then in G player one must be able to reach some winning position $v \in W'$, since all the vertices in W' were contracted into or lead to t.

This reduction requires only logarithmic space: merely copying data requires only two pointers of size $log(n)$ – one for the input position and one for the output position. To modify E', we require two additional pointers to t and t'.

Now we reduce from Bipartite Alternating Reachability, showing that Wining Strategy is P-hard, as follows:

This time, given an instance (V, E, X, Y, s, t) of Bipartite Alternating Reachability, we output the Winning Strategy instance $(X, Y, E, s, \{t\} \cap X, \{t\} \cap Y, \emptyset)$.

Again, this reduction uses only logarithmic space: we need only two $\log(n)$-sized pointers, to copy data from the input instance to the output instance. We perform no modifications to the actual graph, and so we need no additional pointers. The one place where we may do some calculation is for W and L, but the definitions given ($\{t\} \cap X$ and $\{t\} \cap Y$ respectively) are effectively just $W \leftarrow \{t\}$ if $t \in X$ and $L \leftarrow \{t\}$ if $t \in Y$. In other words, t is a winning position if player one would move from it, otherwise it is a losing position.

Lastly, we show that in this reduction a winning strategy in $G' = (X \cup Y, E)$ is a strategy for reaching t from s in $G = (V, E)$ and vice-versa. The graphs are identical,

as only the target has been transformed. However, the targets remain essentially the same; if t is in X then it is also in W, which means moving to it is a win for player one since he would move from it. Similarly, if t is in Y then it is also in L, meaning that moving into it in this case is also a win for player one since it is a loss for player two. This means that the one and only way for player one to win is by getting to t (since there are no other vertices in W or L). Thus, if player one has a winning strategy then he can get from s to t, and vice-versa. □

4.2 Type 3 Games – Unbounded

Games of the third type in Table 4.1 are those in which the number of reachable positions is unbounded. This often results in an exponential number of positions due to the necessity of representing the board position with a polynomial number of bits. One might wonder why we have gone from games of the first type to games of the third. The reason is that we model games of the second type as a restriction of the problem which we use to model games of the third type, and thus presenting them in this order results in a more logical progression of ideas.

In our analysis of games of the first type, the problem included a game graph as part of its input. This is problematic for games of the third type as they have exponentially-large game trees.

Moreover, combinatorial games are never played directly on their game graphs. Instead, combinatorial games typically present some board, a set of pieces which can be distributed about the board in combinatorially many ways, and a list of of rules which dictates what positions are terminal (win, lose or draw) and to what positions you may move to from any given position. For example: when presented with a chess problem you are shown a board configuration (and, implicitly, the rules of chess), but you are not presented a game graph. In this case, the input is the board, and thus the size of the input is the size of the board rather than the size of the game graph (as it is in Winning Strategy).

This brings us to a concept known as *succinct representation*. Informally, a succinct representation describes something exponentially larger than the description itself. For example, a graph G could be described with a circuit which takes as input two vertices (u and v) represented as bitstrings, and returns as output a single true/false value depending on whether G includes an edge from u to v. Such a representation is called a succinct circuit representation. Research into the complexity of

graph problems which use such a representation began with [31] and [54], which found that using succinct circuits typically results in in exponential complexity blow-up; L- and NL-complete graph problems become PSPACE-complete, P-complete problems become EXP-complete, and NP-complete problems become NEXP-complete. Several conversion lemmas, starting with [54], make it possible to lift hardness reductions for the original non-succinct problems into the succinct setting, and thus reuse them to obtain hardness results for the succinct variants.

For our purposes, a version of Winning Strategy where the graph G and sets W, L, and D are defined succinctly by circuits seems a reasonable model of the problem human game-players are presented with because evaluating game rules takes at most polynomial time, and the circuit value problem is P-complete. If there is anything wrong with the succinct circuit model, it is that that we suspect circuits (or rather, polynomial-time rule verifiers) may be too powerful; we can envision algorithms to decide M, W, L, and D for tic-tac-toe using only logarithmic space. In fact, it might be fair to differentiate type 2 and type 3 games based on the power they require for the verifiers: type 2 games appear to require only logarithmic space, while verifiers for type 3 games seem to require polynomial time.

We define this new problem using succinct circuits as follows:

Succinct Winning Strategy
Input: A circuit M which takes in two positions as bit strings p_1, p_2 and outputs true if moving from p_1 to p_2 is legal, a bitstring p representing a starting position, circuits W, L, and D which take as input a bitstring representing some position q and return true if the player to move from q has won, lost, or drawn the game respectively.
Question: Starting from position p, does player one have a winning strategy in the two-player game described by circuits M, W, L, and D?

Note: We sometimes describe an instance of Succinct Winning Strategy as a tuple (M, p, W, L, D).

To classify this problem, we turn to the succinct variant of the problem we used in the non-succinct setting: Succinct Alternating Reachability (A.33). This problem has been shown to be EXP-complete [9].

Succinct Alternating Reachability
Input: A circuit E describing a bipartite alternating graph which takes

in two positions as bit strings p_1, p_2 and outputs true if there is an edge from p_1 to p_2 in the graph, a starting position s represented as a bitstring, a destination vertex t.
Question: Is it possible to reach t from s in the graph described by E?

Note: We sometimes describe an instance of Succinct Alternating Reachability as a tuple (E, s, t).

Even in succinct forms, Succinct Alternating Reachability and Succinct Winning Strategy (A.36) bear the same resemblance to one another as their corresponding original problems. Thus, we can show that Succinct Winning Strategy is EXP-complete by reworking Lemma 2 for the succinct setting.

Lemma 3. Succinct Winning Strategy *is EXP-complete.*

Before we prove this lemma, we present a generic transformation which we can use on any game represented by succinct circuits to make the partite sets of positions distinct. Given a moves circuit M, we construct M' as follows:

$$M'(b,c) = \left(b_0 \wedge \neg c_0 \wedge \bigwedge_{0<i<|b|} (b_i = c_i) \right) \vee \left(\neg b_0 \wedge c_0 \wedge M\left(b_{1\ldots|b|-1}, c_{1\ldots|c|-1} \right) \right)$$

The notation b_x refers to the xth bit of b (indexed from 0) while the notation $b_{x\ldots y}$ indicates the substring consisting of bits x through y.

This transformation adds an additional bit at the front of the bitstrings which represent positions. This extra bit denotes which player's turn it is to move. Specifically, given some position b, if $b_0 = 0$ then it is first player's turn while if $b_0 = 1$ then it is the second player's turn. This technique is used several times in this section so that we do not have to construct more complex circuits to derive which player's turn it is.

Now we are ready to prove Lemma 3.

Proof. We show that Succinct Winning Strategy is EXP-complete by showing that it is equivalent to Succinct Alternating Reachability.

First, we prove that Succinct Winning Strategy is EXP-hard by reduction from Succinct Bipartite Alternating Reachability.

Our reduction is as follows: set M equal to E, only with the modification described above which uses the first bit (the bit numbered zero) to denote the partite sets. Set p equal to s. Circuit $W(b) = (b_{1\ldots|b|-1} = t \wedge \neg b_0)$. Circuit $L(b) = (b_{1\ldots|b|-1} = t \wedge \neg b_0)$. In effect, W checks if the input is t and position t represents a existential vertex, while L checks if the input is t and position t represents a universal vertex.

Thus, given an instance of SUCCINCT BIPARTITE ALTERNATING REACHABILITY (E, s, t) our reduction produces the SUCCINCT WINNING STRATEGY instance $(E', s, W, L, \emptyset)$ where W and L are as described in the previous paragraph.

If there is a strategy to get from s to t in the SUCCINCT BIPARTITE ALTERNATING REACHABILITY, then player one can use that strategy to win the game represented by (E', s, W, L, D) since the graphs represented by M and E are isomorphic, retaining even the same existential/universal natures of their vertices, and both the starting (s, p) and end positions (t) are the same. The end position must be t because it is the only position for which W or L returns true, and they always do so in a manner which awards victory to player one: when the game reaches position t it will either be in an existential node – in which case $W(t)$ is true and it is player one's turn, so he has won the game – or a universal node – in which case $L(t)$ is true and it is second player's turn, again resulting in a win for player one.

On the other hand, if there is a winning strategy for player one in the game (E', s, W, L, D) then he must be able to force the game from position s to position t (since W and L can only award victory when the game reaches position t), meaning that he has a strategy to get from s to t in the graph represented by E' (again, since the graphs are identical).

We prove membership by reducing SUCCINCT WINNING STRATEGY to SUCCINCT BIPARTITE ALTERNATING REACHABILITY.

As was the case in Lemma 2, our biggest obstacle is turning multiple winning positions into one single winning position. In the previous lemma we contracted vertices, but in this case we seek to avoid modifying the inner workings of the circuit M. Instead, we shall add an extra vertex which all the winning positions will have an edge to.

To accomplish this we construct a new circuit E' by first transforming M to M' to mark the partite sets as described above, and then we build E' from M' as follows:

$$E'\left(p^1,p^2\right) = \left(M'\left(p^1,p^2\right) \wedge \neg W\left(p^1\right) \wedge \neg L\left(p^1\right) \wedge p^2 \neq t'\right) \vee \left(W\left(p^1\right) \wedge p_0^1 = 0\right) \vee \left(L\left(p^1\right) \wedge p_0^2 = 1\right)$$

Vertex t' is new – a larger-valued bitstring than any existing position in the original instance (meaning that we may have to use bitstrings that are one bit longer).

Thus given an instance of Succinct Winning Strategy (M, p, W, L, D), our reduction is to transform into the Succinct Bipartite Alternating Reachability instance (E', p, t') where E' and t' are described above.

If there is a winning strategy for player one in the game (M, p, W, L, D) then there is an alternating path from s (p) to t in the graph described by E. The "extra" logic we added to build E' from M' concerns edges to t, which come from positions in which player one has won (either it's his turn and W accepts the current position, or it is player two's turn and L accepts the current position). In other words, player one can use his winning strategy in the reachability graph to get to a position which was a win for him, at which point the game must proceed to t'.

On the other hand, if there is an alternating path from s to t in the Succinct Bipartite Alternating Reachability instance, then there is a winning strategy in the original game since the only way to reach t is through a node which represents a win for player one. □

4.3 Type 2 Games – Polynomial Termination

Games of the second type, as given in table 4.1, are those for which the game is guaranteed to terminate in at most a number of moves which is polynomial in the size of the board (or position). In [39] these are referred to simply as bounded games. Let us begin by defining this concept formally.

Definition 18. *A combinatorial game* $\mathbb{G}$ *has* polynomial termination *if and only if there exists a polynomial function* $p()$ *such that* $p(n)$ *(where* n *is the size of a position in* $\mathbb{G}$*) is an upper bound for the length of the longest path in the game graph of* $\mathbb{G}$*.*

Our first observation is that, with some minor adjustments to accommodate the inputs, we can use the minimax algorithm to solve polynomially-terminating instances

of Succinct Winning Strategy in polynomial space (proof of this follows below in Lemma 4):

```
algorithm succinct_minimax(Circuits M, W, L, D, Position p)
   result := −∞
   for each bitstring q of length |p| do
      if circuit_value(M, (p, q)) then
         f := -minimax(M, W, L, D, q)
         if f > result then
            result := f
   if result = −∞ then
      if circuit_value(W, p) then return 1
      if circuit_value(L, p) then return -1
      return 0
   return result
end algorithm
```

Here, the function `circuit_value(C,v)` returns the output value of circuit C for input v. The Circuit Value problem (A.4) is known to be P-complete [44], and thus this function can be completed in polynomial time. In all cases here the circuits we use return a single bit, which we treat as a true/false value.

> Circuit Value
> *Instance*: A encoding of boolean circuit α, inputs $x_1, \ldots, x_n$, and a designated output y.
> *Question*: On inputs $x_1, \ldots, x_n$, is output y of α true?

We formally prove that this algorithm finds winning strategies for games with polynomial termination below. We also show that the problem is hard for PSPACE, thus making it PSPACE-complete.

Lemma 4. Succinct Winning Strategy *with a guarantee of polynomial termination is PSPACE-complete.*

Proof. First, we prove that Succinct Winning Strategy with a guarantee of polynomial termination is in PSPACE by using the `succinct_minimax` algorithm

presented above. To iterate through all possible bitstrings in lexicographical order we simply perform the increment-by-one operation each time the loop iterates. This enables us to check every possible position to see whether moving to it from the current position is a valid move, and thus we can reproduce the set of possible moves from the current position. With this done, the rest of the algorithm functions as before. This algorithm uses at most polynomial space since at any given time it will be at most some polynomial number deep in recursive calls (due the the polynomial bound on the number of moves), and each recursive call stores just the one bitstring needed to record the position of the loop and the current value of the *result* variable (plus some constant-size maintenance data).

Second, we prove that Succinct Winning Strategy with a guarantee of polynomial termination is PSPACE-hard, by reduction from Quantified Boolean Formula (A.19). Intuitively, given a Quantified Boolean Formula instance (F) we create a game (M, p, W, L, D) where player one chooses the existential variables, player two chooses the universal variables, and player one wins if the boolean formula is satisfied once all the variables have been set.

In the Succinct Winning Strategy instance we construct, game positions are represented by bitstrings of length equal to the number of variables in the original instance plus the number of quantifiers. The number of quantifiers is equal to the number of variables, so if v is the number of variables in the given Quantified Boolean Formula instance, then our bitstrings are of length $2v$.

We construct W and L depending on the number of quantifiers in the Quantified Boolean Formula instance. If the number of alternations in the quantifiers at the beginning of F is odd then W is a circuit which checks whether the first v bits satisfy F and the last v bits are all true, and L is a circuit which always returns false. In the other case – the number of alternations in the quantifiers at the beginning of F is even – the circuits are swapped; L checks the first v bits against F and that the last v bits are all true, while W always returns false.

For circuit D we can use any circuit which always returns false – for example, the conjunction of the first bit of the input and its negation ($b_0 \wedge \neg b_0$). For the starting position p, we give the all-zero bitstring.

This leaves circuit M, which we now describe. Recall that our bitstrings are of length $2v$. The first v bits of the bitstring represent the current variable assignment while the last v bits give the number of the current quantifier in unary. For ease of notation, let us say that $Q(b)$ is the unary number given by the last v bits of b.

Given two bitstrings b^1 and b^2, M checks that $Q(b^2)$ is equal to $Q(b^1)+1$, that the first $Q(b^1)$ bits of b^1 and b^2 are identical, and that bits $Q(b^2)+1$ through v of b^2 are all zeros. To be more precise:

$$\begin{aligned} M\left(b^1,b^2\right) \;\; = & \bigwedge_{0\le i<v-1} \left(b_i^1 = b_i^2 \vee \left(b_i^1 = 0 \wedge b_i^2 = 1 \wedge b_{i+1}^1 = 0 \wedge b_{i+1}^2 = 0\right)\right) \\ & \wedge \bigwedge_{0\le i<v} \left(\left(\neg b_{v+i}^1 \vee b_i^1 = b_i^2\right) \wedge \left(\neg b_{v+i}^2 \vee b_v^2 = 0\right)\right) \end{aligned}$$

Note that when constructing an actual circuit, the equality operator $(=)$ can be implemented using a XOR gate $(\otimes)$ since $(x = y)$ is equivalent to $\neg(x \otimes y)$.

Intuitively, this circuit ensures that on each turn we pick a value for the first remaining undecided variable. In other words, the only legal moves in the game consist of picking a value for the next variable of the QUANTIFIED BOOLEAN FORMULA instance (either true or false).

Specifically, the circuit must verify three conditions to return true: that $Q(b_2) = Q(b_1)+1$, that the first $Q(b_1)$ bits of both bitstrings be identical, and that $b^2_{Q(b_2)\ldots v-1}$ is an all-zero string. If $Q(b_2) = Q(b_1)+1$ fails then the counter tracking the number of the variable to set next has been thrown off. If the first $Q(b_1)$ bits of the two strings are not identical, then a player may have altered one of his opponent's previous moves. Finally, if $b^2_{Q(b_2)\ldots v-1}$ is not an all-zero string then someone is attempting to set variables before they are supposed to be set. On the other hand, if all three conditions are met then the value chosen for bit $Q(b_2)$ will be frozen for the rest of the game, and thus the player has picked a value for that variable.

Given a scheme for picking existential variables to satisfy the QUANTIFIED BOOLEAN FORMULA instance we have a winning strategy for the SUCCINCT WINNING STRATEGY instance – on each of your turns simply set the next existential variable according to the scheme for satisfying F. Player two must play legally and pick values for each universal variable in sequence because the moves circuit M does not permit illegal moves. Thus, the final position arrived at will be one in which all the variables are picked and F is satisfied, which means W or L is true (depending on whether the next move would belong to player one or two) and player one wins the game.

Similarly, if we have a winning strategy for our game, then we have a strategy for picking the existential variables of F which guarantees that the formula is satisfied regardless of how the universal variables are set. $\square$

4.4 Predictable Opponent

Now we consider a variant of the winning strategy problem where player one can predict player two's moves with complete certainty and in polynomial time. In other words, player one has access to his opponent's strategy function.

Lemma 5. Succinct Winning Strategy *where player one knows player two's strategy is PSPACE-complete.*

We prove this by showing that Succinct Winning Strategy with access to player two's strategy function is equivalent to Succinct Directed Reachability (A.35), which is complete for PSPACE [54].

> Succinct Directed Reachability
> *Input*: A circuit E describing a directed graph which takes in two positions as bit strings p_1, p_2 and outputs true if there is an edge from p_1 to p_2 in the graph, a starting position s represented as a bitstring, a destination vertex t.
> *Question*: Is it possible to reach t from s in the graph described by E?

Now we begin the proof.

Proof. First, we reduce from Succinct Winning Strategy with access to player two's strategy function to Succinct Directed Reachability, which shows that the former is in PSPACE. Intuitively, we use the fact that we know which out-edge player two will choose from universal nodes to contract each vertex representing one of the positions he may move from into the vertex representing the position he would choose to move to. This eliminates player two's vertices from the game graph, which leaves player one to solve an instance of directed reachability.

We construct an instance of Succinct Directed Reachability from an instance of Succinct Winning Strategy as follows: we set s equal to p_1. For t we create a new vertex and set t equal to it. This leaves us with only the edge circuit E left to define.

Let s_2 be the opponent's strategy function. For E we create a modification of circuit M which checks for one of two cases: if v_2 is t, then player one must have won the game so either v_1 is in W or $s_2(v_1)$ is in L. Otherwise, we return $M(v_1, s_2^{-1}(v_2))$. Formally:

$$E \quad = \quad (v_2 = t \wedge (W(v_1) \vee L(s_2(v_1)))) \vee (v_2 \neq t \wedge M(v_1, s_2^{-1}(v_2)))$$

If there is a path from s to t in G, then player one has a winning strategy in the original game; he can simply follow each move given and his opponent will reply by moving him to the next position in the directed path, right up until he reaches a winning position.

Similarly, if player one has a winning strategy in the game, then he can reach t in the SUCCINCT DIRECTED REACHABILITY with the following procedure: check the winning strategy for a move x from the current position, then check s_2 for the opponent's response to x, then move to the node corresponding to the opponent's response. Since the strategy is winning, repeated iterations of this will eventually lead to a winning position, which corresponds to t.

Now we reduce from SUCCINCT DIRECTED REACHABILITY to SUCCINCT WINNING STRATEGY to show that the latter is PSPACE-hard. Intuitively, we construct a game from the SUCCINCT DIRECTED REACHABILITY instance where player one's goal is to find a path from s to t while player two's only legal move is to pass his turn.

All positions in the game have one additional bit at the front. W is a circuit that checks whether the input is equal to t (with a zero at the front). L and D are circuits that always return false. We set p equal to s with an additional zero at the front. Strategy s_2 simply flips the first bit of the input string (effectively, it is always turning off the first bit).

For M we check the first bit of p_1. If it is a one, then the circuit returns true if and only if p_2 is p_1 with the first bit turned off. If the first bit is a zero, we make sure that the first bit of p_2 is a one and then return E on the two bitstrings without their first bits. The formula representation is as follows:

$$M\left(p^1, p^2\right) = \left(p_0^1 \wedge \neg p_0^2 \wedge \bigwedge_{0<i<|p^1|} \left(p_i^1 = p_i^2\right)\right) \vee \left(\neg p_0^1 \wedge p_0^2 \wedge E\left(p^1_{1\ldots|p_1|-1}, p^2_{1\ldots|p_2|-1}\right)\right)$$

If there is a directed path in the SUCCINCT DIRECTED REACHABILITY instance, then playing out the sequence of moves represented by the directed path is a winning strategy in our game since player two is just flipping the first bit off every turn while

player one actually changes positions. Similarly, if there is a winning strategy in this game then it describes a path from s to t in G since G is the game graph contracted along the edges given by s_2. □

Now we consider a game with both the above restrictions – a game with polynomial termination in which player one knows player two's strategy.

Lemma 6. SUCCINCT WINNING STRATEGY *where the game is guaranteed to have polynomial termination and player one has access to player two's strategy function is NP-complete.*

Proof. To show that the problem is in NP, we present the following nondeterministic algorithm: guess a polynomially-long sequence of moves for player one (each of which is a bitstring of polynomial length) to follow, use player two's strategy function to determine his moves, and then check to see if this strategy leads to a winning position for player one.

We show that this problem is NP-hard by reducing from SATISFIABILITY (A.21). Intuitively, we create a game where first player sets the values for the variables and wins if he finds a satisfying assignment, while player two always passes on his turns.

This reduction uses much the same scheme as we used for reducing QUANTIFIED BOOLEAN FORMULA (A.19) to SUCCINCT WINNING STRATEGY – our bitstrings have a set of bits to represent the variables and then extra bits to count out the quantifier position in unary. We again use the $M \to M'$ transformation to add a partite set bit as described in the previous section. Player two's strategy (s_2) is to turn off the first bit on his turn (thus giving the turn back to first player without effecting any actual change to the position). The win circuit W just checks that the variable-representing portion of the input bits satisfy the formula F ($W(b) = F(b_{1\dots|b|-1})$). Circuit M checks that the first bit is flipped off if it is on, and otherwise that the first bit is flipped on and the remaining bits satisfy the circuit we used for the QUANTIFIED BOOLEAN FORMULA reduction. Thus the formula for M is:

$$M(b,c) = \left(b_0 \wedge \neg c_0 \wedge \bigwedge_{0<i<|b|} (b_i = c_i) \right) \vee \left(\neg b_0 \wedge c_0 \wedge E\left(b_{1\dots|b|-1}, c_{1\dots|c|-1}\right)\right)$$

If the SATISFIABILITY instance is satisfiable, then a winning strategy for this game is to choose values for the variables such that the formula F is satisfied, since satisfying

the formula is the winning condition. Similarly, if there is a winning strategy for the game then there is a satisfying assignment for F since player one's moves in the game are to pick values for variables (while player two effectively does nothing) and the winning condition is to satisfy F. □

Thus, we can quantify the computational benefit of "getting into your opponent's head"; type 3 games – which by Lemma 3 are EXP-complete in general – become PSPACE-complete, and type 2 games – which by Lemma 4 are PSPACE-complete in general – become NP-complete.

Puzzles

In his thesis, Hearn [39] considered (among other things) the distinctions in computational complexity between two-player games, one-player games (puzzles), and zero-player games (which he calls simulations). While we focus on two-player games throughout this thesis, there is a connection between two-player games where first player knows his opponent's strategy, and puzzles. Specifically, by giving first player this knowledge we have in essence removed the second player from the game and turned it into a one-player game, since first player knows how second player will reply to each of his moves.

Therefore, we observe that our results for Succinct Winning Strategy when first player knows his opponent's strategy apply equally to succinctly-represented puzzles. That is, the problem of determining whether a puzzle given by a succinct representation admits a solution is NP-complete if the puzzle has polynomial termination and PSPACE-complete otherwise.

4.5 Complexity of Specific Games

As previous sections have covered games in general, we now review some specific games which have already been subjected to complexity analysis.

Curiously, a slight modification to Winning Strategy makes the problem substantially harder:

> Generalized Geography
> *Input*: A directed graph $G = (V, A)$ and a starting position $s \in V$.
> *Question*: Does player one have a winning strategy in the following game?

> Players alternate choosing a new arc from A. The first arc chosen must have its tail at s and each subsequently chosen arc must have its tail at the vertex that was the head of the previously chosen arc. The first player unable to choose such an arc loses.

For those unfamiliar with it, the game for which this problem is named (Geography) has two players who alternate giving country names where each name must start with whatever letter the preceding name ended with (the first country named is free of this constraint). An example of play might go: France, England, Denmark, Kenya, Afghanistan, and so on. The first player who is unable to name a previously-unnamed country that adheres to the letter constraint loses. You could implement this game as an instance of Generalized Geography (A.12) in a manner similar to how you would construct a game graph: first, create a vertex for each letter. Second, for each country add an edge from the letter the country starts with to the letter the country ends with.

The only significant difference between bipartite Generalized Geography and bipartite Alternating Reachability (A.1) is that in Generalized Geography no edge may be used more than once. Yet while Alternating Reachability and its restriction to bipartite graphs are P-complete, Generalized Geography is PSPACE-complete [64] even when restricted to planar bipartite graphs with no in- or out-degree that exceeds 2 and a maximum total degree of 3 [45]. This variant is historically significant as it is used as the basis for reductions to many PSPACE-complete board games such as Go [45] and Checkers [29]. However, Short Generalized Geography (A.27) with maximum out-degree 2 is in FPT when using the number of turns as a parameter, as we show in Corollary 2 on page 67.

Other Games

In the literature, there are many PSPACE-hardness results for specific games. Most are derived by reduction from Generalized Geography, or a variant of Generalized Geography which is planar, bipartite, no in- or out-degree exceeds 2, and no total degree exceeds 3 [45]. These reductions start from this problem, use an argument about finding a representation in the plane [45, 29], and then show how they can represent edges and vertices with pieces on the game board. This technique has been used for othello [41] as well as for checkers [29] and go [45] with polynomial termination.

Hex is PSPACE-complete for an arbitrary position [59], though a non-constructive proof shows that player one can force a win from the initial position. Go-moku is also PSPACE-complete [58].

On the other hand, when there is no polynomial termination rule, many games are complete for EXP. Checkers [61] and go [62] are two examples. Additionally, chess is also complete for EXP [30].

More recently, Hearn [39] derived results from a problem he calls PLANAR BOUNDED TWO-PLAYER CONSTRAINT LOGIC (PLANAR BOUNDED 2CL). His technique allows for hardness reductions which are noticeably simpler than the GENERALIZED GEOGRAPHY-based reductions. Using this technique, Hearn proved PSPACE-hardness for several additional combinatorial games: Amazons, Konane, and Cross-Purposes.

Game	Complexity	Ref.
Geography	PSPACE-complete	[64]
Go-moku	PSPACE-complete	[58]
Hex	PSPACE-complete	[59]
Checkers (with polynomial termination rule)	PSPACE-complete	[29]
Checkers (without polynomial termination rule)	EXP-complete	[61]
Othello	PSPACE-complete	[41]
Chess	EXP-complete	[30]
Go (with polynomial termination rule)	PSPACE-complete	[45]
Go (without polynomial termination rule)	EXP-complete	[62]
Amazons	PSPACE-complete	[39]
Konane	PSPACE-complete	[39]
Cross-Purposes	PSPACE-complete	[39]

Table 4.2: Existing Complexity Results for Generalized Games

In Table 4.2 we see that results for specific games stick very much to the patterns established by the general results; games with polynomial termination are PSPACE-complete while those without are complete for EXP.

4.6 Summary

In this chapter we have explored the computational complexity of determining whether winning strategies exist in three types of games. Games of the first type are games where the number of positions is polynomial (in the size of the input) and are P-complete. Games of the second type have polynomial termination and are PSPACE-

complete. Games of the third type are games which do not fall into the first two categories, and are EXP-complete.

Further, we have shown that if we give player one knows his opponent's strategy, games of the third type are PSPACE-complete and games of the second type are NP-complete. We could doubtless apply the same technique to games of the first type and find that they are NL-complete under this condition. Giving player one access to his opponent's strategy function effectively removes the need for using universal quantifiers and thus the need for an alternating Turing machine, enabling us to solve these problems with nondeterministic Turing machines. We also noted that game problems where first player knows second player's strategy function were equivalent to puzzle problems – the opposing strategy function for a puzzle simply maps each position to itself – and thus the problem of determining whether a puzzle given by a succinct representation is solvable is PSPACE-complete for puzzles in general and NP-complete for puzzles with polynomial termination.

While the majority of this chapter is a review of existing results, the hardness results for games with polynomial termination are new.

Lastly, we revisit the game brainstones from Section 3.3. The techniques we have discussed in this chapter lead us to ask what is the complexity of the winning strategy problem for a generalized brainstones game. This, however, leads us to another issue: what is an appropriate generalization of brainstones? It is not clear to us how to generalize the pattern by which symbols are laid out on the board. This issue can be circumvented by including the symbols as part of the input, but this means considering game boards where the symbols are laid out with no structure whatsoever.

Chapter 5

Short Games

In [3], Abrahamson, Downey, and Fellows introduced the idea of *short* or k-move or games. This idea adds a slight modification to the usual question of whether the player to move has a winning strategy – does the player has a winning strategy *which takes at most k moves to execute*? The short version of the winning strategy problem can be seen as natural in its own right; there are many chess problems where the goal is to identify a checkmate in x moves (where x is most often 1 or 2). This problem is also implied in the basic artificial intelligence approach to combinatorial games of cutting the minimax algorithm off at some convenient depth. It could also be viewed as an optimization take on the usual winning strategy problem – finding a winning strategy is fine, but perhaps you want to find the quickest path to victory. This gives us a common parameter to work with across all games: the number of turns necessary to execute the winning strategy in full.

In this chapter we study the complexity of various short game problems. We begin in Section 5.1 by considering Short Succinct Winning Strategy, a short version of the Succinct Winning Strategy problem we studied in Chapter 4. Then in Section 5.2 we introduce a list of techniques which we use throughout the remainder of this thesis for proving the parameterized complexity – both membership and hardness – of various game problems. The rest of the chapter is devoted to proving membership results for a number of short game problems: Short Alternating Hitting Set, Short Generalized Hex, Short Generalized Geography, Endgame Generalized Checkers, Endgame Generalized Othello, and Short Generalized Othello.

5.1 Short Succinct Winning Strategy

We now define Short Succinct Winning Strategy (A.32), adding the concept of a k-move game to the Succinct Winning Strategy (A.36) problem we considered in Chapter 4.

> Short Succinct Winning Strategy
> *Input*: a positive integer t, a circuit M which takes in two positions as bit strings p_1, p_2 and outputs true if moving from p_1 to p_2 is legal, a starting position p represented as a bitstring, circuits W, L, and D which take as input a bitstring representing some position q and return true if the player to move from q has won, lost, or drawn the game respectively.
> *Question*: starting from position p, does player one have a winning strategy that takes at most t turns to execute in the two-player game described by circuits M, W, L, and D?
> *Parameter*: t.

After our investigation into the classical complexity of Succinct Winning Strategy, it seems natural to begin our investigation of the parameterized complexity of short games with a look at the parameterized complexity of Short Succinct Winning Strategy. We can immediately show that Short Succinct Winning Strategy is in XP for games in which the maximum number of moves from any position is polynomially-bounded:

Lemma 7. *If the number of moves available from any position in some game $\mathbb{G}$ can be bounded by a function $p(n)$ which is polynomial in the size of the input, and the amount of work necessary to generate all the moves available from a position can be bounded by some polynomial $q(n)$, then* Short Succinct Winning Strategy *is in XP for $\mathbb{G}$.*

Proof. If the polynomial bound on the number of moves from any position is equal to $p(n)$, then the number of nodes in the game graph is at most $p(n)^k$ where k is the number of turns. Since the amount of processing necessary at each position is bounded by a polynomial $q(n)$, we can run a modified minimax algorithm in time $O(p(n)^k \cdot q(n))$. □

A natural question at this point is how often games admit such polynomial bounds, as they may initially appear to be quite restrictive. We observe that, with the exception

of checkers, they apply to every game considered in this thesis. We also note that while the game graph may contain cycles, our modified minimax algorithm cuts off exploration when the depth reaches k.

For many games the number of positions on the board is a natural and hard upper bound on the number of moves available from any position – tic-tac-toe, gomoku, connect four, hex, othello, and go immediately fall into this category. Things are not quite to simple for chess, but as we show in Chapter 6 every move ultimately corresponds to moving a piece from one square to another and this is the board size squared.

Generating this polynomial number of positions is also polynomial for nearly all the games we consider. For many of the games we consider, a turn consists either of placing a single piece in an unoccupied square (tic-tac-toe, gomoku, connect four, hex, othello, go). Generating all possibles moves from a position in these two kinds of games is simply a matter of checking every square against the rules, which is $O(n \cdot r(n))$ ($r(n)$ is the time is takes to evaluate circuit M, which is polynomial). Chess positions can be generated by trying to move each individual piece to every other square on the board ($O(n^2 \cdot r(n))$).

Two notable exceptions to this polynomial bound are checkers and pursuit-evasion. Checkers is problematic because of the rule which allows a player to perform several captures in sequence, and in pursuit-evasion the problem is that k cops running around a graph with n vertices can result in $\binom{n}{k} = O(n^k)$ possible moves.

This gives us a membership result. Next, let us consider the hardness of Short Succinct Winning Strategy.

Lemma 8. Short Succinct Winning Strategy *is AW[P]-hard.*

Proof. We show that Short Succinct Winning Strategy is AW[P]-hard by reduction from Parameterized QCSAT (A.18), which is known to be AW[P]-complete [21].

We assume that each set of variables in the Parameterized QCSAT instance was of equal size v and weight w (if not, we can easily rectify this by adding padding variables). We construct our legal moves circuit M from scratch. It takes in positions represented as bitstrings of length $|S_1| \cdot r$. A move from p^1 to p^2 is legal if and only if the bitstring p^2 is a copy of p^1 with k_x more bits flipped on in the section representing the next move. Specifically:

$$
\begin{aligned}
M \quad \leftarrow \quad & \bigwedge_{0 \le i < t \cdot v} (p_i^1 = p_i^2 \vee \neg p^1_{(t+1)\cdot v - (\lfloor \frac{i}{v} \rfloor + 1)}) \\
& \wedge \bigwedge_{1 \le i < t} (p^1_{v \cdot t - i} = p^2_{v \cdot t - (i+1)}) \wedge p^2_{v \cdot t - 1} \wedge S_{w-1,2v} \wedge \neg S_{w,2v}
\end{aligned}
$$

The first part of the formula checks that for all bits $i < t \cdot v$, either the ith bit is unchanged ($p_i^1 = p_i^2$), or the simulated game has not yet decided turn i ($\neg p^1_{(t+1)\cdot v - (\lfloor \frac{i}{v} \rfloor + 1)}$). The second part checks that the turn bits of p_2 are one turn advanced from p_1 ($\bigwedge_{1 \le i < t} (p^1_{v \cdot t - i} = p^2_{v \cdot t - (i+1)})$), and that the bit indicating that the first turn has been completed is on in p_2 ($p^2_{v \cdot t - 1}$).

The last part of the formula employs S, a sorting network, which sorts the bits of b^1 and b^2 in descending order. Thus, checking that $S_{w-1,2v}$ is true and $S_{w,2v}$ is not tells us whether the number of bits in the string is exactly w (e.g., that the weight constraint is satisfied). We construct this sorting network S as follows:

$$
S_{i,d} \leftarrow \begin{cases}
d = 0: & \bigvee_{0 \le j < t} (b^2_{v \cdot (t+1) - (j+1)} \wedge \neg b^1_{v \cdot (t+1) - (j+1)} \wedge b^2_{v \cdot j + i}) \\
d > 0, d + i \text{ is even}: & S_{i-1,d-1} \wedge S_{i,d-1} \\
d > 0, d + i \text{ is odd}: & S_{i,d-1} \vee S_{i+1,d-1}
\end{cases}
$$

$S_{i,d}$ is the value in position i after iteration d. In odd-numbered iterations, bits are paired off (0-1, 2-3, 4-5, etc.) and the bits are swapped if the greater-numbered bit is on while the lesser-numbered bit is off. The same takes place in even-numbered iterations, except that the pairings are shifted (1-2, 3-4, 5-6, etc.).

This sorting network arranges bits into descending order, at which point the weight is at least w if $S_{w-1,2v}$ is on, and no more than w if $S_{w,2v}$ is off, so we check both to ensure that the weight is exactly w.

If the number of quantifiers is even, then we use the original circuit C as W. Otherwise, we use C as L. The other win/loss circuit is C with a negated output, and D is a circuit that always returns false. The initial position is the all-zero string.

The game being played here is the choosing of bits, and first player wins if the final configuration of bits satisfies the original circuit C. Thus, if there is a winning strategy for one of these problems there is a winning strategy for the other as well.

This reduction takes polynomial time; circuit M is of polynomial size and the other inputs to Short Succinct Winning Strategy are copied directly from the

inputs to Parameterized QCSAT. □

Unfortunately, a membership reduction remains elusive. The biggest obstacle is that the Short Succinct Winning Strategy problem has no notion of weight control. In the hardness reduction we could enforce that aspect of Parameterized QCSAT with the move circuit, but for a membership reduction there would be no notion of weight control to feed into the weight parameters. In fact, as we see later, this weight control can be quite restrictive and sometimes the trick to finding an AW[*]-membership reduction for a game is coming up with a way to represent moves which requires at most a parameterized number of bits (out of a polynomial number of variables). This is certainly the case for othello and chess.

Coincidentally, the bound imposed by the Hamming weight – at most a parameterized number of some polynomial number of variables – effectively leaves us with a hard bound of n^k possible bitstrings for each turn (n is the number of variables per quantifier and k is the Hamming weight). Given the argument we made earlier about games presenting polynomial numbers of moves at any given position, it seems plausible that games for which the "natural" board representation lacks weight control (in other words, where the number of pieces on the board cannot be bounded strictly as a function of the turns parameter) we can use artificial bitstrings where we actually record moves and then reconstruct board positions from move sequences. We use such schemes when we work with othello and checkers later in this chapter, and again for chess in Chapter 6.

5.2 Techniques for Proving Complexity of Short Games

First, we review a quick method for showing that short variants of hard games are themselves hard. Then we review the three primary techniques we use throughout the remainder of this thesis for proving parameterized complexity results involving short games: two are reductions to show AW[*] membership and hardness, and the last is something of an FPT-by-sledgehammer argument.

5.2.1 Classical Hardness for Short Games

When we have a winning strategy problem X, the problem SHORT X is typically defined in a manner identical to X except with the addition of input k and a proviso that the winning strategy found must take at most k turns to execute. We exploit this pattern to produce a generic reduction strategy to show hardness for many short game problems, using the following lemma:

Lemma 9. *If combinatorial game* $\mathbb{G}$ *has polynomial termination, and* GENERALIZED $\mathbb{G}$ *– the problem of determining whether a winning strategy exists for game* $\mathbb{G}$ *– is hard for class* $\Gamma \supseteq NP$*, then* SHORT GENERALIZED $\mathbb{G}$ *– the problem of determining whether a k-move winning strategy exists for* $\mathbb{G}$ *– is also hard for* Γ*.*

Proof. Since $\mathbb{G}$ has polynomial termination, there is some polynomial bound $p(n)$ on the maximum length (given in the number of moves played) of a game. Thus, given an instance Φ of the GENERALIZED $\mathbb{G}$, we can create an instance $(\Phi, p(n))$ of SHORT GENERALIZED $\mathbb{G}$ for which a winning strategy exists if and only if there was a winning strategy in the original problem. Clearly, if you have a $p(n)$-move winning strategy then you have a winning strategy, and vice-versa if there was a winning strategy for the original problem then it must be a $p(n)$-move winning strategy since a game cannot last longer than $p(n)$ moves. □

5.2.2 Reduction into AW[*]

We use two AW[*]-complete problems to prove membership and hardness for AW[*]. To show membership we reduce to PARAMETERIZED QBFSAT_t (A.17). To show hardness we reduce from a highly restricted variant of the problem, which we call UNITARY MONOTONE PARAMETERIZED QBFSAT_2.

UNITARY PARAMETERIZED QBFSAT_t is the restriction of PARAMETERIZED QBFSAT_t to instances where all k_i are set to 1, and is also AW[*]-complete [4, 3]. UNITARY MONOTONE PARAMETERIZED QBFSAT_t is UNITARY PARAMETERIZED QBFSAT_t with the additional restriction that no negations may appear in the boolean formula F. We show that this problem is AW[*]-complete with the following proof.

Lemma 10. UNITARY MONOTONE PARAMETERIZED QBFSAT_t *is AW[*]-complete.*

Proof. Since every instance of UNITARY MONOTONE PARAMETERIZED QBFSAT$_t$ is also an instance of UNITARY PARAMETERIZED QBFSAT$_t$, the former is in AW[*] because the latter is.

One can also transform any instance of UNITARY PARAMETERIZED QBFSAT$_t$ into an instance of UNITARY MONOTONE PARAMETERIZED QBFSAT$_{t+1}$. To do this, find each place in F where a variable (v_i) is negated. Remove the negation, find the set S_j to which v_i belongs, and replace this instance of v_i in the formula with the conjunction $(\bigvee_{x \in S_j, x \neq v_i} x)$. This conjunction contains no negations and is true if and only if v_i is false because exactly one variable in S_j is true and thus v_i is false if and only if some other variable in S_j is true. Repeat this process until all negations are removed. This process takes polynomial time, and at the end the formula contains no negations and is true if and only if the original formula was true. The new formula may have increased the alternating conjunction/disjunction depth by one since we replaced negated variables with bracketed conjunctions, however this is why we chose the target problem to have depth $t + 1$ □

Finally, UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ is the restriction of UNITARY MONOTONE PARAMETERIZED QBFSAT$_t$ to inputs where F is in conjunctive normal form (CNF). One might observe that the reduction we used to show UNITARY MONOTONE PARAMETERIZED QBFSAT$_t$ is AW[*]-hard does not necessarily cover the $t = 2$ case since the target problem has maximum depth $t+1$ and the $t = 1$ case is not AW[*]-hard. However, for the $t = 2$ case we are adding conjunctions within conjunctions and thus there is no additional alternation between conjunctions and disjunctions (i.e., t remains at two).

5.2.3 FPT by Bounded Game Graph

We typically begin a proof that a short winning strategy problem is in FPT by showing that the size of the game graph can be bounded by a function of the turns parameter t, independent of the board size n. Once this is established, a minimax-type algorithm can find a winning strategy in polynomial time so long as we can establish that the amount of work performed at each node of the game graph is at most polynomial.

5.3 Short Alternating Hitting Set

The first specific game we consider in this chapter is SHORT ALTERNATING HITTING SET (A.23). This problem was mentioned as a candidate for AW[*]-completeness by Downey and Fellows in [21]. The original problem, ALTERNATING HITTING SET, is PSPACE-complete [64]. The game also exhibits polynomial termination and thus by Lemma 9 SHORT ALTERNATING HITTING SET is PSPACE-hard. As a first step toward verifying Downey and Fellow's conjecture, we show here that the problem is in AW[*]. It remains an open question whether SAHS is hard for AW[*].

> SHORT ALTERNATING HITTING SET (SAHS)
> *Input*: A set of elements E, a set S of subsets over E, integer k.
> *Question*: Does player one have a strategy to force a win within the next k moves in the following game?
> *Game Rules*: Players one and two alternate choosing (unchosen) elements of E, which they add S'. S' starts the game empty. The player who chooses the last element such that S' becomes a hitting set for S wins.
> *Parameter*: k

Note that a *hitting set* for S is a set which shares at least one element in common with each set of S. In other words, R is a hitting set for S if for every set $T \in S$, $R \cap T \neq \emptyset$.

We show that SHORT ALTERNATING HITTING SET is in AW[*] by reducing the problem to PARAMETERIZED QBFSAT$_t$ (A.17). That is, given instance (E, S, k) of SAHS we transform it into instance $(r, S_1 \ldots S_r, k_1 \ldots k_r, F)$ of PARAMETERIZED QBFSAT$_t$.

For our reduction we create a set of boolean variables $V_{e,t}$ over all $e \in E$ and all $t \in \mathbb{N} : 1 \leq t \leq k$. $V_{e,t}$ is true if and only if element e was added to S' on or before turn t.

Now we define the sets $S_j = \{V_{e,j} : e \in E\}$. Intuitively, S_j tells us which elements are in S' on a given turn j. For the weights, we set $k_j = j$ so that the simulated S' must contain one element after the first turn, two elements after the second turn, and so on. We use the formula F to enforce the requirement that the set retain all the elements added. We now describe F in a top-down manner.

$$F \leftarrow R_{ODD} \wedge (W \vee \neg R_{EVEN})$$

Intuitively, our idea is for formula F to be true if and only if the rules are followed on the odd turns (player one's turns) and either player one wins or rules are broken on an even turn (that is, player two cheats).

$$R_X \leftarrow \bigwedge_{\substack{t \in \mathbb{N}^X : 1 \leq t \leq k \\ e \in E}} (V_{e,t-1} \rightarrow V_{e,t}) \text{ for } X \in \{ODD, EVEN\}$$

We use $\mathbb{N}^X$ where X is ODD or $EVEN$ to denote the sets of odd and even natural numbers (respectively).

In regards to checking that each player moves legally, we only explicitly enforce one rule: any element which was in S' last turn must also be in S' this turn. Since the weight k_j of each set grows by one each turn this means that if they have otherwise followed the rules, each player must add exactly one new element each turn as per the rules.

$$W \leftarrow \bigvee_{\substack{t \in \mathbb{N}^{ODD} \\ 1 \leq t \leq k}} H_t \wedge \neg H_{t-1}$$

The winning condition is that on an odd-numbered turn (player one's turn) there is a hitting set, and there was no hitting set on the preceding turn.

$$H_t \leftarrow \bigwedge_{Q \in S} \bigvee_{q \in Q} V_{q,t}$$

This is the general formula for a hitting set; in every set Q in S, at least one element of Q is in S' ($V_{e,t}$ in the simulated instance).

Lemma 11. *Formula R_{EVEN} is false if and only if the V variables are set in a manner which implies that player two cheats.*

Proof. As stated earlier, the only rules regarding a legal turn are that a player must choose one new element to add to their set (while retaining all the previous elements chosen). The weight for turn t (k_t) is one higher than the weight for turn $t-1$ (k_{t-1}), so if all the previously picked elements are retained then there is only room to choose one additional element on turn t. Thus, to ensure that the turn was conducted properly we only need to ensure that all the elements from the previous turn were

retained. To this end, if some element e was not retained, then $V_{e,t-1} \rightarrow V_{e,t}$ is false and thus R_{EVEN} is false.

On the other hand, this implication is the only operator in the expression which operates on literals. Thus, if R_{EVEN} is false then one of these implications must be false. Since t is always even, the set of implications being evaluated is exactly the set which corresponds to player two's moves. □

Lemma 12. *Formula R_{ODD} is false if and only if the V variables are set in a manner which represents player one cheating.*

Proof. This is identical to the previous proof, except that t is always odd and thus it is only player one's moves which are evaluated in R_{ODD}. □

Lemma 13. *Formula W is true if and only if player one meets the winning condition.*

Proof. If on some turn the chosen elements form a hitting set for the original SAHS instance (E, S, k), then H_t is true. If H_t is true and H_{t-1} is not, then the hitting set was formed on turn t. If t is odd, then this happened on player one's turn and thus he has met the winning condition.

On the other hand, if W is true then a hitting set was formed on an odd-numbered turn where there was not a hitting set on the previous turn. Thus, the hitting set was formed on player one's turn and thus he has met the winning condition. □

Note that W does not care if either player cheated; the task of checking for illegal play is left to R_{ODD} and R_{EVEN}.

Theorem 5. SHORT ALTERNATING HITTING SET *is in AW[*].*

Proof. By the reduction given above, we reduce from SHORT ALTERNATING HITTING SET to PARAMETERIZED QBFSAT$_t$.

We first show that if player one has a winning strategy in the SAHS game, then the PARAMETERIZED QBFSAT$_t$ instance is satisfiable: set the existential variables corresponding to the choices player one would make under his winning strategy. If the universal variables are chosen in such a way that the simulated player two follows the rules, player one can win the simulated game with his winning strategy in which case F is satisfied because W and R_{ODD} are true. If the universal variables are instead set such that player two cheats, R_{EVEN} is false while R_{ODD} is still true, and thus F is true in this case as well.

On the other hand, if player one does not have a winning strategy, then universal variables can be picked which correspond to legal moves by player two which prevent player one from winning and thus satisfy R_{EVEN} while keeping W from being satisfied. The result is that F evaluates to false. If player one cheats, player two can still move legally, making R_{EVEN} true while R_{ODD} is false and thus F evaluates to false in this case as well.

The entire reduction takes polynomial time to construct. Note that R_X is of size $O(k|E|)$ (we generate one clause for each element on each turn) and H_t is of size $O(|S| \cdot |E|)$ (we generate one literal for every element of each subset). From this we can derive that W is of size $O(k \cdot |S| \cdot |E|)$ (we check H_t twice for each odd-numbered turn) and thus for F itself is also $O(k \cdot |S| \cdot |E|)$ (we check R_{EVEN}, R_{ODD}, and W). □

5.4 Hex

Besides Short Alternating Hitting Set (A.23), Short Generalized Hex (A.28) was the second of the two problems Downey and Fellows proposed as candidates for AW[*]-completeness [21]. This problem is based on a generalization of the board game Hex to arbitrary graphs, which is also known as the Shannon switching game on vertices [23]. Determining whether a winning strategy exists (without the short constraint) for the Shannon switching game on vertices in an arbitrary graph is known to be PSPACE-complete [23]. Hex has polynomial termination, so by Lemma 9 Short Generalized Hex is PSPACE-hard as well.

Short Generalized Hex
Input: A graph $G = (V, E)$, vertices $\alpha, \omega \in V$, integer ℓ.
Question: Does player one have a ℓ-turn winning strategy in the following game?
Game Rules: Two players alternating place marked pebbles on unpebbled vertices. Player one wins if he can create a path from α to ω with his pebbles. Player two wins if player one is unable to form such a path.
Parameter: ℓ

We now prove part of Downey and Fellow's conjecture, namely that Short Generalized Hex is in AW[*]. The most significant technical hurdle this proof must overcome is that the winning condition of the game is implicitly an instance of the Undirected Reachability (A.38) problem and thus cannot be expressed strictly

with a first-order logic formula. The solution to this problem is to use the quantifiers to find the path by making a nondeterministic guess, but we must be careful to do so in a parameterized manner.

Theorem 6. SHORT GENERALIZED HEX *is in AW[*].*

Proof. We prove this by reducing SHORT GENERALIZED HEX to PARAMETERIZED QBFSAT$_5$.

This reduction is structured in much the same manner as our SHORT ALTERNATING HITTING SET membership reduction in Section 5.3. First, we create a set of variables to represent the board each turn:

$$x_{v,t} \text{ is true if and only if vertex } v \text{ is chosen on turn } t.$$

Next, we set up the first k quantifiers to represent the next k moves of the game:

$$1 \leq t \leq \ell : S_t \leftarrow \{x_{v,t} : v \in V\} \text{ and } k_i \leftarrow 1.$$

In other words, on each turn exactly one vertex is picked. The one and only rule regarding what constitutes a legal move is that you must pick one vertex which was not previously picked. The weight forces the player to choose exactly one vertex, and we can check that it is a new vertex by checking that for every vertex v and every previous turn i that either v wasn't picked on turn i or it wasn't picked this turn:

$$R_t \leftarrow \bigwedge_{\substack{v \in V \\ 1 \leq i \leq t}} (\neg x_{v,i} \vee \neg x_{v,t})$$

It is not possible to check the winning condition using only the formula (unless FO = L), so we use the quantifiers to help out by making a parameterized guess. That is, we use an existential quantifier to guess (in a non-deterministic manner) a path which satisfies the winning condition. To do this, we first define more variables:

$$e_{v,w} : v, w \in V \text{ and } f_i : 1 \leq i \leq k$$

If k is even, then we add one more existential quantifier ($\delta = 1$). If k is odd we add a universal quantifier as padding and then add the existential quantifier which we want ($\delta = 2$):

$$S_{\ell + \ delta} \leftarrow \{e_{v,w} : (v, w) \in E\} \cup \{f_i : 1 \leq i \leq \ell + \delta\} \text{ and } k_{\ell+\delta} \leftarrow \ell + 1$$

This additional existential quantifier chooses up to $\ell + 1$ edges in G. The winning condition checks that this path connects α to ω using only vertices chosen by player one.

$$\begin{aligned} W \leftarrow & \bigwedge_{v,w \in V} (\neg e_{v,w} \vee ((\bigwedge_{\substack{1 \le i \le \ell \\ i \in \mathbb{Z}^{ODD}}} x_{v,i}) \wedge (\bigwedge_{\substack{1 \le i \le \ell \\ i \in \mathbb{Z}^{ODD}}} x_{w,i}))) \\ & \wedge \bigwedge_{(v,w) \in E} (\neg e_{v,w} \vee (\bigvee_{u \in N(w)} (e_{w,u} \wedge \bigwedge_{\substack{y \in N(w) \\ y \neq u}} \neg E_{w,y}))) \\ & \wedge (\bigvee_{v \in N(s)} e_{\alpha,v}) \wedge (\bigvee_{v \in N(t)} e_{v,\omega}) \end{aligned}$$

In short, the edges chosen are such that α has in-degree 0 and out-degree 1, ω has in-degree 1 and out-degree 0, and all other vertices have either in-degree and out-degree 0 or in-degree and out-degree 1 with the incoming and outgoing edges being adjacent to different vertices. This means that for W to be satisfied there to be a path from α which cannot loop back on itself (as this would create a vertex of in-degree 2) and can only terminate at ω (since any other vertex with in-degree 1 must also have out-degree 1). The selected edges may include some self-contained cycles, but the existence of such cycles alone cannot satisfy W – a path connecting α to ω must still exist.

The last thing we require before we can construct F is some additional machinery to check whether player one or two cheated:

$$\begin{aligned} \overline{R_t} &\leftarrow \bigvee_{\substack{v \in V \\ 1 \le i \le t}} (x_{v,i} \wedge x_{v,t}) \\ R_{ODD} &\leftarrow \bigwedge_{1 \le i \le \frac{\ell}{2}} R_{2i-1} \\ R_{EVEN} &\leftarrow \bigvee_{1 \le i \le \frac{\ell}{2}} \overline{R_{2i}} \end{aligned}$$

The formula $\overline{R_t}$ is the negation of R_t. R_{ODD} checks that the rules were followed on all of player one's turns while R_{EVEN} checks that the rules were broken on at least one of player two's turns. With these formulas we can define formula F to check that player one followed the rules (R_{ODD}), and either player one won (W) or player two cheated (R_{EVEN}). In other words, F is satisfied if and only if player one wins the game.

$$F \leftarrow R_{ODD} \wedge (W \vee R_{EVEN})$$

Our last observation is that R_{ODD} and R_{EVEN} are both 2-normalized, while W is 3-normalized, making F 5-normalized and thus a legal formula input to PARAMETERIZED QBFSAT$_5$. □

5.5 Geography

We first discussed GENERALIZED GEOGRAPHY (A.12) in Section 4.5, where we noted that determining whether the game admits a winning strategy is PSPACE-complete [64]. Here we consider a short version of the problem.

> SHORT GENERALIZED GEOGRAPHY
> *Input*: A directed graph $G = (V, E)$, a starting position $s \in V$, and an integer t.
> *Question*: Does player one have a winning strategy that takes at most t turns to execute? Players alternate choosing a new arc from E. The first arc chosen must have its tail at s and each subsequently chosen arc must have its tail at the vertex that was the head of the previously chosen arc. The first player unable to choose such an arc loses.

Abrahamson, Downey, and Fellows showed this game to be AW[*]-complete [3]. However, more restricted variants were used to show that a number of combinatorial games are PSPACE-hard (including othello [41], checkers [29], and go [45]). Some of these variants are more restrictive than others, but all include the restriction that the vertices of G have maximum out-degree 2. However, such a restriction allows us to bound the maximum size of the game tree to 2^t, enabling us to run the minimax algorithm in FPT time. Further, any fixed upper bound c on the number of choices available to a player at any time reduces the game graph to a fixed-parameter-tractable size, that is c^t.

Corollary 2. SHORT GENERALIZED GEOGRAPHY *with bounded out-degree is FPT. [66]*

This result, though not particularly complex, is significant in that it has an immediately consequence for the reduction technique used to show classical hardness for most PSPACE-complete board games. Specifically, it tells us that this technique cannot be reused in the parameterized setting.

5.6 Checkers

A typical checkers endgame has very few pieces on the board. We exploit this by using the number of pieces left on the board as a parameter. This does mean that we are essentially ignoring endgames where one player retains a large number of pieces, but we argue that the outcome of such a game is not in doubt since one player has a significant material advantage.

> ENDGAME GENERALIZED CHECKERS
> *Input*: A checkers position with k pieces on the board, positive integer t.
> *Question*: Does first player have a t-turn winning strategy from the given position in the following game?
> *Game Rules*: As per checkers.
> *Parameters*: k, t

We show that this problem is in FPT.

Lemma 14. ENDGAME GENERALIZED CHECKERS *is in FPT.*

Proof. Observe that there are at most four directions for each piece to move in. It may be possible to extend a move by jumping additional pieces, but the total number of moves available remains parameterized; whichever of the k pieces you choose to move, that piece cannot jump more than k other pieces, for a maximum of $k \cdot 4^k$ possible sequences of jumps. In some variants you may also choose to terminate a sequence of jumps, so you in fact have a fifth move available (stop moving). Therefore on any turn there are at most $k \cdot 5^k$ possible moves available to the player. Thus we can bound the size of the game graph with $(k \cdot 5^k)^t$. □

5.7 Othello

In this section we consider two different othello problems: ENDGAME GENERALIZED OTHELLO (A.10), in which the number of empty squares left on the board is the parameter, and SHORT GENERALIZED OTHELLO (A.29), in which we explore the k-move-game parameterization more typically used throughout this thesis. Before delving into the complexity of these problems, let us review the rules of the game.

5.7.1 Rules of Othello

Othello is a two-player game played on an 8-by-8 board. Player one controls the black pieces, while player two controls the white. All othello pieces are made such that one side is white and the other is black, so that when control of a piece changes the piece can simply be flipped (rather than having to replace captured pieces). The following rule set is derived from the World Othello Federation's rules [2].

1. A move consists of a player placing a new token of his colour on the board in an unoccupied square in such a way that the new piece is said to *outflank* at least one partial row, column, or diagonal of the opposing colour. To *outflank* means to place a disc on the board such that a row, column, or diagonal of opposing discs (of length at least one) becomes bordered at each end by a disc of your colour. Pieces which are outflanked change over to the outflanking player's control, and are flipped over to their new controller's colour.

2. A player may pass his turn if and only if he has no other available moves.

3. When a position is reached where neither player may move (most frequently because the entire board is full), each player counts the number of pieces of his color on the board and the player with the higher total is declared the winner.

5.7.2 Endgame Othello

The rules of othello state that the game ends only when neither player is capable of moving. Experience playing the game suggests that this most commonly occurs when the entire board has been filled; neither player can move in this circumstance because there are no unoccupied squares for them to place a piece on. This scenario leads us to the following problem:

> ENDGAME GENERALIZED OTHELLO
> *Input*: An $n \times n$ othello board position with exactly k unoccupied squares remaining.
> *Question*: Starting from the given position, does the next player to move have a strategy which guarantees a win within the next k moves?
> *Parameter*: k

We show that this problem is in FPT, once again illustrating the FPT-by-bounded-game-graph technique described in Section 5.2.

Lemma 15. Endgame Generalized Othello *is in FPT.*

Proof. Recall that each move in othello consists of placing one piece in an empty square (or passing if none of the available squares are valid moves). Therefore, the player to move has at most k choices for his next turn. Further, on each successive turn there is one fewer choice than on the preceding turn since the number of open squares decreases by one with each move. Therefore, the number of paths from the given position to any terminal node is at most $k!$ and thus the size of the game tree is $O(k!)$. Further, the minimax algorithm performs polynomial work at each node (evaluating a terminal position and iterating over all valid moves both take time polynomial in the size of the board), which means that it completes in FPT time. □

Again, experience suggests that filling the board is the most common way to finish a game of othello. In other words, the vast majority of games are resolved as instances of the preceding problem.

The proof above makes use of what is essentially a brute-force algorithm. We are unsure whether the theoretical running time could be significantly improved, but cleverly ordering the sequence in which moves are considered (which is important for heuristic improvements to the minimax algorithm like alpha-beta pruning [60]) may result in a practical performance improvement.

5.7.3 Short Generalized Othello

Next we consider othello in the general case, without any special endgame conditions.

> Short Generalized Othello (SGO)
> *Input*: An $n \times n$ othello board position, integer k.
> *Question*: Starting from the given position, does the next player to move have a strategy to force a win within the next k moves?
> *Parameter*: k

This is the k-move version of the Generalized Othello (A.13) problem. Generalized Othello is known to be PSPACE-complete [41], and the game has polynomial termination, so by Lemma 9 Short Generalized Othello is PSPACE-hard.

It considers a superset of the instances considered by Endgame Othello, these additional instances being ones where k is smaller than the number of open squares on the board. We strongly suspect that this problem is AW[*]-complete.

Conjecture 1. Short Generalized Othello *is AW[*]-complete.*

To this end (and as an illustration of the technique described in Section 5.2) we use the remainder of this section to prove half of our conjecture:

Theorem 7. Short Generalized Othello *is in AW[*].*

We begin by describing the reduction we use to prove this theorem.

Reduction Overview

The basic structure of this reduction is similar to that of the reduction we used in Section 5.3, showing that Short Alternating Hitting Set (A.23) is in AW[*] by reduction to Parameterized QBFSAT$_t$ (A.17), but in this case the details are much more complex.

As before with SAHS, our reduction relies upon creating a method of encoding moves with variables in such a manner that the alternating quantifiers of the Parameterized QBFSAT$_t$-instance simulate the players alternating taking turns and the formula F checks that player one is the winner of the simulated game either by reaching a position where he wins, or because player two cheated.

The Parameterized QBFSAT$_t$ instance A which we construct has the form:

$$A = (r, S_1, \ldots, S_r, k_1, \ldots, k_r, F)$$

5.7.4 Variables

We first define all the boolean variables we use in A. In each definition below, we assume the variable being defined is true if the conditions in its description are met and otherwise it is false.

$x_{a,t}$: a piece was placed in square a on turn t

The most basic element of our encoding is recording which square the new piece is placed in each turn. As we will show later, by using these variables in conjunction with the p span variables defined below we can derive what colour a piece is in each

square on any given turn (formulas $b(a,t)$ and $w(a,t)$, defined after the variables). Note that we reference squares by their coordinates on the board. That is, $a = (i, j)$ where i is the column and j is the row, both indexed from 1.

$$x_{a,t} \leftarrow \text{a piece was placed on square } a \text{ on turn } t.$$

We refer to the set of all squares $x_{a,t}$ on an othello board as *BOARD*. More formally,

$$\begin{aligned} BOARD \leftarrow\ & \{x_{a,t} | a = (i,j) \text{ ,where} \\ & 1 \leq i \leq n, 1 \leq n, 1 \leq t \leq k\}. \end{aligned}$$

$p_{a,m,t,d}$: the board span described by a, m, and d was outflanked on turn t

This encoding records whether any specific segment of a column, row, or diagonal was flipped on a given turn.

$$p_{a,m,t,d} \leftarrow \text{on turn } t\text{, the span between square } a \text{ and the square which is } m \text{ squares in direction } d \text{ from } a \text{ was flipped. Here, } d \in \{\rightarrow, \downarrow, \nearrow, \searrow\}$$

As above, $a = (i, j)$ for integers i and j between 1 AND n. As such, a *span variable* may also be referred to as $p_{(i,j),m,t,d}$.

Note that at most eight of these span variables may be true for any valid move, as there are only eight directions in which a piece can outflank other pieces. This will become important when we discuss the weight inputs for A.

There are at most n^2 values for a, n possible values for m, four possible values for d, and k possible values for t. Therefore, the number of span variables is bounded by $n^3 \cdot 4k$.

We define the set *SPANS* to be the set of all spans over all turns. Specifically, a span is a tuple (a, m, t, d) for which the values have the same meaning as the subscripts of the p variables. Thus,

$$\begin{aligned} SPANS \leftarrow\ & \{a, m, t, d | a = (i,j), \\ & 1 \leq i \leq n, i \leq j \leq n, 1 \leq m \leq n, 1 \leq t \leq k, d \in \{\rightarrow, \downarrow, \nearrow, \searrow\}\} \end{aligned}$$

$q_{y,\gamma,t}$: player y controls γ of the pieces on the board at the end of turn t

$$q_{y,\gamma,t} \leftarrow \text{player } y \text{ has exactly } \gamma \text{ pieces of his colour on the board on turn } t.$$

Here, y is either b or w (signifying black or white – player one or player two), γ is an integer between zero and n^2, and t is an integer between one and k.

c_t and g_t: the player to move passed on turn t

$$\begin{aligned} c_t &\leftarrow \text{on turn } t \text{ the player to move did not place a piece.} \\ g_t &\leftarrow \text{on turn } t \text{ no spans were captured.} \end{aligned}$$

Clearly, under legal play if you do not place a piece you can not capture any spans. Similarly, if you did not capture any spans then you must have not placed a piece since the only legal way to place a piece requires you to capture at least one span. Thus, under legal play either both variables are true or both are false. Collectively, if both variables are false then the player placed a piece normally. On the other hand, if they are both true then the player passed his or her turn.

$b(a,t)$: there is a black piece in square a on turn t

$w(a,t)$: there is a white piece in square a on turn t

On turn $t = 0$, the values $b(a,0)$ and $w(a,0)$ over all $a \in BOARD$ are defined by the initial board position input to our SGO instance. That is, $b(a,0)$ is true if and only if the initial position includes a black piece in square a, and $w(a,0)$ is defined analogously for white pieces. Note that this means that the first move made from the given position is, from the perspective of our formula, turn number 1.

On later turns ($t > 0$), the values of $b(a,t)$ and $w(a,t)$ are derived by formulas. To aid in this derivation, we first define a sub-formula, $h(a,z,t)$, which checks a bitstring z against the history of square a to see if it has indeed been flipped (as indicated by span variables) exactly on those turns Ψ for which bit number Ψ of z is on. The formal definition of $h(a,z,t)$ is as follows:

$$h\left(a,z,t\right)\leftarrow\bigwedge_{1\leq i\leq t}\left(\left(\neg z_i\vee\left(\bigwedge_{\substack{x\in\{1\ldots N\}\\ d\in\{\rightarrow,\downarrow,\nearrow,\searrow\}}}p_{a,x,i,d}\right)\right)\wedge\left(z_i\vee\left(\bigwedge_{\substack{x\in\{1\ldots N\}\\ d\in\{\rightarrow,\downarrow,\nearrow,\searrow\}}}\neg p_{a,x,i,d}\right)\right)\right)$$

In addition to h, we also define $\omega(z)$ to be the Hamming weight of bitstring z. Armed with these helper functions we are now ready to define $b(a,t)$ and $w(a,t)$:

$$\begin{aligned}
b\left(a,t\right) &\leftarrow \left(\left(w\left(a,0\right)\vee\bigvee_{0<2z\leq t}x_{a,2z}\right)\wedge\bigwedge_{\substack{z\in\{0,1\}^t\\ \omega(z)\in ODD}}h\left(a,z,t\right)\right)\\
&\quad\vee\left(\left(b\left(a,0\right)\vee\bigvee_{0<2z-1\leq t}x_{a,2z-1}\right)\wedge\bigwedge_{\substack{z\in\{0,1\}^t\\ \omega(z)\in EVEN}}h\left(a,z,t\right)\right)\\
w\left(a,t\right) &\leftarrow \left(\left(b\left(a,0\right)\vee\bigvee_{0<2z\leq t}x_{a,2z}\right)\wedge\bigwedge_{\substack{z\in\{0,1\}^t\\ \omega(z)\in EVEN}}h\left(a,z,t\right)\right)\\
&\quad\vee\left(\left(w\left(a,0\right)\vee\bigvee_{0<2z-1\leq t}x_{a,2z-1}\right)\wedge\bigwedge_{\substack{z\in\{0,1\}^t\\ \omega(z)\in ODD}}h\left(a,z,t\right)\right)
\end{aligned}$$

Next we prove the correctness of formulas b and w.

Lemma 16. *1. $b(a,t)$ is true if and only if there is a black piece in square a on turn t.*

2. $w(a,t)$ is true if and only if there is a white piece in square a on turn t.

Proof. Let us begin by proving the correctness of $b(a,t)$. Observe that formula $h(a,z,t)$ checks whether the bitstring z describes exactly when a was flipped from the first turn to turn t. As such, the main purpose of the formula for $b(a,t)$ is to check for one of two cases:

1. Square a initially contained a white piece which has since been flipped an odd number of times.

2. Square a initially contained a black piece which has since been flipped an even number of times.

We can check the history of a square a up to turn t with $h(a, z, t)$. Thus, we can check for an odd number of flips by checking bitstrings with an odd Hamming weight and for an even number of flips with even-weighted bitstrings. We check the initial colour of the piece in the square by checking the initial colour ($b(a, t)$ and $w(a, t)$) and by checking if a piece was placed later ($x_{a,t}$, knowing that black pieces are placed on odd-numbered turns and white on even-numbered turns).

The proof for $w(a, t)$ is identical to that for $b(a, t)$, only with odd and even adjustments. $\square$

The Reduction

Now that we have defined all the variables we need to encode the position, we can give values for the instance A which we construct.

Intuitively, each turn requires weight 11: 1 new piece, 2 scores, and up to 8 spans. Because the weight is so cleanly divided, we can simplify the process of checking weights by breaking each turn up across several quantifiers by using filler quantifiers ($S_{7(t-1)+2}$,$S_{7(t-1)+4}$,$S_{7(t-1)+6}$) with filler variables ($f_{1,t}$,$f_{2,t}$,$f_{3,t}$).

$$
\begin{aligned}
S_{7(t-1)+1} &\leftarrow \{x_{a,t} : a \in BOARD\} \cup \{c_t\} \\
S_{7(t-1)+2} &\leftarrow \{f_{1,t}\} \\
S_{7(t-1)+3} &\leftarrow \left\{q_{b,i,t} : 0 \le i \le n^2\right\} \\
S_{7(t-1)+4} &\leftarrow \{f_{2,t}\} \\
S_{7(t-1)+5} &\leftarrow \left\{q_{w,i,t} : 0 \le i \le n^2\right\} \\
S_{7(t-1)+6} &\leftarrow \{f_{3,t}\} \\
S_{7(t-1)+7} &\leftarrow \{p_{i,j,m,t,d} : 1 \le i \le n, 1 \le j \le n, 1 \le m \le n, \\
&\qquad d \in \{\rightarrow, \downarrow, \nearrow, \searrow\}\} \cup \{f_{4,t}, \ldots, f_{10,t}, g_t\} \\
k_{7(t-1)+1} \ldots k_{7(t-1)+6} &\leftarrow 1 \\
k_{7(t-1)+7} &\leftarrow 8
\end{aligned}
$$

Note that we have defined seven quantifiers for a single turn. This means that for A, $r \leftarrow 7k$.

In contrast to other AW[*]-membership reductions in this thesis (for example, the membership reduction we gave for SAHS), we have split up the variables which represent each turn across multiple quantifiers. Doing this enables us to enforce individual weight constraints on each subset of variables; only one x variable can be turned on, exactly one score variable is true for each player, and the combined weight of span variables, filler variables, and g_t is eight. Setting up these constraints up-front saves us from having to worry about invalid weight combinations when we construct formula F to check the result of the game. However, for this to work we must add one more set of variables.

$f_{1,t} \ldots f_{10,t}$: filler variables

$f_{i,t} \leftarrow$ not enough other variables were set to true to meet the weight requirement k_j of the set S_j containing this variable.

The values of these variables carry no direct meaning in terms of the game being represented. Rather, we add them so that they can be set true to meet weight requirements whenever the other variables (which actually have meaning) of a given quantifier set do not have sufficient weight to meet the required Hamming weight for the set. In other words, by adding these variables to a quantifier set, for the rest of the variables in that quantifier set the Hamming weight ceases to be an exact constraint and instead acts as an upper bound. Specific uses of these variables are explained when we define the variable sets and Hamming weights for the Parameterized QBFSAT$_t$ instance.

Completing the Reduction

Now in the case of the span variables p we have seven filler variables $f_{4,t} \ldots f_{10,t}$ which can be set to true to pad the weight of the spans since we do not necessarily need eight spans for every move. That is, if we use fewer than eight spans, filler variables can be turned on to ensure that the weight restriction is met.

At this point we have defined all the values for instance A save one: formula F. The next section is devoted to this formula.

5.7.5 Formula F

We construct formula F to enforce the rules and check that player one wins. We do this in a top-down manner, where we often introduce and use sub-formulas that we need before formally defining them later.

$$F \leftarrow (R_{all} \wedge W) \vee \overline{R_{two}}$$

Intuitively, F is *true* in exactly two cases: if no rules were broken before the game ended (R_{all}) and player one is the winner (W), or if player two broke a rule before the game ended (R_{two}). For our purposes, if a player breaks a rule the game ends and victory is awarded to the other player.

Note that at one point we use the overline notation $\overline{X}$. Functionally, the overline is equivalent to negation – that is, $\overline{X} = \neg X$. However, PARAMETERIZED QBFSAT$_t$ does not allow negation except of literals. Therefore, $\overline{X}$ actually corresponds to a separate formula which is equivalent to $\neg X$, but in which negations have effectively been pushed down to the literals by flipping AND/OR operators as per DeMorgan's law. Several of the formulas defined below are presented with their overlined (negated) versions to provide examples.

W: player one wins the game

Formula W checks two things: that the game is over ($END(t)$), and that player one has more pieces on the board than his opponent. The latter is achieved by checking for every combination of score variables where there is a larger number of black pieces than white.

$$W \leftarrow \bigvee_{1 \le t \le k} \left(END\,(t) \wedge \left(\bigvee_{i \in \mathbb{N}: 0 \le j < i \le n^2} (q_{b,t,i} \wedge q_{w,t,j}) \right) \right)$$

R_{all}: both players moved legally

R_{two}: player two moved legally

We define R_{all} and R_{two} to handle the rules regarding legal moves. R_{all} tests that for all $1 \le t \le k$ either turn t was played legally ($M(t)$) or the game was over ($END(t)$). R_{two} tests whether on each of player two's turns ($2t$, where $1 \le t \le \frac{k}{2}$) either he followed the rules ($M(2t)$) or the game was over. In this case the game can be over

either because neither player could move ($END(2t)$) or because player one cheated on a previous turn ($\bigvee_{1 \le s \le t} \overline{M(2s-1)}$), where $1 \le s \le t$. Both formulas use formula $M(t)$; intuitively, $M(t)$ is true if and only if the change in position from turn $t-1$ to t describes a legal move.

$$
\begin{aligned}
R_{all} &\leftarrow \bigwedge_{1 \le t \le k} \left(M(t) \vee END(t-1)\right) \\
R_{two} &\leftarrow \bigwedge_{1 \le t \le \frac{k}{2}} \left(M(2t) \vee END(2t-1) \vee \bigvee_{1 \le s \le t} \overline{M(2s-1)}\right) \\
\overline{R_{two}} &\leftarrow \bigvee_{1 \le t \le \frac{k}{2}} \left(\overline{M(2t)} \wedge \overline{END(2t-1)} \wedge \bigwedge_{1 \le s \le t} M(2s-1)\right)
\end{aligned}
$$

***END*(t): the game is over on or before turn t**

The formula $END(t)$ checks whether the game is over on turn t. The game ends when neither player is able to move, and therefore when on two consecutive turns the player to move legally passed. Thus, $END(t)$ checks whether two consecutive turns have been passed ($g_{i-1} \wedge g_i$) on or before turn t.

$$
\begin{aligned}
END(t) &\leftarrow \bigvee_{2 \le i \le t} \left(g_{i-1} \wedge g_i\right) \\
\overline{END(t)} &\leftarrow \bigwedge_{2 \le i \le t} \left(\neg g_{i-1} \vee \neg g_i\right)
\end{aligned}
$$

$M(t)$: the move on turn t is legal

$$
\begin{aligned}
M(t) &\leftarrow P(t) \wedge C(t) \wedge U(t) \wedge B(t) \\
\overline{M(t)} &\leftarrow \overline{P(t)} \vee \overline{C(t)} \vee \overline{U(t)} \vee \overline{B(t)}
\end{aligned}
$$

Much like circuit M throughout Chapter 4, formula $M(t)$ is left with the task of checking that the variables set on turn t correspond to a legal move. This means checking that $x_{a,t}$ (the new piece), $p_{i,j,m,t,d}$ (the span variables), $q_{y,i,t}$ (the score variables), and c_t and g_t (the passing variables) are all set in a legal configuration. Each is checked by a separate subformula: $P(t)$ checks the the new piece was placed legally, $C(t)$ checks that the span variables are set correctly, $U(t)$ checks that the score variables are set

correctly, and $B(t)$ checks whether the passing variables were set in a legal manner.

$P(t)$: no piece was placed in an already-occupied square on turn t

$$\begin{aligned} P(t) &\leftarrow \bigwedge_{a \in BOARD} \left(\neg x_{a,t} \vee \left(\overline{w(a,t-1)} \wedge \overline{b(a,t-1)}\right)\right) \\ \overline{P(t)} &\leftarrow \bigvee_{a \in BOARD} \left(x_{a,t} \wedge \left(w(a,t-1) \vee b(a,t-1)\right)\right) \end{aligned}$$

$P(t)$ checks that the new piece on a given turn is placed in an unoccupied square. In other words, the new piece is not on a square that was occupied last turn.

Lemma 17. *$P(t)$ is true if and only if the piece placed on turn t occupies a square which is vacant on turn $t-1$.*

Proof. If on turn t a piece is placed on square a which was already occupied, then either $w(a,t-1)$ or $b(a,t-1)$ is true because the square must have contained either a white or a black piece last turn, and thus $P(t)$ is false. On the other hand, if the square was occupied then both $w(a,t-1)$ and $b(a,t-1)$ are false while $x_{e,t}$ is only true for $e = a$, thus $P(t)$ is false. □

$C(t)$: the span variables for turn t are set correctly

Span variables are checked using $C(t)$, to ensure that they are being set appropriately. That is, if a span variable $p_{a,m,t,d}$ is true, then in the board position described on turn t by the x variables, span (a,m,t,d) is outflanked. On the other hand, if the board position on turn t described by the x variables shows that span (a,m,t,d) is outflanked, then $p_{a,m,t,d}$ is true.
For odd t: $C(t) \leftarrow C'(t,b,w)$.
For even t: $C(t) \leftarrow C'(t,w,b)$.

$$
\begin{aligned}
C'(t,\alpha,\beta) &\leftarrow \bigwedge_{\substack{1\le i,j<N\\ 2\le x\le N-i}} C_{\rightarrow}(i,j,x,t,\alpha,\beta) \wedge \\
&\qquad \bigwedge_{\substack{1\le i,j<N\\ 2\le x\le min(N-j,N-i)}} C_{\searrow}(i,j,x,t,\alpha,\beta) \wedge \\
&\qquad \bigwedge_{\substack{1\le i,j<N\\ 2\le x\le min(N-j,i-1)}} C_{\nearrow}(i,j,x,t,\alpha,\beta) \wedge \\
&\qquad \bigwedge_{\substack{1\le i,j<N\\ 2\le x\le N-j}} C_{\downarrow}(i,j,x,t,\alpha,\beta) \\
C_{\downarrow}(i,j,x,t,\alpha,\beta) &\leftarrow \left(\overline{SC}(i,j,i,j+x,t,\alpha,\beta) \vee p_{(i,j),x,t,C_{\downarrow}}\right) \wedge \\
&\qquad \left(\neg p_{i,j,x,t,C_{\downarrow}} \vee SC(i,j,i,j+x,t,\alpha,\beta)\right) \\
C_{\rightarrow}(i,j,x,t,\alpha,\beta) &\leftarrow \left(\overline{SC}(i,j,i+x,j,t,\alpha,\beta) \vee p_{(i,j),x,t,C_{\rightarrow}}\right) \wedge \\
&\qquad \left(\neg p_{i,j,x,t,C_{\rightarrow}} \vee SC(i,j,i+x,j,t,\alpha,\beta)\right) \\
C_{\searrow}(i,j,x,t,\alpha,\beta) &\leftarrow \left(\overline{SC}(i,j,i+x,j+x,t,\alpha,\beta) \vee p_{(i,j),x,t,C_{\searrow}}\right) \wedge \\
&\qquad \left(\neg p_{i,j,x,t,C_{\searrow}} \vee SC(i,j,i+x,j+x,t,\alpha,\beta)\right) \\
C_{\nearrow}(i,j,x,t,\alpha,\beta) &\leftarrow \left(\overline{SC}(i,j,i+x,j-x,t,\alpha,\beta) \vee p_{(i,j),x,t,C_{\nearrow}}\right) \wedge \\
&\qquad \left(\neg p_{i,j,x,t,C_{\nearrow}} \vee SC(i,j,i+x,j-x,t,\alpha,\beta)\right) \\
SC(i_1,j_1,i_2,j_2,t,\alpha,\beta) &\leftarrow \alpha(i_1,j_1,t) \wedge \alpha(i_2,j_2,t) \wedge \\
&\qquad SC'(i_1,j_1,i_2-i_1,j_2-j_1,t,\beta) \wedge (x_{i_1,j_1,t} \vee x_{i_2,j_2,t}) \\
SC'(i,j,x,y,t,\beta) &\leftarrow \bigwedge_{1\le z<max(x,y)} \beta\left(i+\frac{z\cdot x}{max(x,y)}, j+\frac{z\cdot y}{max(x,y)}, t-1\right)
\end{aligned}
$$

Here, α and β are functions and are called as such in SC and SC'.

As suggested by the symbols employed, $C_{\rightarrow}$, $C_{\downarrow}$, $C_{\searrow}$, and $C_{\nearrow}$ check row, column, and the two types of diagonal spans. In turn, they all employ SC (short for "span check"), which checks the span from square (i_1, j_1) to (i_2, j_2) on turn t.

Lemma 18. *The following two statements are equivalent:*

1. *$C(t)$ is true.*

2. *Each span variable on turn t is true if and only if the associated span is outflanked on turn t.*

Proof. First of all, SC checks that the given span $(i_1, j_1) - (i_2, j_2)$ was outflanked on turn t. It does this by checking that the endpoints are the player-to-move's colour (α), that the span itself is the opposing player's colour SC', and that one of the two endpoints was placed this turn.

Now let start with $C_{\rightarrow}$. This function is true if, for the given span, the span check SC and the span variable p have the same true/false value. If they have opposing values, the one of the two clauses of $C_{\rightarrow}$ is true. Analogous proofs apply for $C_{\downarrow}, C_{\searrow}$, and $C_{\nearrow}$. $C(t)$ checks all of these functions in turn over every board square and every possible span length. Thus, $C(t)$ checks that the span variables p match the state of the board on turn t for each and every span on the board.

Note that on odd-numbered turns, we check for spans of white pieces to be flipped to black, while on even-numbered turns we check for spans of black pieces to be flipped to white. □

$U(t)$: the score variables for turn t are set properly

$U(t)$ verifies that the score variables were updated as appropriate for the span variables which are set true. This includes every possible combination of up to 8 span variables. This means creating $O(n^{16})$ clauses for each turn.

There are $O(n^3)$ spans for each turn since each span consists of a square, one of four directions, and number from 1 to n. In turn, there are $\sum_{i=1}^{8} \binom{n^3}{i}$ ways to choose up to eight spans. This means that $\lambda(t)$ contains $O(n^{24})$ clauses, each of which contains n^2 subclauses pushing our upper bound on the size of the formula to $O(n^{26})$. Though very large, this is still polynomial.

Now we define $U(t)$ formally.

For odd t: $U(t) \leftarrow \kappa(b, w, t)$

For even t: $U(t) \leftarrow \kappa(w, b, t)$

$$
\begin{aligned}
\kappa(\alpha,\beta,t) \;\leftarrow\;& (\neg g_t \wedge \lambda(\alpha,\beta,t)) \\
& \vee \left(g_t \wedge \bigwedge_{1\le i\le n^2} ((\neg q_{b,t-1,i} \vee q_{b,t,i}) \wedge (\neg q_{w,t-1,i} \vee q_{w,t,i})) \right) \\
\lambda(\alpha,\beta,t) \;\leftarrow\;& \bigwedge_{\substack{S\subseteq SPANS \\ 1\le |S|\le 8}} \bigvee_{y\in S} (\neg p_y \vee \\
& \left(\bigwedge_{1\le i\le n^2} \left(\neg q_{\alpha,t-1,i} \vee q_{\alpha,t,i+1+\sum_{z\in S}|z|} \right) \right. \\
& \left. \wedge \bigwedge_{1\le i\le n^2} \left(\neg q_{\beta,t-1,i} \vee q_{\beta,t,i-\sum_{z\in S}|z|} \right) \right))
\end{aligned}
$$

The $|z|$ notation used here returns the length of the span z.

Lemma 19. *$U(t)$ is true if and only if on turn t the values recorded by the score variables changed in accordance with how the numbers of pieces on the board changed.*

Proof. If g_t is true, then the turn was passed and as such we check that the score variables are set to the same values as they were for last turn.

If g_t is not true, then let us assume that it is black's turn to move. Let us denote black's score on the previous turn (turn $t-1$) with the variable i, and white's score that turn with the variable j. Now, on turn t black takes his move, capturing spans $S \subset SPANS$. Given this, on turn t black should have score $i+1+\sum_{z\in S}|z|$ while white should have score $j-\sum_{z\in S}|z|$. If it is instead white's move, the same argument applies, only with white and black swapped. □

$B(t)$: if turn t was passed, the player had no moves available

Finally, we have $B(t)$, which handles the rules regarding passing turns.
For odd t: $B(t) \leftarrow B_1(b,w,t)$
For even t: $B(t) \leftarrow B_1(w,b,t)$

$$
\begin{aligned}
B_1(\alpha, \beta, t) &\leftarrow (\neg g_t \wedge \neg c_t) \vee (f_{4,t} \wedge \ldots \wedge f_{10,t} \wedge g_t \wedge c_t \wedge B_2(\alpha, \beta, t)) \\
B_2(\alpha, \beta, t) &\leftarrow \bigwedge_{\substack{(i,j) \in BOARD \\ 2 \leq x \leq N-2}} B_3(i, j, x, t, \alpha, \beta) \\
B_3(i, j, x, t, \alpha, \beta) &\leftarrow B_4(i, j, i, j + x, t, \alpha, \beta) \wedge B_4(i, j, i + x, j, t, \alpha, \beta) \wedge \\
&\quad\; B_4(i, j, i + x, j + x, t, \alpha, \beta) \wedge B_4(i, j, i + x, j - x, t, \alpha, \beta) \\
B_4(i_1, j_1, i_2, j_2, t, \alpha, \beta) &\leftarrow (\alpha(i_1, j_1, t) \vee \alpha(i_2, j_2, t)) \wedge \\
&\quad\; B_5(i_1, j_1, i_2 - i_1, j_2 - j_1, t, \beta) \wedge \\
&\quad\; ((\neg\alpha(i_1, j_1, t) \wedge \neg\beta(i_1, j_1, t)) \vee \\
&\quad\; (\neg\alpha(i_2, j_2, t) \wedge \neg\beta(i_2, j_2, t))) \\
B_5(i, j, x, y, t, \beta) &\leftarrow \bigwedge_{1 \leq z < max(x,y)} \beta\left(i + \frac{z \cdot x}{max(x, y)}, j + \frac{z \cdot y}{max(x, y)}, t - 1\right)
\end{aligned}
$$

Lemma 20. *$B(t)$ is true if and only if either the player to move did not pass on turn t, or he did pass and had no available moves.*

Proof. If g_t is not true, then $B(t)$ is true since the turn was not passed and thus no checks need to be performed. Otherwise, we check that all the other filler span variables and also every span to make sure that it is not viable.

B_4 checks that span $(i_1, j_1) - (i_2, j_2)$ could not be flanked on the turn t by ensuring that one of the following is true: one of the endpoints was already occupied by the opposing player, one or more of the squares within the span itself did not contain an opposing player's piece (B_5), or both the endpoints were unoccupied.

B_3 checks that for square (i, j), distance x, and turn t, four different spans starting from (i, j) could not have been part of an outflanking maneuver on turn t: the column stretching x squares down from (i, j), the row stretching x squares right from (i, j), the diagonal stretching x squares down and to the right of (i, j) and the diagonal stretching x squares up and to the right of (i, j).

B_2 checks that on turn t no outflanking maneuver was possible anywhere on the board by checking B_3 over every board square and every possible span length. □

Proving that SGO is in AW[*]

Lemma 21. *$M(t)$ is true if and only if turn t was conducted legally.*

Proof. $M(t)$ is true if and only if $P(t)$, $C(t)$, $U(t)$, and $B(t)$ are all true. In short, $M(t)$ is true if and only if on turn t a new piece was placed on an unoccupied square (Lemma 17), for which the span (Lemma 18) and score variables (Lemma 19) were set correctly and if he passed on turn t, he did so legally (Lemma 20). □

Theorem 7. Short Generalized Othello *is in AW[*].*

Proof. We prove this theorem by proving correctness of the reduction above, which in turn shows that we reduce from Short Generalized Othello to Parameterized QBFSAT$_t$. This means that Short Generalized Othello is in AW[*] since Parameterized QBFSAT$_t$ is already known to be [21].

If player one has a winning strategy in the SGO instance, then A is satisfiable: if the existentially-quantified variables are set in correspondence with player one's winning strategy, then the universal variables can either be set to play legally or to cheat. In the former case, formula F is satisfied because the variables set to true correspond to a legally-played game in which player one was victorious, meaning that both W and R_{all} are true. In the latter case, F is satisfied because because player two cheated before player one and thus R_{two} is false.

If A is satisfiable, then player one has a winning strategy in the SGO instance: player one can use the same strategy used for setting $x_{a,t}$ of odd-numbered t to decide how his pieces are placed. Since the Parameterized QBFSAT$_t$ instance is satisfiable, this means that by doing so he will win the game, since that is the condition which satisfies F.

All of the formulas used in this reduction are polynomial in size except for $h(a, z, t)$, which is exponential in the size of the parameter but not the input size. Thus, this reduction takes fixed-parameter tractable time. □

5.8 Summary

In this chapter, we first defined short games, then studied the complexity of short games in general as well as for a number of specific games.

From our work with short games in general, we also showed that short games with polynomial termination are PSPACE-hard, and introduced several techniques for proving membership and hardness of short games.

We also discussed Downey and Fellow's conjecture that AW[*] is the "natural home" for short games. However, when we considered a generalization of short win-

ning strategy problems (SHORT SUCCINCT WINNING STRATEGY (A.32)), we showed that it was actually hard for AW[P]. This does not directly contradict the conjecture, it does suggest that there may be games which are not in AW[*]. We noted that SHORT GENERALIZED CHECKERS (A.25) may be one of them.

In terms of specific games, we have shown that SHORT ALTERNATING HITTING SET (A.23), SHORT GENERALIZED HEX (A.28) and SHORT GENERALIZED OTHELLO (A.29) are in AW[*], and that ENDGAME GENERALIZED OTHELLO (A.10) and ENDGAME GENERALIZED CHECKERS (A.9) are in FPT. We also showed, via Lemma 9 that SHORT ALTERNATING HITTING SET, SHORT GENERALIZED HEX, and SHORT GENERALIZED OTHELLO are all hard for PSPACE.

On the open problems front, we suspect that all the problems for which we showed AW[*] membership in this chapter are also AW[*]-hard. AW[*]-hardness for SHORT ALTERNATING HITTING SET and SHORT GENERALIZED HEX, was already proposed by Downey and Fellows [21]. We ourselves added the conjecture that SHORT GENERALIZED OTHELLO is hard for AW[*] (Conjecture 1).

Finally, our last open problem from this chapter is the membership of SHORT SUCCINCT WINNING STRATEGY (A.32). As we already discussed in Section 5.1, finding weight control poses a significant obstacle to the problem. We suspect that the problem can be overcome by restricting the input to games where no player is ever faced with more than a polynomial number of choices on any given turn, and then using the bitstrings from each turn to represent those moves rather than positions. This, of course, would require us to be able to derive all the data for any given position from nothing but the sequence of moves which lead to that position. Our SHORT GENERALIZED OTHELLO already does this in large part, but the question is how to generalize that approach.

Chapter 6

Short Chess

In this chapter we apply our techniques to classify the complexity of finding a short winning strategy in generalized chess, a problem which we refer to as Short Generalized Chess (A.26). This result was originally published in [68].

6.1 The Game of Short Generalized Chess

Generalized chess is played by two players (white and black) who alternate taking turns moving pieces on a generalized *chessboard*, an $n \times n$ grid of squares.

> Short Generalized Chess
> *Input*: An $(n \times n)$-chessboard position[1], a positive integer k.
> *Parameter*: k
> *Question*: Starting from the given position, does player one (white) have a strategy to force a win within the next k moves?[2]

The following provides a basic outline of the rules of chess. For complete rules, see [1].

1. Short Generalized Chess is played by two players who we shall call white and black, white being player one and black being player two. These roles are determined before the game begins (i.e. via mutual agreement, coin toss, etc.).

[1] A *chessboard position* includes the position of every piece (including captured pieces), the turn number, and a flag for each king and rook indicating whether that piece has moved yet (essential information to castling, Rule 16).

[2] A move in chess usually consists of a player moving one of his pieces, except in the case of a capture, castling (cf. Rule 14 and Rule 16), or promotion, when two pieces are moved.

2. Short Generalized Chess is played with six types of *pieces*: *pawns, rooks, knights, bishops, queens*, and *kings*. Each piece belongs to one of the two players. To distinguish which pieces belong to which player, the pieces belonging to player one and two are coloured white and black respectively. Each player starts the game with n pawns, two rooks, two knights, $n-6$ bishops, one queen, and one king, though pawns may be promoted to other captured pieces later in the game.

3. A pawn may always move one square "forward" (along its column toward the opponent's end of the board), so long as the square it would move to is unoccupied. A pawn cannot capture another piece this way.

4. A pawn may move two spaces forward instead of one if it is moving from its initial position (the second row) and both the squares in front of it are unoccupied.

5. A pawn may move one square diagonally provided that the move still advances the pawn toward the opponent's end of the board and it captures an opposing piece.

6. If a pawn reaches the opponent's end of the board (row n) it may be *promoted*: the piece may be replaced with any captured piece (other than a pawn). The promoted pawn is removed from the board (put in S_0).

7. A rook may move to any square along either the same row or column as its current position, provided that the path of movement along that row or column is not obstructed by another piece.

8. A knight may move in an "L", i.e. two squares along the current row or column and then one square perpendicular to that, and may jump over other pieces in doing so.

9. A bishop may move to any square along either of the diagonals that intersect its current position, so long as the path of movement is clear.

10. A queen may move as a rook or bishop.

11. The king may move to any adjacent square, except in the special case of *castling*. A player may perform a castle to move both his king and a rook on the same

turn provided she has not yet moved either piece, all the squares between them are empty, the move does not put the king in check, and the square the king would pass through is not in check. When castling, the king moves two squares toward the rook and then the rook is advanced to the first square in which it is on the other side of the king.

12. A chess piece must occupy exactly one square at a time, while a square may only be occupied by one piece at a time. The only exception is auxiliary square S_0, which is occupied by all captured pieces.

13. Players alternate in taking *turns*, starting with white.

14. On her turn, a player must move exactly one of her pieces (except in the case of castling, Rule 16) from the square it currently occupies to another square (which must be empty or occupied by an opposing piece), in a manner legal for the piece being moved. If the square entered is occupied by an opposing piece, then that piece is *captured* and moved to S_0.

15. A player is said to be *in check* if her king could be captured by an opposing piece with a single legal[3] move. A player cannot make a move that would leave her in check. If, on her turn, a player is in check and every legal move available leaves her in check, then she is in *checkmate* and loses the game.

16. On her turn, instead of moving a single one of her pieces, a player may choose to *castle* her king with one of her rooks provided that she has not yet moved either piece that game, all the squares between them are empty, the move does not put her king in check, and no square the king would pass through is in check. When castling, the king moves two squares toward the rook and then the rook is advanced to the first square in which it is on the other side of the king.

6.2 Parameterized Membership of Short Generalized Chess

We show that Short Generalized Chess is in AW[*] by reducing it to Parameterized QBFSAT$_t$ (A.17): we create a Parameterized QBFSAT$_t$-formula that

[3] A *legal* move is one that does not break any of the rules of chess. Naturally, any other move is *illegal*.

captures the rules of chess and applies them to the initial position of the given input instance. We employ much the same approach as we did throughout Chapter 5: the existential and universal quantifiers simulate moves for white and black respectively, while formula F enforces the rules, tests the winning condition, and sets the initial position. For our purposes, a player can win either by achieving checkmate against the opponent, or because the opponent breaks one of the rules of chess. We remark that, to enforce checkmate rules, the formula used for our reduction actually simulates $k+2$ moves rather than just k. Any test if a rule has been broken or the game has been won has to handle the possibility that the game has already ended.

6.2.1 Encoding Positions

Let $S_{x,y}$ correspond to the square on the chessboard at row x and column y, where x and y are positive integers between 1 and n. If $a = S_{i,j}$, then we denote (i, j) also with (a_r, a_c). Further, $\mathit{board} \leftarrow \{S_{1,1}, \ldots, S_{n,n}\}$, *white* and *black* are the sets of pieces belonging to white and black respectively, and $\mathit{pieces} \leftarrow \mathit{white} \cup \mathit{black}$. Note that $S_0 \notin \mathit{board}$. We next consider encoding the positions with the following set of variables:

$$v_{p,a,t} \leftarrow \begin{cases} \mathit{true} & : \quad p \in \mathit{pieces} \text{ is on } a \in \mathit{board} \cup \{S_0\} \text{ on turn } t \\ \mathit{false} & : \quad p \in \mathit{pieces} \text{ is not on } a \in \mathit{board} \cup \{S_0\} \text{ on turn } t \end{cases}$$

Unfortunately, this natural approach to encoding positions is not parameterized. The number of chess pieces in a game is polynomial in n (i.e., $\leq 4n$), and this encoding scheme sets exactly one variable *true* for each piece on each turn. As each turn corresponds to a single quantifier, this requires that we introduce n into the Hamming weights on those quantifiers. These weights are all parameters; introducing n into any one of them prevents the reduction from preserving the parameter.

To avoid this problem we record only the changes from turn to turn rather than the positions themselves. Given an existing position, we describe a chess move as (at most) four changes to that position. Two changes always occur because a piece leaves one square (a) and enters another (b). If the move entails a capture then another two changes occur; the captured piece leaves b and enters S_0. These two observations are sufficient to describe every chess move but two: castling and pawn promotion. However, both these actions entail exactly two pieces moving. In castling (Rule 16) a rook and a king move simultaneously, while we pawn promotion involves

the pawn essentially swapping places with a piece on S_0 (Rule 6). As such, these moves are described with exactly four changes each. We define the variables for this change-of-position encoding scheme.

$$x_{p,a,t} \leftarrow \begin{cases} true & : \quad v_{p,a,t} \neq v_{p,a,t-1} \\ false & : \quad v_{p,a,t} = v_{p,a,t-1} \end{cases}$$

Using $x_{p,a,t}$, we derive the values of the original $v_{p,a,t}$-variables as follows:

$$v_{p,a,t} \quad = \quad \bigvee_{S \in Y} \left(\bigwedge_{i \in S} x_{p,a,i} \wedge \bigwedge_{i \in \{1,2,\ldots,t\}-S} \neg x_{p,a,i} \right)$$

Here, Y is the set of all even-sized subsets of $\{1, 2, \ldots, t\}$ if $v_{p,a,0}$ is *true* and the set of all odd-sized subsets of $\{1, 2, \ldots, t\}$ otherwise.

The position on turn 0 corresponds to the initial position given as input to our Short Generalized Chess-instance. Thus, in using these formulas we have implicitly encoded the initial position into the Parameterized QBFSAT$_t$-instance.

This formula is a brute-force test of all possible move sequences that result in $v_{p,a,t} = true$. Each *true* variable in $x_{p,a,u}$, $u \in \{1, \ldots, t\}$, implies that the value of $v_{p,a,u}$ has been inverted w.r.t. the previous turn; an even number of inversions preserves the initial value, while an odd number flips it.[4] For both these formulas, the number of clauses is bounded by 2^k (the number of all possible bit sequences) and thus is in FPT.

If at some point we need to negate one of the $v_{p,a,t}$-values, we simply simply start with $\neg v_{p,a,t}$ and apply DeMorgan's rule recursively until negations are applied only to literals. We denote this negation as $\overline{v_{p,a,t}}$.

This enables us to encode the board itself. However, just looking at the board does not tell us whether a piece has been moved at some time in the past – information which is necessary to enforce the rules of castling (Rule 16). We introduce flags to aid the enforcement of these rules. For each t, we define a set $CF_t \leftarrow \{c_t^{W\ell}, \overline{c_t^{W\ell}}, c_t^{Wr}, \overline{c_t^{Wr}}, c_t^{B\ell}, \overline{c_t^{B\ell}}, c_t^{Br}, \overline{c_t^{Br}}\}$ where flag c_x^{ab} is *true* if and only if on turn x castling is allowed on the left ($b = \ell$) or right ($b = r$) side for white ($a = W$) or black ($a = B$). The overlined flags carry the opposite *true/false* value of the corre-

[4] Hence why we look for an even number when the value was initially *true* and an odd number when false.

sponding non-overlined ones, so that exactly four of the variables in this set are always *true*. This means that each quantifier of our PARAMETERIZED QBFSAT$_t$-instance has a Hamming weight of 8 – 4 bits for the pieces and another 4 for the castling flags.

We now define $J_t \leftarrow \{x_{p,a,t} : p \in \mathit{pieces}, a \in \mathit{board} \cup \{S_0\}\} \cup \{y_{t,0}, y_{t,1}\}$. $y_{t,0}$ and $y_{t,1}$ are "sink" variables which are set to true if only two changes to the board positions occur on a turn. They have no other purpose. With this we can define the quantifier sets for our PARAMETERIZED QBFSAT$_t$-instance, which are pair-wise disjoint because each uses a different turn t.

$$\underset{8}{\exists}(J_1 \cup CF_1) \underset{8}{\forall}(J_2 \cup CF_2) \ldots \underset{8}{\forall}(J_t \cup CF_t)$$

6.2.2 The Winning Condition

Before we define white's winning condition W, we define several sets that we refer to throughout the reduction. We first define the sets *current* and *opponent*. For odd t, let *current* $\leftarrow$ *white* and *opponent* $\leftarrow$ *black*. For even t, let *current* $\leftarrow$ *black* and let *opponent* $\leftarrow$ *white*. We also use the set *pawns* of all pawns. Other pieces that we refer to individually are: WK, the white king, BK, the black king, WR^ℓ, the left white rook, WR^r, the right white rook, BR^ℓ, the left black rook, and BR^r, the right black rook.

$$W \leftarrow \bigvee_{0 \le t \le k} (v_{BK,0,t} \wedge \overline{v_{WK,0,t}})$$

Rather than testing whether a position is checkmate for either player, we test whether a king has been captured on the following turn. This simple test has the advantage that it also handles the rules regarding check, as shown below. The winning condition for white as presented in W is simply to capture BK before WK is captured, as WK being captured previously would imply that black had already won.

Lemma 22. *The rules regarding check and checkmate over the next t turns resolve under optimal play to the winning condition of capturing the opponent's king within the next $t+2$ turns.*

Proof. Rule 15 states the rules regarding check and checkmate. A player cannot choose to make a move that would result in his king being left in check at the end of his turn. If a player violates this rule then her opponent can immediately capture that player's king with his next move. Thus, if a player violates this rule on turn t,

then he will lose the game on turn $t+1$. Similarly, if a player is in checkmate then she has no legal move but to leave her king in check and consequently she will lose on her opponent's next move. If the move that creates the checkmate occurs on turn t, then on turn $t+1$ the player moving must either break a rule (in which case she loses immediately) or leave her king in check (in which case the king can be captured on turn $t+2$). This resolves both check and checkmate. However, by enforcing the rules in this manner we delay enforcement by two turns. Because of this delay, our reduction evaluates two additional moves (to turn $t+2$). Otherwise, leaving a king in check or creating a checkmate on turn t will have no consequence. □

6.2.3 Formula F

We use F – the formula for the Parameterized QBFSAT$_t$-instance produced by the reduction – to enforce the rules of chess, including winning condition W. Intuitively, F is *true* in exactly two cases: if no rules are broken and player one wins, or if black breaks a rule. F is false otherwise. To handle the rules of chess, we define R_{all} and R_{black}. R_{all} tests that all the rules were followed, R_{black} tests that on each of his turns, either black follows the rules or the game is over.[5] Both formulas use formula L_t. Intuitively, L_t is *true* iff the change in position from turn $t-1$ to t describes a legal move. The formal definition of L_t appears in the next subsection.

$$R_{all} \leftarrow \bigwedge_{1\le t\le k} (L_t \vee v_{BK,0,t-1} \vee v_{WK,0,t-1})$$

$$R_{black} \leftarrow \bigwedge_{1\le t\le \frac{k}{2}} \left(L_{2t} \vee v_{BK,0,2t-1} \vee v_{WK,0,2t-1} \vee \bigvee_{1\le s\le t} \overline{L_{2s-1}}\right)$$

We now combine these formulas resulting in formula $F \leftarrow (R_{all} \wedge W) \vee \overline{R_{black}}$. F is satisfied if both players play a legal game that ends with white as the winner, or if black moves illegally before the game ends.

6.2.4 Testing for Broken Rules

We define $L_t \leftarrow \overline{M_t} \wedge P_t \wedge C_t \wedge E_t \wedge \overline{D_t} \wedge K_t$, where M_t tests for illegal movements, P_t tests whether the path of moving pieces is clear, C_t tests that the capture rules

[5]Here, the game is be over because either king was captured or because white moved illegally on a previous turn.

have been followed, E_t tests that every piece exists exactly once this turn, D_t tests if two pieces were moved simultaneously, and K_t maintains flags for handling castling moves. We elaborate on each of these sub-formulas.

Range of Movement (M_t). M_t tests for illegal movements. To handle movement rules we introduce:

$$\Delta(p,a,b) \leftarrow \begin{cases} \mathit{true} & : \quad \text{square } b \text{ is within } p\text{'s range of movement from square } a \\ \mathit{false} & : \quad \text{square } b \text{ is outside } p\text{'s range of movement from square } a \end{cases}$$

A piece's *range of movement* is considered to be its available moves under ideal conditions (e.g. there are no pieces in the way). A pawn's range of movement includes the diagonal movements that a pawn can make only if the move captures an opposing piece.

$$M_t \leftarrow \bigvee_{\substack{p\in current\\ a,b\in board\\ \Delta(p,a,b)=false}} (v_{p,a,t-1} \wedge v_{p,b,t}) \vee \bigvee_{\substack{p\in opponent\\ a,b\in board}} (v_{p,a,t-1} \wedge v_{p,b,t})$$
$$\vee \bigvee_{\substack{p\in current\cap pawns\\ a,b\in board, a_c\neq b_c\\ \Delta(p,a,b)=true}} \left(v_{p,a,t-1} \wedge v_{p,b,t} \wedge \bigwedge_{q\in opponent} \overline{v_{q,b,t-1}}\right)$$

Lemma 23. *M_t is* true *iff on turn t a piece is moved illegally.*

Proof. If the current player moves one of his own pieces in a way that the given piece is not allowed to move (such as a rook moving diagonally) then the first part of the formula $\bigvee_{\substack{p\in current\\ a,b\in board\\ \Delta(p,a,b)=false}} (v_{p,a,t-1} \wedge v_{p,b,t})$ is *true* because the piece went outside its range of movement. Thus, M_t is *true.* If the current player moves one of his opponent's pieces to a location other than S_0, then the second part of the formula $\bigvee_{\substack{p\in opponent\\ a,b\in board}} (v_{p,a,t-1} \wedge v_{p,b,t})$ is *true.* Thus, M_t is *true.*

The two cases above handle every case except for the conditional range of pawns. If the current player moves one of his own pawns diagonally, then the third part of

the formula $\bigvee_{\substack{p\in current\cap pawns\\ a,b\in board, a_c\neq b_c\\ \Delta(p,a,b)=true}} (v_{p,a,t-1} \wedge v_{p,b,t} \wedge \bigwedge_{q\in opponent} \overline{v_{q,b,t-1}})$ is *true* if square b moved to contains no opposing piece. □

Path of Movement (P_t). P_t tests whether the path of every moving piece, except the knight's, is clear. We define a path $path(a,b)$ on the chessboard using two squares $a, b \in board$ with a and b belonging to the same column, row, or diagonal of the chessboard. The squares visited by moving from a to b along the associated row, column, or diagonal are on the path between a and b.

$$\begin{aligned} P_t &\leftarrow \bigwedge_{\substack{p\in current\\ a,b\in board}} \left(\overline{v_{p,a,t-1}} \vee \overline{v_{p,b,t}} \vee Q_{a,b,t} \wedge \bigwedge_{\substack{q\in pieces\\ q\neq p}} \overline{v_{q,b,t}} \right) \\ &\wedge \bigwedge_{\substack{p\in current\cap pawns\\ a,b\in board, a_c=b_c}} \left(\overline{v_{p,a,t-1}} \vee \overline{v_{p,b,t}} \vee \bigwedge_{q\in opponent} \overline{v_{q,b,t-1}} \right) \\ Q_{a,b,t} &\leftarrow (a \text{ and } b \text{ share a row, column, or diagonal in common}) \vee \bigwedge_{\substack{p\in pieces\\ d\in path(a,b)}} \overline{v_{p,d,t}} \end{aligned}$$

Lemma 24. *If squares a and b are the same row, column, or diagonal, then $Q_{a,b,t}$ is* true *iff all the squares between (but not including) a and b along the associated row, column, or diagonal are unoccupied on turn t.*

Proof. If on turn t, a player moves a piece p from a to b through or into a square that is occupied by another piece q, then $(\overline{v_{p,a,t-1}} \vee \overline{v_{p,b,t}})$ is false (since p was in square on turn $t-1$ and moved to square b on turn t). If p was moved through an occupied square, then $Q_{a,b,t}$ is false. Otherwise, p was moved into an occupied square and $\bigwedge_{\substack{q\in pieces\\ q\neq p}} \overline{v_{q,b,t}}$ is false. In either case, the first clause of the formula is false and so is P_t.

Again, pawns are an exception here. Unlike all other chess pieces, they cannot always capture when they move. Specifically, a pawn cannot capture by moving forward. If the current player moves a pawn p forwards from a into a square b

occupied by an opposing piece q, the last part of the formula

$$\bigwedge_{\substack{p \in current \cap pawns \\ a,b \in board \\ a_c = b_c}} \left(\overline{v_{p,a,t-1}} \vee \overline{v_{p,b,t}} \vee \bigwedge_{q \in opponent} (\overline{v_{q,b,t-1}}) \right)$$

is false because $v_{p,a,t-1}$ and $v_{p,b,t}$ are both *true*, as is $v_{q,b,t-1}$. □

If a and b do not share a row, column, or diagonal, then either the movement is illegal and rejected by M_t, or the piece moved is a knight and can legally jump over other pieces. In either case, $Q_{a,b,t}$ being empty and trivially true is correct. M_t tests for illegal piece movements, and D_t ensures that only one piece is moved on the board at once. Thus, it is sufficient for P_t to ensure that pieces move through empty squares.

Lemma 25. *P_t is* true *iff all pieces legally moved on turn t are moved through empty squares and end their move in a square that is not occupied by a same-colored piece on turn $t-1$.*

Capturing (C_t). C_t tests that if piece p arrived this turn in square a, occupied by a opposing piece q last turn, then q is moved to S_0. Similarly, if an opposing piece q arrived in S_0 this turn, then whichever square q occupied last turn is now occupied by a piece controlled by the current player.

$$
\begin{aligned}
C_t \quad \leftarrow \quad & \bigwedge_{\substack{p \in current \\ q \in opponent \\ a \in board}} (\overline{v_{p,a,t}} \vee \overline{v_{q,a,t-1}} \vee v_{q,0,t}) \\
\wedge \quad & \bigwedge_{q \in opponent} \left(\overline{x_{q,0,t}} \vee \bigvee_{\substack{p \in current \\ a \in board}} (v_{p,a,t} \wedge v_{q,a,t-1}) \right) \wedge \bigwedge_{p \in opponent} (\overline{v_{p,0,t-1}} \vee v_{p,0,t}) \\
\wedge \quad & \bigwedge_{p \in current} \left(\overline{v_{p,0,t-1}} \vee v_{p,0,t} \vee \bigvee_{\substack{i \in \{1,\ldots,n\} \\ q \in pawns \cap current}} U_{p,q,t,i} \right) \\
\wedge \quad & \bigwedge_{p \in current\ \cap pawns} \left(v_{p,0,t-1} \vee \overline{v_{p,0,t}} \vee \bigvee_{\substack{i \in \{1,\ldots,n\} \\ q \in current\ - pawns \cap current}} U_{q,p,t,i} \right) \\
\wedge \quad & \bigwedge_{p \in current\ - pawns} (v_{p,0,t-1} \vee \overline{v_{p,0,t}}) \\
U_{p,q,t,i} \quad \leftarrow \quad & \left(v_{q,S_{i,n-1},t-1} \wedge v_{q,0,t} \wedge v_{p,S_{i,n},t} \wedge \bigwedge_{\substack{r \in pieces \\ r \neq p}} \overline{v_{r,S_{i,n},t}} \right)
\end{aligned}
$$

C_t does not deal with the move that caused the capture, as that is handled by M_t and P_t.

Lemma 26. *C_t = true iff all pieces that were captured on turn t moved to S_0, no opponent's pieces moved to S_0 on turn t that were not captured on turn t, and no pieces left S_0 on turn t except by pawn promotion.*

Proof. If, on turn t, the current player moved piece p into square a that was occupied by an opposing piece q, but did not move q to S_0, then the first clause $(\overline{v_{p,a,t}} \vee \overline{v_{q,a,t-1}})$ is false, and so is C_t.

If the current player moves an opposing piece q from square a to S_0 on turn t without moving another piece p to capture q, then

$$
\bigwedge_{q \in opponent} \left(\overline{x_{q,0,t}} \vee \bigvee_{\substack{p \in current \\ a \in board}} (v_{p,a,t} \wedge v_{q,a,t-1}) \right)
$$

is false, as is C_t. If the current player moves an opposing piece q to square a from S_0, then

$$\bigwedge_{p \in opponent} (\overline{v_{p,0,t-1}} \vee v_{p,0,t})$$

is false and so is C_t. If the current player moves one of his pieces p out of S_0 on turn t, then

$$\bigwedge_{p \in current} \left(\overline{v_{p,0,t-1}} \vee v_{p,0,t} \vee \bigvee_{\substack{i \in \{1,\ldots,n\} \\ q \in pawns \\ q \in current}} \left(v_{q,S_{i,n-1},t-1} \wedge v_{q,0,t} \wedge v_{p,S_{i,n},t} \wedge \bigwedge_{\substack{r \in pieces \\ r \neq p}} \overline{v_{r,S_{i,n},t}} \right) \right)$$

is false unless p is swapped with a pawn q that moved into the opponent's home row on turn t. □

Every Piece Exists Exactly Once (E_t). $E_t \leftarrow O_t \wedge \overline{T_t}$ tests that every piece is in exactly one position on turn t, but not in two positions.

$$\begin{aligned} O_t &\leftarrow \bigwedge_{p \in pieces} \left(\bigvee_{a \in board \cup \{0\}} v_{p,a,t} \right) \\ T_t &\leftarrow \bigvee_{\substack{p \in pieces \\ a,b \in board \cup \{0\} \\ a \neq b}} (v_{p,a,t} \wedge v_{p,b,t}) \end{aligned}$$

Lemma 27. *E_t is true iff on turn t, for every $p \in pieces$ there is exactly one $a \in board \cup \{0\}$: $v_{p,a,t}$ true.*

Pieces Moving Simultaneously (D_t). D_t tests if two pieces were moved simultaneously on the board. Each quantifier has Hamming weight 8. Exactly four elements in the quantified sets are used up by the castling flags – four remain. The four bits available allow two pieces to move in the event of capture, castling, or promotion. Otherwise, we restrict the players from moving two board pieces simultaneously.

$$
\begin{aligned}
D_t &\leftarrow \bigvee_{\substack{p,q \in pieces \\ a,b,c,d \in board \\ a \neq b, c \neq d, p \neq q}} \left(x_{p,a,t} \wedge x_{p,b,t} \wedge x_{q,c,t} \wedge x_{q,d,t} \wedge \neg Y_t^W \wedge \neg Y_t^B \right) \\
Y_t^W &\leftarrow (c = WK \wedge d = WR^\ell \wedge v_{WK,S_{1,\frac{n}{2}},t-1} \wedge v_{WK,S_{1,\frac{n}{2}-2},t} \wedge v_{WR^\ell,S_{1,1},t} \\
&\wedge \quad v_{WR^\ell,S_{\frac{n}{2}-1,1},t} \wedge c_{t-1}^{W\ell} \wedge \overline{c_t^{W\ell}} \wedge \overline{c_t^{Wr}} \wedge A_{S_{1,\frac{n}{2}},t} \wedge A_{S_{1,\frac{n-1}{2}},t}) \\
&\vee \quad (c = WK \wedge d = WR^r \wedge v_{WK,S_{1,\frac{n}{2}},t-1} \wedge v_{WK,S_{1,\frac{n}{2}+2},t} \wedge v_{WR^r,S_{1,1},t} \\
&\wedge \quad v_{WR^r,S_{\frac{n}{2}+1,1},t} \wedge c_{t-1}^{Wr} \wedge \overline{c_t^{W\ell}} \wedge \overline{c_t^{Wr}} \wedge A_{S_{1,\frac{n}{2}},t} \wedge A_{S_{1,\frac{n+1}{2}},t}) \\
A_{s,t} &\leftarrow \bigwedge_{p \in opponent} \left(\bigwedge_{\substack{a \in board: \Delta(p,a,s) \\ a \neq s}} \left(\overline{v_{p,a,t}} \vee \overline{Q_{a,s,t}} \right) \right)
\end{aligned}
$$

Y_t^B is identical to Y_t^W, except B is substituted for W and the squares are adjusted appropriately. Tests $c = WK \wedge d = WR^\ell$ ensure we consider the correct pieces. The next four tests involving v ensure the pieces are engaged in a castling motion. The next three tests (involving c) ensure that the castling flags are maintained. The last two tests (involving A) handle the rule that a castling must not pass through a square in check. $A_{s,t}$ is *true* iff square s is not threatened on turn t by a piece in *opponent*. This explains the first clause; the other is identical, except that ℓ is replaced with r and squares are on the right instead of left.

Lemma 28. *D_t is* true *iff there exist two pieces p, q which both moved on turn t without starting or ending their moves in S_0, and the move is not a legal castling.*

Proof. If the current player on turn t moves two pieces p and q on the board (specifically, p moves from a to b, and q from c to d) then $(x_{p,a,t} \wedge x_{p,b,t} \wedge x_{q,c,t} \wedge x_{q,d,t})$ is *true*. If the move is also not a legal castling, then Y_t^W and Y_t^B are both false. Thus, D_t is *true*. □

Castling Maintenance (K_t). $K_t \leftarrow K_t^W \wedge K_t^B$ tracks castling flags. Remember that a chessboard position includes whether any given king or rook has been moved from its starting square. K_t tracks this information.

$$\begin{array}{rcl} K_t^W & \leftarrow & (c_t^{W\ell} \vee \overline{c_t^{W\ell}}) \wedge (c_t^{Wr} \vee \overline{c_t^{Wr}}) \wedge (\neg c_t^{W\ell} \vee \neg \overline{c_t^{W\ell}}) \wedge (\neg c_t^{Wr} \vee \neg \overline{c_t^{Wr}}) \\ & \wedge & (\neg \overline{c_{t-1}^{W\ell}} \vee \overline{c_t^{W\ell}}) \wedge (\neg \overline{c_{t-1}^{Wr}} \vee \overline{c_t^{Wr}}) \wedge (x_{WR^\ell, S_{1,1}, t} \vee \overline{c_t^{W\ell}}) \\ & \wedge & (x_{WR^r, S_{1,n}, t} \vee \overline{c_t^{Wr}}) \wedge (\neg x_{WK, S_{1,\frac{n}{2}}, t} \vee (\overline{c_t^{W\ell}} \wedge \overline{c_t^{Wr}})) \\ & \wedge & (\overline{c_{t-1}^{W\ell}} \vee c_t^{W\ell} \vee t \text{ is odd }) \wedge (\overline{c_{t-1}^{Wr}} \vee c_t^{Wr} \vee t \text{ is odd }) \end{array}$$

K_t^B is defined as K_t^W, except B is substituted for W, the starting positions are updated accordingly, and t must be even instead of odd.

Lemma 29. *K_t is* true *iff the states of the castling flags have been maintained properly.*

Proof. If one of the flags is neither on nor off, then one of the clauses in $(c_t^{W\ell} \vee \overline{c_t^{W\ell}}) \wedge (c_t^{Wr} \vee \overline{c_t^{Wr}}) \wedge (\neg c_t^{W\ell} \vee \neg \overline{c_t^{W\ell}}) \wedge (\neg c_t^{Wr} \vee \neg \overline{c_t^{Wr}})$ is false and thus K_t is false.

If one of the flags has been turned back on (which is never legal because you cannot undo past moves), one of $(\neg \overline{c_{t-1}^{W\ell}} \vee \overline{c_t^{W\ell}}) \wedge (\neg \overline{c_{t-1}^{Wr}} \vee \overline{c_t^{Wr}})$ is false and thus K_t is also false.

If the current player moves one of his rooks without turning off the associated castling flag, then one of $(x_{WR^\ell, S_{1,1}, t} \vee \overline{c_t^{W\ell}}) \wedge (x_{WR^r, S_{1,n}, t} \vee \overline{c_t^{Wr}})$ is false and again K_t is false.

If the current player moves her king without turning off both her castling flags, then $(\neg x_{WK, S_{1,\frac{n}{2}}, t} \vee (\overline{c_t^{W\ell}} \wedge \overline{c_t^{Wr}}))$ are false and thus K_t is as well.

Finally, if the current player tries to turn her opponent's castling flags off, then one of $(\overline{c_{t-1}^{W\ell}} \vee c_t^{W\ell} \vee t \text{ is odd})) \wedge (\overline{c_{t-1}^{Wr}} \vee c_t^{Wr} \vee t \text{ is odd})$ is false and so is K_t. □

6.2.5 Correctness of the Reduction

Theorem 8. *L_t is* true *iff the differences between position $\{p \in \text{pieces}, a \in \text{board} \cup \{0\} : v_{p,a,t}\}$ on turn $t-1$ and position $\{p \in \text{pieces}, a \in \text{board} \cup \{0\} : v_{p,a,t-1}\}$ on turn t describe a legal chess move.*

Proof. E_t ensures that there is exactly one of each piece, preventing pieces from spontaneously appearing or disappearing. Consequently, pieces can only move from one position to another. M_t ensures that any pieces moved on the board are moved legally, and P_t ensures that the paths of those moves are clear. C_t ensures that any piece which should be captured is indeed moved to S_0, that no pieces are moved to S_0

without being captured, and that any pieces which are captured do not return to the board. D_t ensures that only one piece moves on the board (excluding captures and castling), and K_t ensures that the castling flags are maintained properly. Finally, the hamming weight of 4 (8 total – 4 for castling flags), combined with the fact that we have only two "sink" variables means that at least one move must take place. □

Lemma 30. *Formula F is 10-normalized.*

Proof. This can be done by inspection, starting with $v_{p,a,t}$ which is 2-normalized. M_t and D_t are 4-normalized. P_t, C_t, and E_t are 5-normalized. K_t is 3-normalized. Therefore L_t is 6-normalized, R_{ALL} is 8-normalized, and R_{even} is also 8-normalized. W is 4-normalized by inspection, so F is 10-normalized. □

Theorem 9. *F is* true *iff white has a winning strategy that takes at most t turns to execute.*

Proof. If white has a winning strategy and plays legally, then either black plays legally (R_{ALL} is *true*) and white wins the game (W is *true*) or black makes an illegal move (R_{even} is false). In either case, F is satisfied.

If white does not have a winning strategy, then in all instances where black plays legally (R_{even} is *true*), either white does not win (W is false) or else white makes an illegal move (R_{ALL} is false). □

6.3 Hardness of Short Generalized Chess

We show that Short Generalized Chess is AW[*]-hard by reduction from Unitary Monotone Parameterized QBFSAT$_2$.

In our reduction, we create a chess-instance that uses a large block of pawns arranged in a checkerboard pattern at the center of the board (*walls of pawns*). We carve out paths from the inside of this checkerboard, through which the white queen, several black bishops, and two black rooks are able to move. The white queen then guarantees capture of BK within k turns iff the Unitary Monotone Parameterized QBFSAT$_2$-instance is not satisfiable.

We use a set of *variable diagonals* for our chess-instance to represent the variable assignment. Each Unitary Monotone Parameterized QBFSAT$_2$-variable has one corresponding diagonal, and the game simulates setting values for these variables by moving black bishops into the diagonals that are "true". Due to the unitary nature

of Unitary Monotone Parameterized QBFSAT$_2$, each diagonal represents every other variable in its subset as *false*; if a variable $t \in S$ is *true*, then for $u \in S, u \neq t$, u is *false* because the Hamming weight of S is 1.

The variable diagonals are crossed by a set of vertical paths (*clauseways*). The intersections between variable diagonals and clauseways are constructed such that if variable $t =$ *true* implies that clause C is satisfied, then a black bishop patrolling the corresponding diagonal blocks the white queen's progress through the clauseway corresponding to C. If otherwise assigning $t =$ *true* does not satisfy C, then the white queen captures a black bishop by blocking the associated clauseway without any repercussions. Thus, white has only enough turns to capture *BK* if all k bishops in the chosen clauseway can be captured. Any bishop that cannot be captured forces white to spend extra turns moving the queen along an alternate route.[6]

Before we present the detailed gadgets of the reduction, we note important properties of the position that we maintain: The only white piece free to move is the queen. Other pieces are able to move only if black does not move as intended and attacks a white pawn. Consequently, white is unable to win the game if her queen is captured. If the white queen moves as the reduction intends, then on every turn black is forced to *move in reply*[7]; failing to react to white's move[8] results in white being able to capture *BK*. Thus, we can guarantee certain constructs of black pieces to be *unassailable*. That is, if white attacks these pieces with her queen, black can capture the queen and prevent white from winning the game. For example, consider a black pawn with a bishop in the diagonally adjacent square below and to the right of it. If the white queen takes the black bishop, then the black pawn can capture the white queen, and vice versa. Because these two pieces protect each other the white queen cannot capture either. Furthermore, this protection can be extended to any other black piece in a square that either of these first two pieces could attack; the white queen cannot capture these other protected pieces without first capturing the piece that protects themselves, but we know that these pieces are unassailable. The protection can be extended recursively; this allows us to eliminate the possibility of the white queen tunneling through the pawn walls.

Walls of Pawns. Pawns have unique properties: they can be blocked from moving

[6]This makes it impossible to reach the king in the number of turns allotted.

[7]That is, white will win unless black chooses a move (from a very limited set) that can alleviate the threat.

[8]That is, moving an unrelated piece elsewhere on the board.

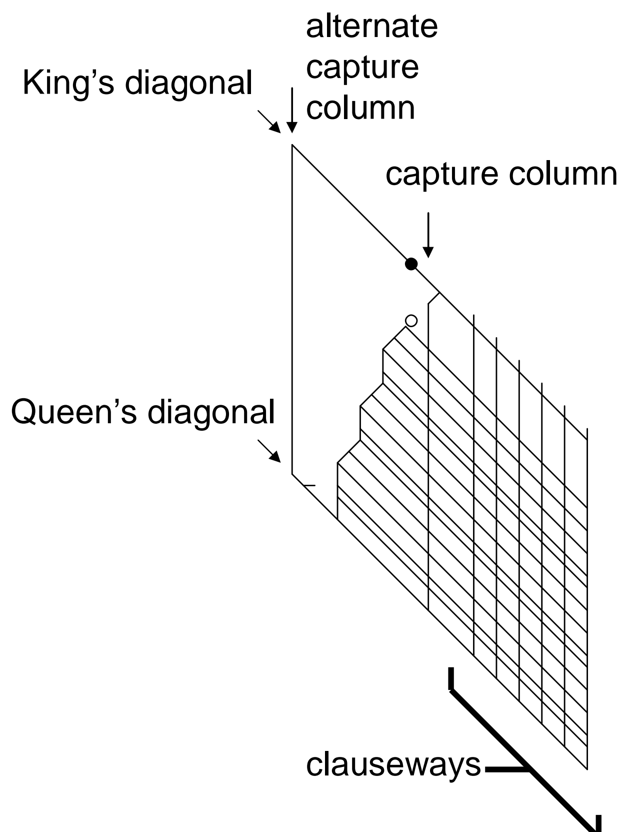

Figure 6.1: An overview of the chessboard as built by the reduction.

forward by an opposing piece, and they cannot move diagonally unless attacking an opposing piece. With these two properties, we can create a situation where no moves are possible simply by placing a single white pawn one square above a single black pawn on an otherwise-empty chessboard (Figure 6.2).

We extend this observation by arranging an infinite number of pawns in a checkerboard pattern. No pawn can move to the next square up or down because that square is occupied by a pawn of the opposite color. No pawn can move diagonally because those squares are occupied by pawns of the same color (Figure 6.3).

Given such a checkerboard pattern, we are able to remove pawns while still preserving the property that no moves or captures are possible. The most basic way to do so is by removing an adjacent pair of pawns consisting of a white pawn one square above a black pawn. Then the pawns above and below along the same column are still blocked, and the pawns on either side cannot move diagonally because there are

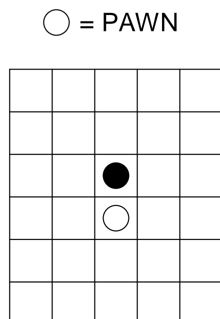

Figure 6.2: Deadlocked pawns.

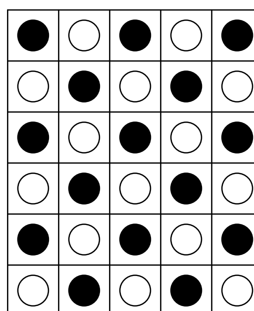

Figure 6.3: Deadlocked pawns laid out in a checkerboard arrangement.

no pieces in the newly-emptied squares to capture. We can further carve out paths by using repeated deletions. We can cut vertically (as demonstrated in Figure 6.4), diagonally (Figure 6.5), and horizontally (Figure 6.6) so long as the topmost deleted pawn in any column in black and the bottommost is white.

Assignment (Crooked Path) Gadget. This gadget enables the two players to alternate setting values for the variables of the UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$-instance. It does this by having the queen start at the top of a crooked path (as can be seen in Figure 6.1). This path alternates between vertical and diagonal segments (see Figure 6.7). In each segment, exactly one quantifier is handled. The game simulates setting the values for the variables in the same order as the UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ instance.

Universal variables are set by white. Each segment of the crooked path intersects with all the assignments diagonals corresponding to variables handled by the universal

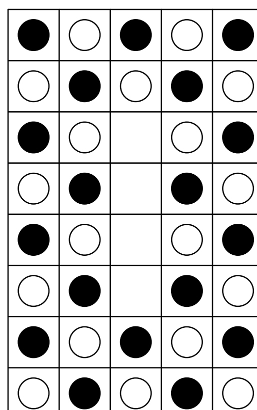

Figure 6.4: Pawns deadlocked in a checkerboard arrangement with a column segment removed.

quantifier corresponding to that segment. When the queen moves to the end of a variable diagonal, the available black bishop must move to block the queen from moving down the diagonal into the capture column. A second black bishop protects the first (Figure 6.8).

Existential variables are set by black. At the end of each segment is a horizontal path leading to a diagonal which intersects the perpendicular variable diagonals (as above for universal variables). A black bishop is placed in this diagonal such that the queen can attack it using the horizontal path. When the queen reaches the end of a segment, it can attack this bishop and gain entry to the capture column unless black moves the bishop. Since the bishop is forced to move, black can protect one of the variable diagonals (Figure 6.9).

Lemma 31. *Black must react to white in the assignment gadget.*

Proof. When the white queen positions itself to enter one of the variable diagonals, black must immediately react to prevent white from entering the capture column. The only way this is possible is for black to put a bishop in the way.

Similarly, when the queen arrives at the end of a segment, it threatens to capture a black bishop. The bishop must move out of the way or the queen can capture it and enter the capture column. Black might alternatively choose to move a black pawn down to block the queen, but then black uses his turn and loses the chance to guard a variable diagonal (i.e. black loses the opportunity to block any clauseways with that bishop). □

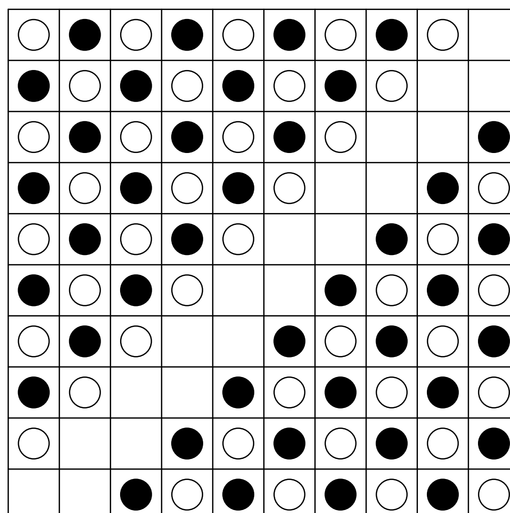

Figure 6.5: Pawns deadlocked in a checkerboard arrangement with a diagonal removed.

Lemma 32. *Black cannot make a move in the assignment gadget to prevent white from using the capture column.*

Proof. While black could move a bishop into the capture column, it cannot actually attack the queen. As such, the queen can simply move up the column and capture it. If the bishop is not actually blocking the queen's passage, then the queen can simply move past it. □

Lemma 33. *White cannot win by circumventing the assignment gadget.*

Proof. White may choose to bypass the selection of a universally-quantified variable, and in doing so white effectively gains an extra turn. However, the bishop remains in its initial position, and that diagonal can protect every clauseway so white will be forced to use two turns going around it. This results in a net loss of one turn, and thus white will not have enough turns to simply go around all the bishops in the clauseways.

Attempting to attack a bishop blocking a variable diagonal is futile. As shown in Figure 6.8, any square that the queen might try to move into is protected, and thus black can capture the queen if any attempt is made. Black cannot tunnel through the wall of pawns because every pawn is protected (as shown by the arrows).

Similarly, if the white queen attempts to attack through the horizontal passage for an existentially-qualified variable, Figure 6.9 shows that there are again no safe

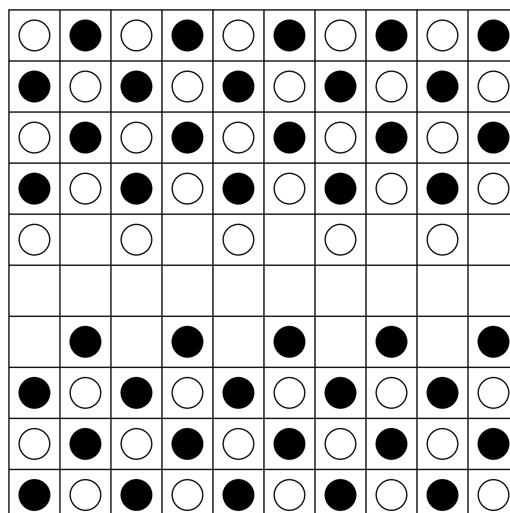

Figure 6.6: Pawns deadlocked in a checkerboard arrangement with battlement cuts.

spaces and black can capture the queen. Again, the queen cannot tunnel through the wall because the pawns (or the noted black bishops) can capture the queen. □

Capture Column. The capture column is to the right of the assignment gadget, and intersects every variable diagonal. The top of the column connects to the king's diagonal through a short diagonal. The bottom of the capture column intersects with the queen's diagonal in the one-way gadget, explained below.

Alternate Capture Column. There exists a second, alternate capture column on the left side of the assignment gadget. This column intersects only the king's diagonal at the top, and the queen's diagonal at the bottom. A small gadget (Figure 6.10) located in the queen's diagonal allows black to prevent the queen from entering the alternate capture column. In fact, black must do this when the queen enters the queen's diagonal because otherwise white wins. Once this route is blocked, the only route available to the queen is through the one-way gadget and toward the clauseways.

One-Way Gadget. This gadget is placed between the white queen's exit point from the crooked path and the entry points of the clauseways. It enables black to protect the capture column after white has passed through. This is accomplished by placing a black pawn at the bottom of the capture column in the queen's path, with a white one beneath it. When the queen captures the black pawn, the black rook takes the

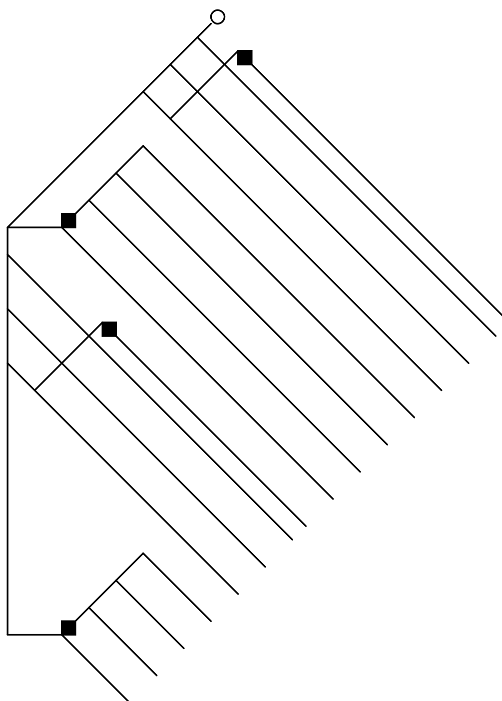

Figure 6.7: The layout of the assignment gadget. This shows only two segments. Additional segments are added to the end as necessary.

white pawn. If the queen now enters the capture column the rook can capture it.

Lemma 34. *Black must react to white in the one-way gadget.*

Proof. When the queen takes the black pawn, black must move the rook to guard the capture column (and coincidentally threaten the queen) or else the queen can move up it and checkmate the king.

Alternatively, black can move a bishop into the capture column to prevent the queen from advancing all the way to the end. However, bishops are not protected in the capture column so the white queen can simply take this bishop (and any others that are moved to block). If black moves the rook to threaten the capture column after one or more bishops have been captured then white can still move the queen into the clauseways and has gained an advantage there since black has fewer bishops to guard them.

Black has no other mobile pieces that can interact with the capture column. □

Lemma 35. *Black cannot prevent white from using the capture column while the*

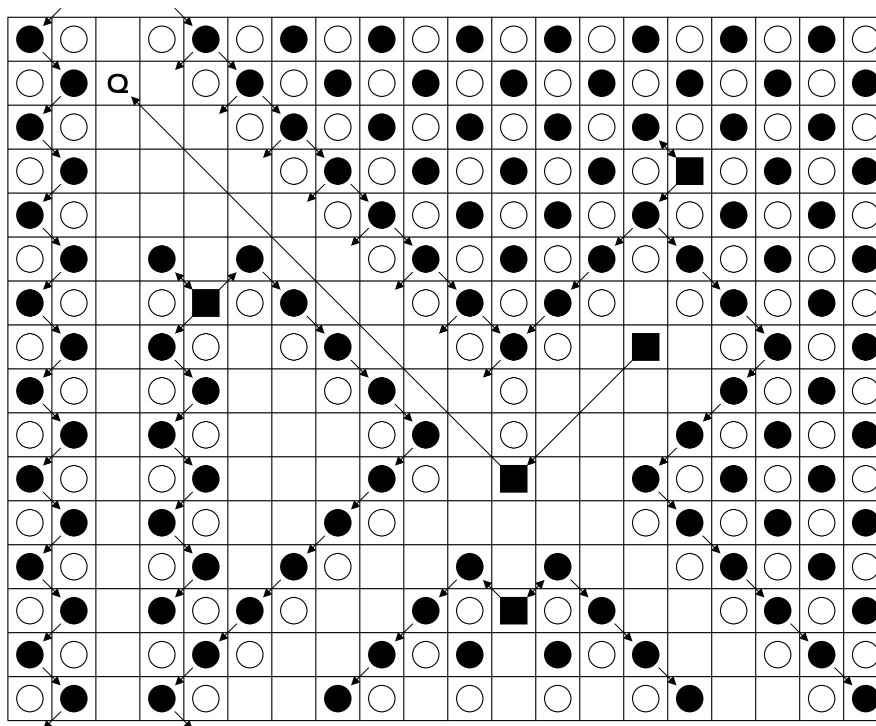

Figure 6.8: Interaction in the variable assignment gadget. Black has responded to white's attack on the column. Arrows indicate possible moves by black to capture the queen if white attacks the gadget.

queen is in the assignment gadget.

Proof. Black has no queen, so a rook is the only piece that can threaten the column.

If the black pawn that the queen is supposed to capture is not moved or captured, then to threaten the capture column the rook must move into the square directly above this pawn. However, a white pawn protects this square.

Black can, instead, capture the white pawn with the rook, move it back to its original position in the gadget, then move the black pawn down to open up a square from which the rook can threaten the capture column. At that point it would take two more turns to move the rook to where the black pawn originally started (alternatively, two turns to move the pawn down another square and the rook into the capture column). This is, in total, five moves. However, as we show below this sequence is too long to stop the queen from using the capture column.

If black ignores the setting of a universal variable to start this sequence, then the queen can checkmate WK in three turns: one to move into the capture column, one

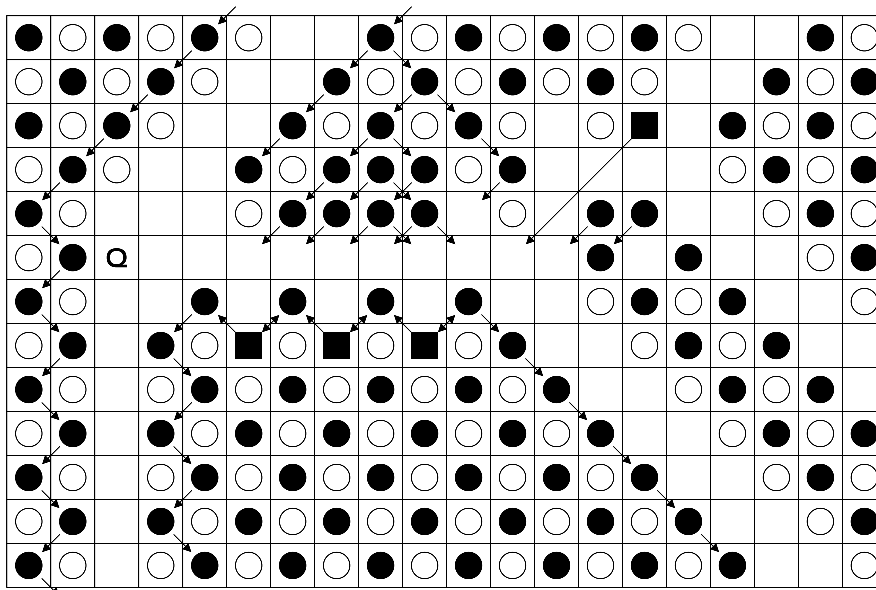

Figure 6.9: Black has moved the bishop out of the queen's path and set the value of an existential variable in the process.

to move to the top, and one to move into the king's diagonal. Even with a one-turn head start, the rook won't even be in position before the queen has left the capture column.

If black instead ignores the setting of an existential variable, then it takes the queen an additional turn to move into the capture column and four turns to checkmate the king. This means that the queen will have the king in checkmate just before black can make the fifth and final move of the sequence.

Black's only other rook is too far away to be moved into this gadget, the capture column, or any variable assignment gadgets in less than five turns. □

Lemma 36. *White cannot circumvent the one-way gadget.*

Proof. Figure 6.11 shows that if the queen moves to a square other than the one occupied by the black pawn, it will be captured. Once the queen captures the black pawn, the black rook can take the white pawn and threaten the entire capture column. The queen cannot attack the rook because it is protected by the bishop. If the queen moves to attack the bishop, it either moves into a square where it can be captured (either by a pawn or the rook) or the rook can move horizontally to protect the bishop (in which case the rook is protected by a pawn). □

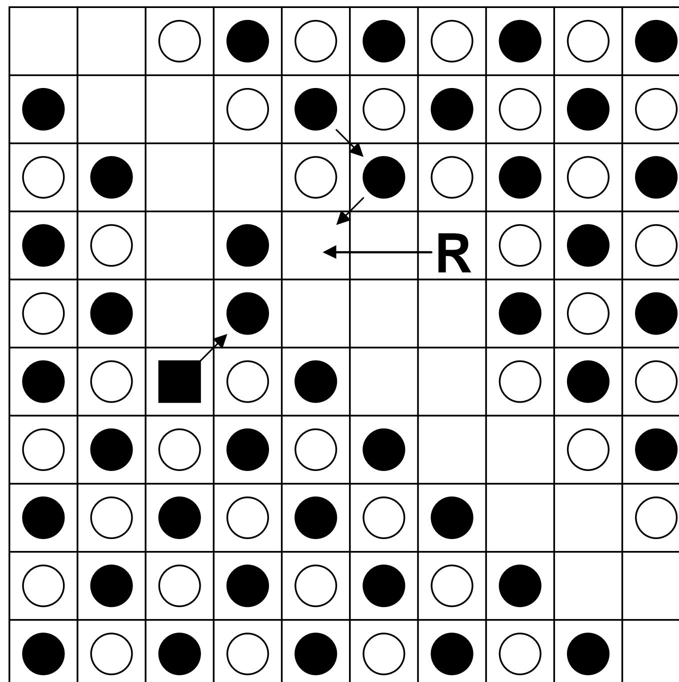

Figure 6.10: A gadget to allow black to block the alternate capture column. Once the rook is moved next to the two black pawns, the diagonal is blocked and all the black pieces blocking it are protected.

Clauseways. A clauseway is a narrow vertical path that crosses every variable diagonal. For each variable and each clause, if setting the given variable to true would satisfy that clause then a black bishop can safely protect the intersection between the given variable diagonal and clauseway. That is, a queen moving up this clauseway would be forced to move around the bishop rather than capture it.

To create this property, we use a wall construct that is slightly different from the checkered pawn configuration. The intersections between the vertical clauseways and variable diagonals produce parallelogram-shaped walls. We use a different construct for these specific walls, illustrated in Figure 6.12. The advantage of these is that their height can be adjusted arbitrarily by extending the segment within the dotted lines. Thus, we can protect or not protect the bishop simply by moving the bottom of the parallelogram sitting above and to the right of the intersection (Figures 6.14 and 6.13).

Lemma 37. *Black must react to white in the clauseways.*

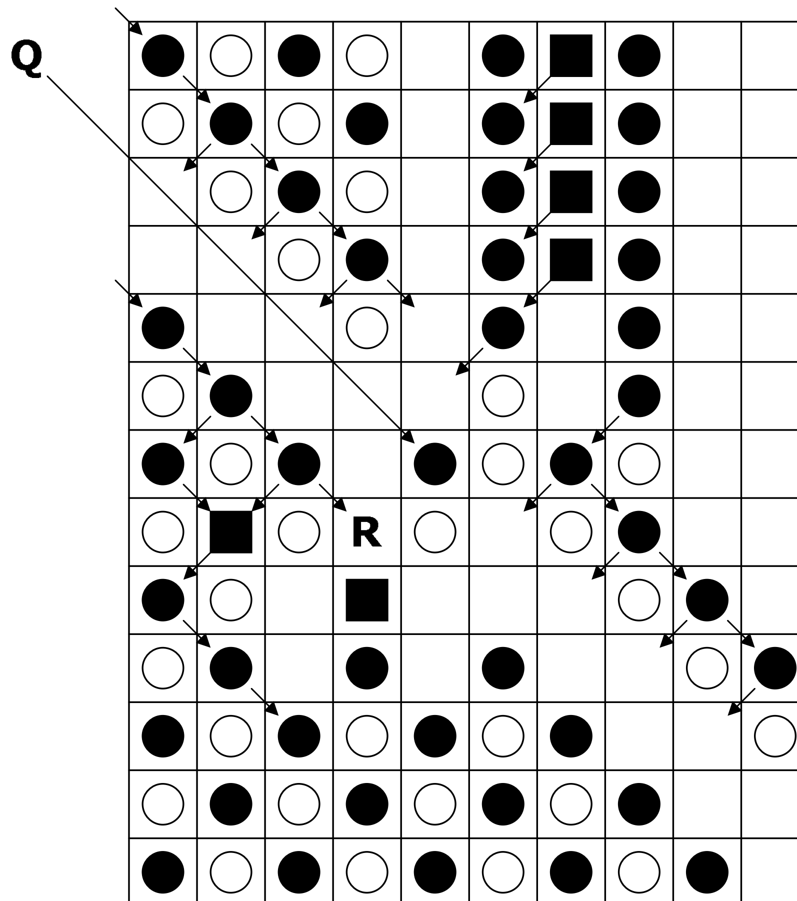

Figure 6.11: Interaction in the one-way gadget. The queen is about to move in and take the black pawn. Black responds by taking the white pawn below with the rook, guarding the capture column.

Proof. Once the white queen positions itself at the bottom of a clauseway, black must immediately move to block that clauseway with a bishop or else the queen can move to the king's diagonal and checkmate the king. If the blocking bishop is unprotected and the queen captures it, then black must move the next bishop to block to again prevent the queen from moving to the king's diagonal. □

Lemma 38. *Once the queen enters the clauseways, white cannot circumvent the clauseways.*

Proof. Assuming the expected line of play, white has $r + 1$ moves left to checkmate the king once the queen is positioned at the bottom of a clauseway. This is precisely

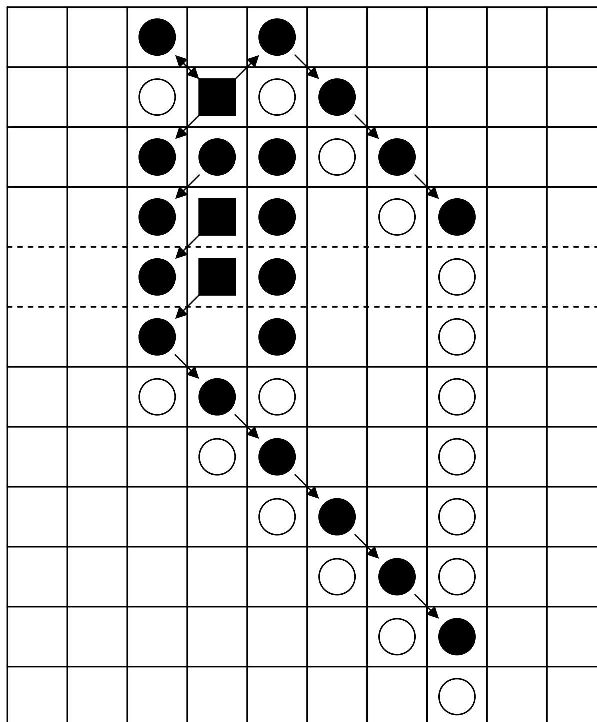

Figure 6.12: The parallelogram gadget.

enough to capture the r bishops available to protect the variable diagonals and then make the one additional move necessary to checkmate the king. While physically possible, white cannot afford to move diagonally to another clause (either now or later while the queen is actually in a clauseway) because it uses up another turn and then black can simply throw all the remaining bishops in the queen's way and white does not have enough turns to capture them all and still checkmate the king.

Attacking the parallelograms between the clauseways and variable diagonals is also not an option. The arrows in the diagram (Fig 6.14) show how every exposed black piece is protected by at least one other piece. □

King's Diagonal. BK is located in a diagonal that runs parallel to the variable diagonals. It is intersected by every clauseway, the capture column, and the alternate capture column. Figure 6.15 shows the king in the diagonal. The king itself cannot move because any move would put it in check. If the white queen is placed on the specific diagonal occupied by the king the result is checkmate. The king cannot move

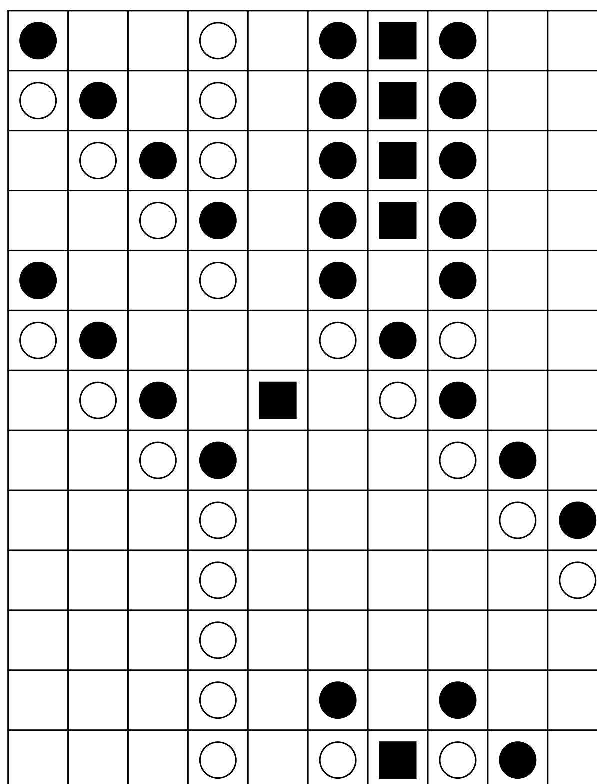

Figure 6.13: Parallelograms to leave the bishop vulnerable to the queen.

out of check, and none of the black pieces can attack the queen in the king's diagonal.[9]

Correctness of the Reduction.

Theorem 10. *White has a winning strategy iff X is not satisfiable.*

Proof. If X is not satisfiable, then for any variable assignment black chooses, there is a variable assignment available for white that results in at least one unsatisfied clause. Once this has been achieved, white can move her queen to the clauseway representing the unsatisfied clause and have enough turns to capture all the bishops and checkmate the king. Black cannot prevent this: we know how black must respond within each gadget, and that failing to respond appropriately results in a victory for white.

If X is satisfiable, then the normal lines of play described above[10] result in a loss

[9] Ignoring the possibility of white purposely placing the queen so that a pawn could capture it.
[10] e.g. setting variables and then looking for a clear clauseway

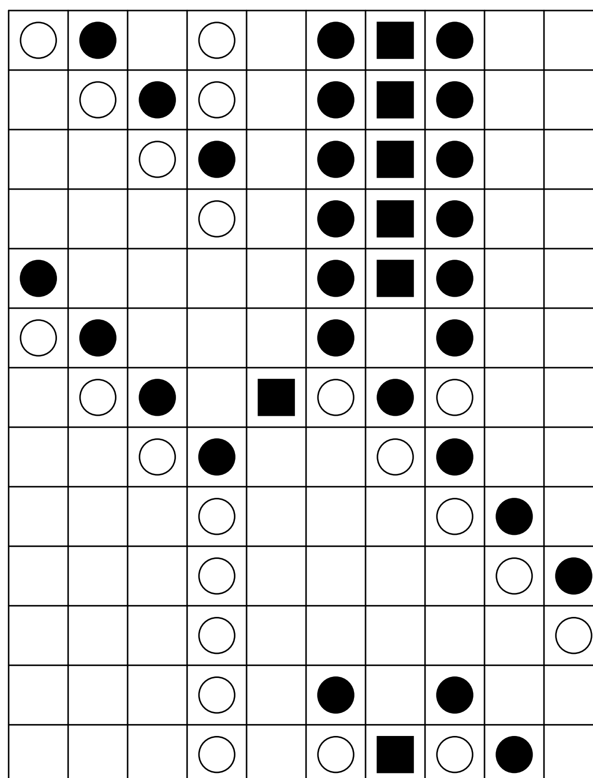

Figure 6.14: Parallelograms to protect the bishop.

for white because black can ensure that every clause has at least one satisfied literal and thus can position bishops such that every clauseway has at least one bishop able to block the white queen for more than one turn. However, we have already proved that white cannot win by attempting to circumvent any of the gadgets, and white cannot attack the pawn walls to escape these gadgets. Thus if black plays optimally, white cannot win within k turns. □

Board Size The size of the board needed by the reduction is tied to the number of pawns used. We can bound the number of pawns needed by estimating the size of the parallelogram produced by the reduction. The height is $n + c + 2$ diagonals, where n is the number of variables and c is the number of clauses in the input instance. Each diagonal is 12 squares high. The width of the board is c clauseways (7 squares), 2 additional vertical paths (5 squares), and the assignment gadget, which cannot be wider than parallelogram is high otherwise it would exceed the bounds of the parallelogram. Thus, the parallelogram is $12(n + c + 2) + 8$ squares high and

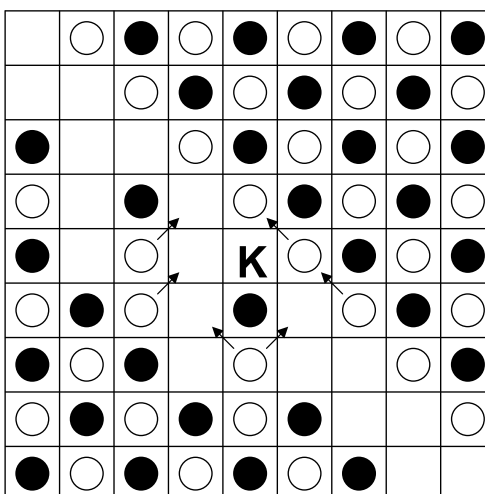

Figure 6.15: The king's position in the king's diagonal. Note that a white piece can attack any square the king can move to.

$19c + 12n + 42$ squares wide. The product gives us a bound on the number of pawns.

6.4 Summary

We have shown that Short Generalized Chess is AW[*]-complete. We note that our reduction is dependent upon having a number of pawns and bishops available which is proportional to the size of the board, so one could devise generalizations of chess for which our reduction does not function. It seems likely that if the gadgets were rotated 45 degrees we could swap bishops and rooks and, with some minor modifications, have a reduction which worked so long as the number of rooks available was proportional to the board.

On the other hand, by restricting the pieces we may be able to derive results regarding endgame play. For example, if the only pieces left on the board are knights and kings, then each piece has at most eight squares to which it can move. Thus, if we include the number of pieces p as a parameter (in addition to the number of turns t) then the size of the game tree we must consider is bounded by $(8p)^t$ and therefore finding a winning strategy is in FPT.

Our analysis of Short Generalized Chess depends on the fact that each player has at their disposal a number of bishops proportional to the width of the board. We suspect that a proof relying on rooks rather than bishops is possible if you were to

rotate all the gadgets by 45 degrees, swap bishops and rooks, and likely make some minor modifications elsewhere. Alternatively, if the number of pawns we get is linear in the size of the board and there are no restrictions on pawn promotion, (besides the obvious prohibition against promoting pawns to kings) then it would seem plausible to argue that we can manufacture the necessary number of bishops. However, we are relatively certain that knights are too limited in their range of movement to enable a reduction. As we mentioned already, an endgame consisting solely of knights and kings in in FPT.

Chapter 7

Parameterized Pursuit

The *pursuit-evasion* game (also known as *cops and robber*) is played on a graph. The cops and robber occupy nodes in the graph and alternate taking turns where they may move to adjacent nodes. The cops win if any cop captures the robber by occupying the same node as him. The robber wins if he or she is able to evade the cops indefinitely.

In this chapter we study different variants and parameterizations of the pursuit-evasion problem. Among other things, this includes parameterized results for winning strategy problems without the short constraint, where the only parameter is the number of pieces.

7.1 Pursuit-Evasion Background

This problem was initially studied for only one cop by Quilliot [55] and independently by Nowakowski and Winkler [52]; both characterized graphs on which one cop was sufficient to capture the robber. Additional cops were added by Aigner and Fromme [5], who showed that three cops are sufficient to capture a robber in a planar graph, and that it is possible construct graphs requiring an arbitrary number of cops to capture a robber. The former result was expanded upon by Quilliot, who proved that $3+2k$ cops are sufficient to capture a robber in a graph of genus k [56]. Goldstein and Reingold considered a seeded variant (i.e., one in which starting positions were given) and showed it to be EXPTIME-complete [34]. Scenarios with multiple robbers were considered in [37]. Surveys of the topic include [8], [36], and [27].

The pursuit-evasion problem and variants of it are connected to many significant

graph properties. If the cops are allowed to choose their starting positions and have only one move to catch a single visible robber, the number of cops needed to ensure capture in a graph G is equal to the size of a minimum dominating set of G [52]. In addition to showing that that three cops are can always catch a single robber in a planar graph, [5] proved that in a graph of girth at least five the number of cops necessary to guarantee capture is at least the minimum vertex degree in that graph. If the robber has infinite velocity (meaning it can move any distance as long as there is a cop-free path from his initial position to its destination), the cops travel by helicopter (each can move from its current vertex to any other vertex without visiting any vertices between), and both sides move simultaneously, the number of cops needed to guarantee capture is equivalent to the treewidth of G [69].

In the first variant of the pursuit-evasion problem we study, SEEDED PURSUIT-EVASION (A.22), the cops and robber both move with a maximum velocity of one, both sides have perfect information (e.g., they can see each other at all times), the starting positions are given (not chosen), and the parameter is the number of cops $|C|$. SEEDED PURSUIT-EVASION is known to be hard for EXP when unparameterized [34]. The parameterized complexity of determining the number of cops necessary to catch the robber when both sides are free to choose their starting position has been shown to be W[2]-hard when parameterized by $|C|$ [26]. We formally define SEEDED PURSUIT-EVASION as follows:

SEEDED PURSUIT-EVASION
Input:

- A simple, undirected graph $G = (V, E)$
- A set $C \subseteq V$ of starting positions for the cops
- Starting position $a \in V, a \notin C$, for the robber

Rules:

1. C defines the starting positions of the cops. Exactly one cop starts at each $v \in C$.
2. The robber begins at vertex a.
3. The cops and robber alternate taking turns, starting with the cops.

4. During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put.[1] Multiple cops may occupy the same node at a time.

5. On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.

6. If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.

7. Both players have complete information. That is, the cops know the positions of the robber and their fellow cops at all times, and vice-versa.

8. You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Parameter: $|C|$
Question: Can the cops guarantee capture of the robber?

We also consider three other variants of this problem: in Short Seeded Pursuit-Evasion (A.31) the cops have only t turns to capture the robber and t is an additional parameter. In Directed Pursuit-Evasion (A.6) the graph is directed and the players choose their starting positions at the beginning of the game. Finally, Short Directed Pursuit-Evasion (A.30) is Directed Pursuit-Evasion with a t-turn limit (as in Short Seeded Pursuit-Evasion).

7.2 Hardness of Seeded Pursuit-Evasion

In this section we prove that Seeded Pursuit-Evasion is AW[*]-hard. We obtain this result by using a parameterized reduction from Unitary Monotone Parameterized QBFSAT$_2$. A parameterized reduction is much like a classical one, but the time permitted to perform the reduction is bounded by a fixed-parameter tractable function rather than a polynomial one, and we must not introduce any aspect of the

[1]Note that we say the cops and robber *may* move, but they can also stay at the vertex they currently occupy. This is equivalent to the variant where all pieces *must* move and every vertex has an edge to itself, since in that variant a piece can stay at its current vertex by moving along the reflexive edge.

input size into the parameters for the target problem [21]. We explain the mechanics of the reduction first; the detailed proof follows after.

7.2.1 Reduction

Let the entire Unitary Monotone Parameterized QBFSAT_2-instance given as input, including the quantifiers, variable sets, weights, and formula F, be referred to as A.

Our strategy is to construct an instance of Seeded Pursuit-Evasion which simulates setting the variables in A and then tests the setting on the formula F. This means the cops can guarantee capture of the robber if and only if they can produce a satisfying assignment in the simulated setting. Therefore, the cops are guaranteed to catch the robber in the reduced instance if and only if A is satisfiable. For our parameter, we set $|C| = r + 1$.

Our reduction employs four component gadgets: runways, the assignment gadget, a set of clause vertices, and escape subgraphs. *Escape subgraphs* enable the robber to evade the cops indefinitely (in other words, the robber wins the game if he enters an escape subgraph), *runways* simulate the variables of A, the *assignment gadget* enables the robber to effectively set the values of the (simulated) universally-quantified variables, and the *clause vertices* enable the robber to test whether a clause is satisfied by the simulated variable setting. The starting positions for the cops and the robber are defined in the gadgets.

The terms *move* and *turn* are used interchangeably in this reduction. Both mean that one player makes his full move. In the case of the robber this entails moving from his current node to an adjacent one (or choosing to stay put). In the case of the cops, this means that *all* of the cops have either moved or committed to staying put. A *round* consists of two turns: one for all the cops, followed by a turn for the robber.

We now describe the construction of each gadget in detail.

Escape Subgraphs

The sole purpose of the escape subgraphs is to provide the robber with a place in which he can evade the cops indefinitely. Any graph of fixed-parameter-tractable size in which a robber could perpetually evade $r + 1$ cops would suffice for our purpose. We use hypercubes.

Definition 19. *The* hypercube graph of degree *n, Q_n, is the graph whose vertices lie on the corners of the n-dimensional unit cube. An edge exists between two vertices in Q_n if and only if their coordinates as expressed in binary differ in exactly one position.*

To ensure that the hypercube fulfills its function as an escape subgraph we rely on the following corollary, which is proved in Section 7.3.

Corollary 3: A robber can evade k cops indefinitely in Q_{2k}.

We use hypercubes of degree $2r + 2$ as escape subgraphs in our reduction, since the number of cops is the constructed instance is $r + 1$.

Runways

For each quantifier in A we create a corresponding runway in our SEEDED PURSUIT-EVASION instance (see Figure 7.1). The runway corresponding to the ith quantifier of A consists of a path of length i which then forks into $|S_i|$ paths, making the number of paths equal to the number of variables associated with the ith quantifier. For each of the variables associated with the quantifier, there is a path of length $r - i + 1$; we call these paths the *forks* of the runway. Moving the cop into a fork corresponds to a setting in which the variable associated with the fork is true and all the other variables in S_i are false. We refer to the first vertex of the length–i path as the *start* of the runway, while the fork vertices furthest from this start are referred to as *end vertices* of the runway. A single cop begins at the start of the runway (see Figure 7.1), and the distance the cop must travel from the start to any end of that runway is at least r.

The Assignment Gadget

We construct the assignment gadget starting from a path of length $r+2$ (be reminded that r is the number of quantifiers). There is one cop, which starts on the first vertex of this path, while the robber starts on the third. The remaining r vertices correspond to the quantifiers of A; specifically, vertex $i+2$ corresponds to quantifier i. Quantifiers are indexed from 1, so the robber's starting position corresponds to the first quantifier. Each vertex which corresponds to a universal quantifier (which, by the definition of UNARY PARAMETERIZED QBFSAT$_2$, is every even-numbered quantifier) is replaced with a clique of size equal $|S_i|$. We call these *assignment cliques*. For an example of an assignment gadget, we refer to Figure 7.2.

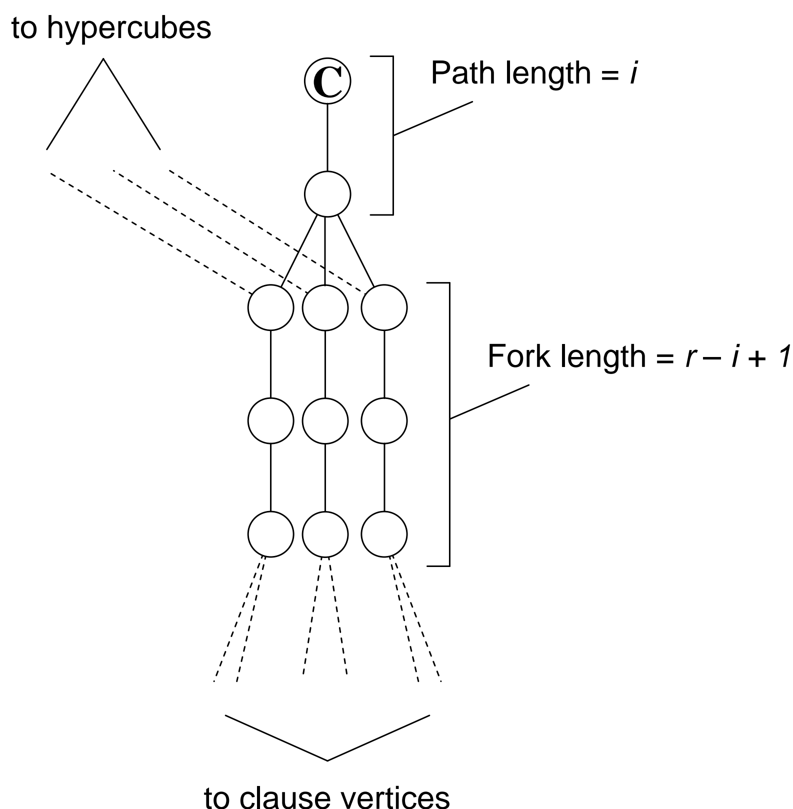

Figure 7.1: Runway gadget construction for the ith quantifier. The cop (C) is shown in his starting position. Note that there is a one-to-one correspondence between the forks and the variables of S_i.

Next, we connect the assignment cliques with the runway forks. We do this as follows: for the assignment clique corresponding to quantifier i, assign to each vertex a unique variable in S_i (this results in a one-to-one mapping from the elements of S_i to the clique's vertices). For each vertex v in the assignment clique, create a unique escape subgraph Q^v_{2r+2}. Choose a vertex $w \in Q^v_{2r+2}$ and add an edge connecting w to v and an edge connecting w to the first vertex of the runway fork from the ith runway which corresponds to the same variable as v. Figure 7.4 shows an example of this construction.

Clause Vertices

Finally we describe the clause vertices (shown in Figure 7.3). Each of these corresponds to a clause in F. Each clause vertex has an edge to every runway fork which

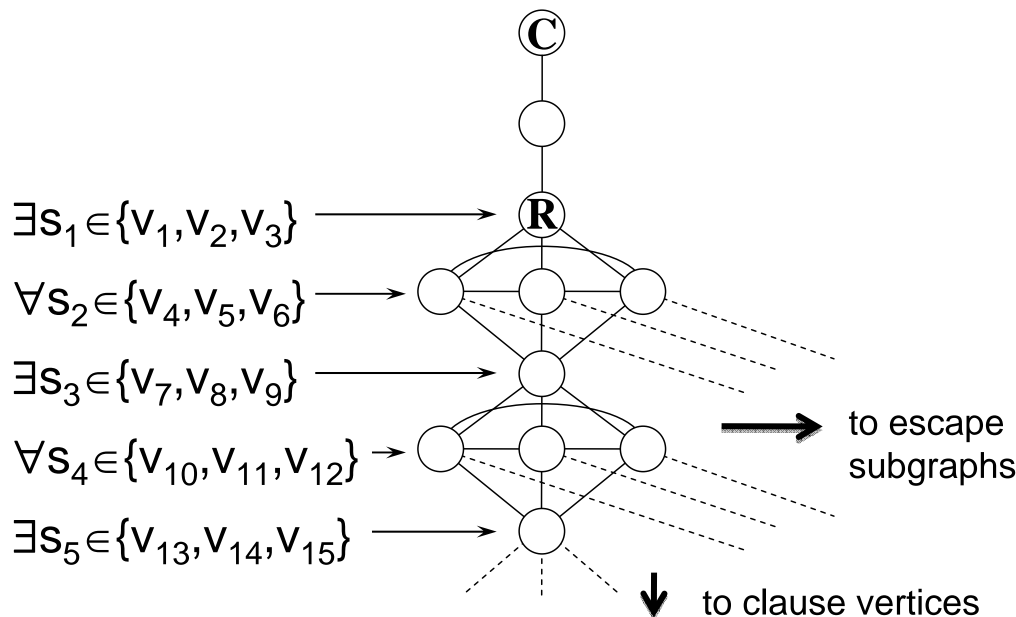

Figure 7.2: An example of the assignment gadget for $r = 5$. The cop (C) and robber (R) are shown in their starting positions.

represents at least one literal that satisfies the vertex's corresponding clause. Every clause vertex is also connected to the end of the assignment gadget (the opposite end of the path from where the cop starts). If the last quantifier in A is existential, then the end of the assignment gadget is a single vertex. On the other hand, if the last quantifier in A is universal, then the end of the assignment gadget is a clique and each clause vertex is connected to every vertex in that clique. Lastly, we create a unique escape subgraph for each clause vertex and connect each clause vertex to a single vertex of its unique escape graph.

7.2.2 Sequence of Play

The general idea of this reduction is that the runways simulate the quantifiers. For existential quantifiers, the cops are free to choose the true variable for that quantifier, while for universal quantifiers the connections between the assignment gadget and the runway enable the robber to manipulate the choice of variable. Our goal is that when the robber reaches the clause vertices, he will have a safe path to an escape subgraph if and only if there is an unsatisfied clause. For an example of an instance constructed by this reduction, see Figure 7.3.

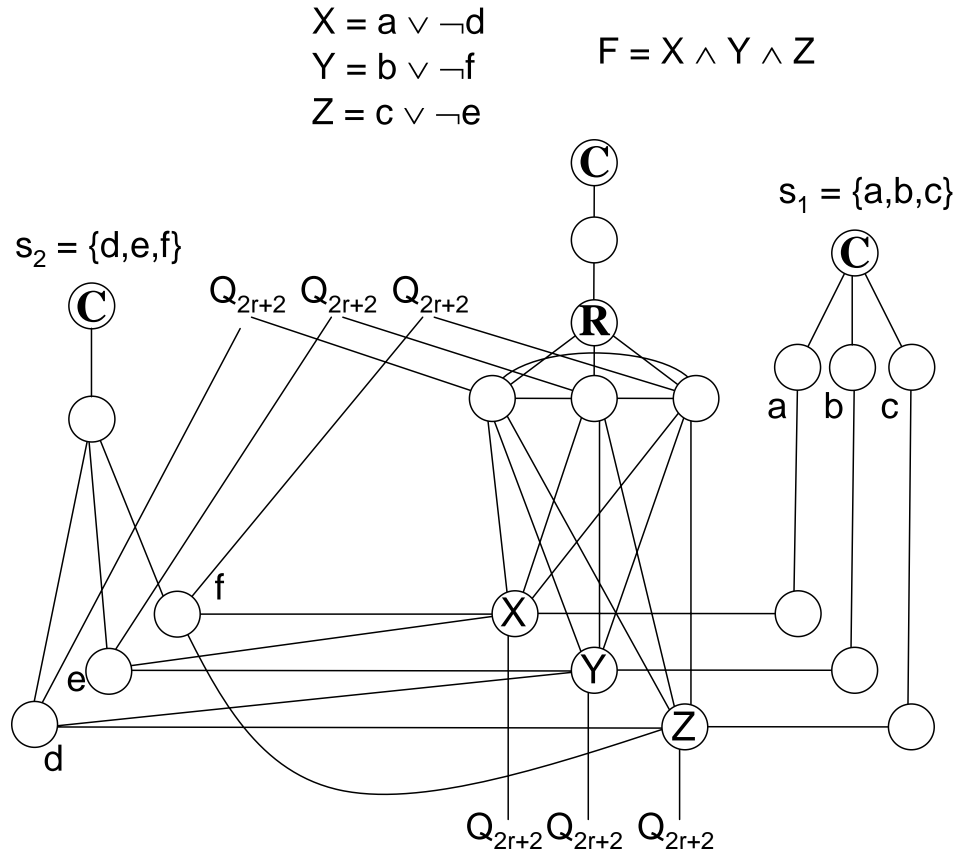

Figure 7.3: A SEEDED PURSUIT-EVASION instance constructed by our reduction from the UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ instance $S_1 = \{a, b, c\}, S_2 = \{d, e, f\}, F = (a \wedge \overline{d}) \vee (b \wedge \overline{f}) \vee (c \wedge \overline{e})$. The cops (C) and robber (R) are shown in their starting positions.

When a cop chooses a fork to follow, this can be viewed as setting the variable corresponding to that fork to true while setting the rest of the variables in the same quantifier set S_i to false. In runways corresponding to existential quantifiers, the cops position themselves as they choose. For universal quantifiers, the runway cop is forced to block the robber: when the robber positions himself in one of the vertices in an assignment clique, the cop must move into the corresponding fork (thus enabling him to catch the robber with his next move should the robber attempt to enter the escape subgraph) or the robber will be able to move into the adjacent escape subgraph where he can elude the cops indefinitely.

Once all the variables are set, the robber moves from the end of the assignment gadget into a clause vertex. Since the clause vertices are connected to the runway

forks, the cops will be able to capture the robber if he moves into a vertex corresponding to a satisfied clause. However, if he moves into an vertex corresponding to an unsatisfied clause, then the cops will be unable to catch him on their next turn and he can flee into an escape subgraph on his subsequent turn.

Proving the Sequence of Play

We now show the correctness of our reduction with a sequence of lemmas. The first four are independent of instance A, while the last three depend on the satisfiability of A.

Lemma 39. *The cops can either capture the robber or force him into a clause vertex on round r.*

Proof. The cop in the assignment gadget starts only two vertices away from the robber (see Figure 7.2) and moves first. Thus if this cop always moves toward the robber, the robber must always move away from the cop. If the cops in the runways corresponding to universally-quantified variables move to guard the escape subgraphs adjacent to the assignment cliques, then the robber cannot move into any of them because he would be captured. Thus the robber can only leave the assignment gadget by entering a clause vertex and he will do so in round r, since that is the distance from his starting position to any clause vertex. □

Lemma 40. *The robber can make it into any clause vertex on round r without getting captured, regardless of the cops' strategy.*

Proof. As this proof depends heavily on distances to clause vertices, Figure 7.4 may be of assistance.

By our definition for the assignment gadget, the robber starts at distance r from the clause vertices. Let us assume the robber's evasion strategy decreases his distance from the clause vertices each turn, since the cop in the assignment gadget can catch the robber otherwise. The cop in the assignment gadget which starts at distance $r+2$ from the clause vertices cannot catch the robber in the first r turns since the cop's distance to the clause vertices will always be greater than the robber's. The cop does have the advantage of moving first, but this only enables him to temporarily close the gap since the robber starts out two vertices ahead.

If a runway cop moves to the end of his runway and enters the clause vertices from there, then the distance covered is at least $r+1$. Further, his distance from

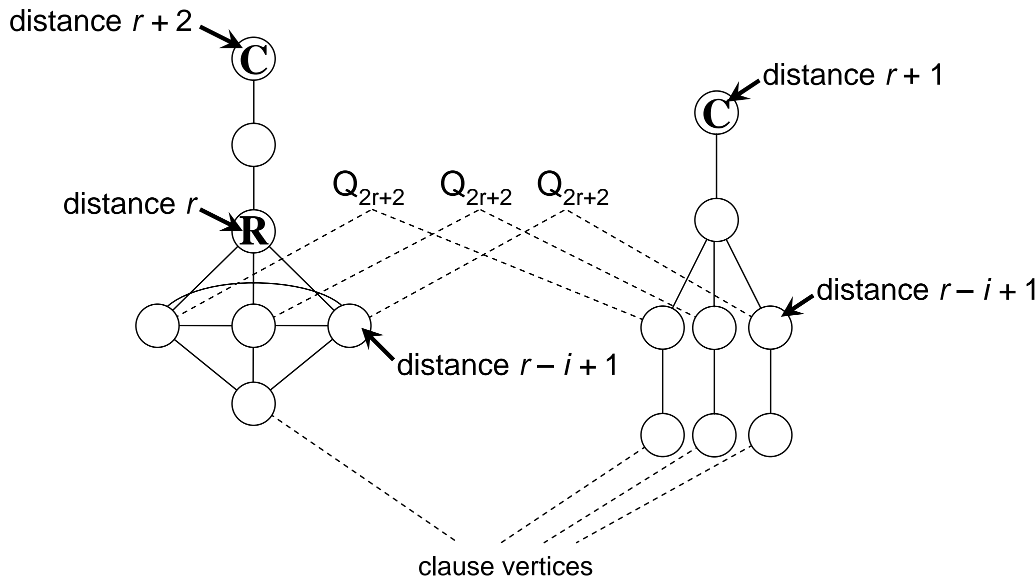

Figure 7.4: Distances from the clause vertices in the assignment gadget and runways.

the clause vertices will always be higher than the robber's at the end of each of the first r rounds. Thus, the only remaining possible way a cop could capture the robber before round $r+1$ is if a runway cop can find a shorter path to the clause vertices by leaving the runway early and entering the assignment gadget. However, this is not the case: for any universal quantifier,[2] if that quantifier is numbered i then both the assignment clique and the connected vertices in the associated runway forks are at distance $r-i+1$ from the clause vertices. It takes two turns to move between the runway forks and the assignment gadget, but these moves do not decrease the cop's distance to the clause vertices and thus these moves are lateral and not part of any shortest path to the clause vertices. Even with the advantage of moving before the robber, a cop using this move would arrive in the assignment gadget at distance $r-i+1$ from the clause vertices on turn $i+2$, while the robber would be at distance $r-(i+1) = r-i-1$ from the clause vertices and would also be next to move.

Thus, any shortest path to the clause vertices for a runway cop is of length at least $r+1$ and no runway cop can catch the robber in the first r turns even if they leave their runways. This leaves no other cop which could possibly catch the robber. □

[2] We do not consider existential quantifiers as their runways are not connected to the assignment gadget – only the clause vertices. Thus the cops in these runways can only leave through clause vertices.

Lemma 41. *If the robber is on a clause vertex v at the end of round r, the cops can capture the robber if and only if there is a cop at the end of a runway fork which corresponds to a literal that satisfies the clause B corresponding to v.*

Proof. By definition, each runway fork corresponds to a set of literals. The end of each runway fork is connected to every clause vertex for which the associated clause contains at least one of its associated literals. Therefore, if there is a cop at the end of a runway fork corresponding to a literal b that satisfies B, then that cop can simply follow the edge to v and capture the robber.

On the other hand, let us assume that no cop is at the end of a runway fork corresponding to a literal that satisfies B. Other than runway forks, the only vertices adjacent to v are those at the end of the assignment gadget and those in the adjacent hypercube. To show that there is no cop adjacent to v, observe that no cop can reach the hypercube in r rounds since it is distance $r + 2$ from the starting position for runway cops and $r + 3$ from the starting position for the cop in the assignment gadget. Similarly, the cop in the assignment gadget starts at distance $r + 2$ from the clause vertices, and the runway cops start at distance $r + 3$ from the end of the assignment gadget. □

Lemma 42. *Once a runway cop has entered a runway fork, he cannot reach the end of any other runway fork by the end of round r.*

Proof. As noted in the definition of the runway gadget, the cop starts at distance r from the ends of the runways. Once he enters a fork, then he increases the distance to the end of every other fork. Thus his distances to the ends of those forks are more than the number of rounds he has left. □

The following lemma shows that when the robber enters an assignment clique, he chooses which fork the cop in the associated runway must start down. This renders the cop unable to reach the end of any other fork.

Lemma 43. *For runways corresponding to universally-quantified variables, the robber decides which fork the cop can reach the end of.*

Proof. When the robber enters a vertex v in an assignment clique,[3] the cop in the runway associated with that clique must move into u, the first vertex of the runway fork associated with v. This prevents the robber from moving into the escape

[3]Remember that we only use assignment cliques when the associated quantifier is universal.

subgraph adjacent to v, since both u and v are adjacent to the same vertex in the escape subgraph (see Figures 7.3 and 7.4). Otherwise, if the cop does not move in this manner the robber can move into the escape subgraph unimpeded and evade the cops indefinitely once in it.

The cop cannot undo the robber's selection by moving back and into another fork as he will be unable to reach the end of another fork with his remaining turns in such a situation by Lemma 42. □

Lemma 44. *If A is satisfiable, then the cops can guarantee capture of the robber.*

Proof. By Lemma 39 the cops can force the robber into a clause vertex in round r (or capture him if he does not comply). Since A is satisfiable, it it possible to create a satisfying assignment taking into account how the universal variables get set. Therefore, regardless of how the robber manipulates the cops in the runways corresponding to universal quantifiers, the cops can choose runway forks in the runways corresponding to the existential quantifiers of a satisfying assignment.

Assuming the cops have played following a satisfying assignment, and every cop has moved to the end of his runway fork, there must be at least one cop adjacent to each clause vertex. This is because the set of literals corresponding to the runways occupied by the cops satisfies F, which means that, by definition, for each clause vertex, there is at least one adjacent runway end (representing a literal which satisfies the clause) which is occupied by a cop. □

Lemma 45. *If A is unsatisfiable, then the robber can evade the cops indefinitely regardless of their strategy.*

Proof. By Lemma 40, the robber can get to a clause vertex safely in round r.

By Lemma 41, if the cop is in clause vertex v corresponding to clause B, he can survive round r and enter the adjacent escape subgraph in round $r+1$ (where he can evade the cops indefinitely) if and only if there is no cop at the end of any runway corresponding to a literal which satisfies B.

By Lemma 43, the robber can decide for each universal variables which fork the runway cop can reach the end of. Since A is unsatisfiable, he can force the cops into runway forks such that the set of literals corresponding to the runways the cops can reach the ends of does not satisfy F. Therefore, even if the cops move to the ends of their runways they will not be able to cover every clause vertex. The robber can survive round r in that clause vertex and then move into the adjacent escape subgraph on his next turn, guaranteeing that he cannot be captured. □

7.2.3 Correctness

With the following proof, we conclude our hardness result:

Theorem 11. Seeded Pursuit-Evasion *is AW[*]-hard.*

Proof. Lemmas 44 and 45 show that the cops can force a win in the constructed instance if and only if A is satisfiable. All that remains to show is that our reduction is parameterized.

If A_C is the number of clauses in A and A_V is the number of variables in A, then for the number of vertices in the constructed instance we have:

- $A_C + A_V$ hypercubes, each of size 2^{2r+2}.
- A_V runways, each of size at most $A_V \cdot (r+1)$.
- A_C clause vertices, each of size 1.
- one assignment gadget of size at most $A_V + 2$.

While the size of the constructed instance is not polynomial in the size of the input because of the exponentially-large hypercubes, the size of the hypercubes is dependent only on r, the parameter. There are only a polynomial number of hypercubes (and other gadgets) so the size of the graph as a whole remains fixed-parameter tractable. The parameter in our target reduction is $|C|$, which is equal to $r+1$ (one cop in each of the r runways, and one in the assignment gadget), so our new parameter remains independent of the input size. Thus we have a parameterized reduction from Unitary Monotone Parameterized QBFSAT$_2$ to Seeded Pursuit Evasion. □

7.3 Pursuit in a Hypercube

We have omitted one crucial proof from the reduction in our previous section: that our robber can evade the cops indefinitely once he enters a hypercube. We provide that proof in this section.

We use coordinates of the hypercube (which are all 0 or 1) as a mapping from vertices to subsets of an n-element set. In this case, an edge between two vertices exists if and only if the sets corresponding to those vertices differ by exactly one element.

We now define a notion of a local dominating set, which can be seen as a generalization of the notion of a *cover* in [52]. Note that we use $N[v]$ to denote the inclusive neighbourhood of vertex v, while $N(v)$ denotes the exclusive neighbourhood.

Definition 20. *A graph $G = (V, E)$ has a* local dominating set *of size k if and only if there exists some $v \in V$ and $V' \subseteq V - \{v\}$ such that $|V'| \leq k$ and V' dominates $N[v]$.*

Note that V' must always contain at least one vertex of $N(v)$ in order to dominate v. You always have a local dominating set when k is larger than or equal to the degree of the lowest-degree vertex in the graph, since for that vertex the local dominating set can consist of all its neighbours.

If a graph does not have a local dominating set of size k (meaning that no vertex in the graph meets the criteria given above) then the robber can evade k cops as follows:

Lemma 46. *If a graph G does not have a local dominating set of size k, then a single robber can evade k cops in G indefinitely.*

Proof. If graph $G = (V, E)$ has no local dominating set of size k and the robber is on a vertex v, then by definition the k cops will not be able to position themselves in such a way that they are on or adjacent to every vertex in $N[v]$. Thus, the robber can escape to an undominated vertex and therefore the cops will not be able to capture him on their next turn. Since this property applies to every vertex in the graph, the robber can evade the cops indefinitely. □

All that remains is for us to show that the hypercube Q_{2k} does not have a local dominating set of size k, thus allowing the robber to employ this simple evasion strategy.

Theorem 12. *Q_{2k} has no local dominating set of size k.*

Proof. We associate with each vertex a length $2k$ bit string – namely its coordinates on the $2k$-dimensional unit cube. In this representation, two vertices in Q_{2k} are adjacent if and only if their bit strings differ in exactly one position. When we refer to a weight-x vertex, we mean a vertex whose bit string has Hamming weight x.

Now, assume without loss of generalization that vertex v is represented by the all-zero string. $N[v]$ – the inclusive neighbourhood of v – consists of all $2k$ weight–1

vertices and the weight–0 vertex (v) itself. Any weight–1 vertex dominates itself and v. Any weight–2 vertex also dominates exactly two weight–1 vertices (its two weight–1 neighbours being the two strings you can get by changing either of the weight–2 bitstring's ones to zeros). You are not allowed to use the weight–0 vertex since it is v, and vertices of weight 3 or higher are not adjacent to any relevant vertices. Thus, all vertices besides v dominate at most two vertices of $N[v]$.

Since each vertex can dominate at most 2 elements of $N[v]$, a set of k vertices can dominate at most $2k$ vertices. However, $N[v]$ contains $2k+1$ vertices and therefore k vertices are insufficient to dominate it. □

Since, by Theorem 12, Q_{2k} has no local dominating set, and by Lemma 46 a robber can evade k cops indefinitely in a graph with no local dominating set of size k, we can derive the result we need for our reduction:

Corollary 3. *A robber can evade k cops indefinitely in Q_{2k}.*

7.4 Short Seeded Pursuit-Evasion

In [3], Abrahamson, Downey, and Fellows sought to parameterize games by considering *short* variants which ask whether the player to move can force a win within the next t turns. This new input t becomes the parameter. Abrahamson, Downey, and Fellows applied this approach to Generalized Geography and Node Kayles in [3] and showed the resulting problems to be AW[*]-hard for parameter t. Downey and Fellows later conjectured that AW[*] was the natural home of k-move (short) games [21]. In this section, we now apply this technique to Seeded Pursuit-Evasion.

Short Seeded Pursuit-Evasion
Input:

- A simple, undirected graph $G = (V, E)$
- A set $C \subseteq V$ of starting positions for the cops.
- Starting position $a \in V$ for the robber.
- Positive integer t.

Rules: *As for* Seeded Pursuit-Evasion.
Parameters: t, $|C|$
Question: Can the cops guarantee capture of the robber within t moves?

We parameterize by both t and $|C|$ because our hardness result then holds for any subset of those parameters (e.g. for parameterizing by t or by $|C|$ alone).

7.4.1 Hardness of Short Seeded Pursuit-Evasion

We use our reduction from Section 7.2 again to inform us as to the parameterized complexity of SHORT PURSUIT EVASION:

Lemma 47. SHORT SEEDED PURSUIT-EVASION *is AW[*]-hard.*

Proof. We reuse the reduction from UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ to SEEDED PURSUIT-EVASION. By Lemma 44, if A (the original UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ instance) is satisfiable then the cops have a winning strategy. The winning strategy given in that lemma takes at most $r + 1$ rounds to execute. In all other cases (that is, when A is not satisfiable) the cops do not have a winning strategy, as per Lemma 45. Since r is the original parameter of A, setting $t = r + 1$ is both correct and parameterized. Thus we have a parameterized reduction to SHORT SEEDED PURSUIT-EVASION and can conclude that the problem is AW[*]-hard. □

Note that a path of length t can be used as an escape subgraph for this problem. Thus we could replace the hypercubes with paths and have a polynomial reduction from UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ to this problem (whereas our reduction for SEEDED PURSUIT-EVASION only qualifies as parameterized). This technique would, in fact, be necessary to ensure a parameterized reduction when t is our only parameter.

7.4.2 Membership of Short Seeded Pursuit-Evasion

We reduce SHORT SEEDED PURSUIT-EVASION to the AW[*]-complete problem PARAMETERIZED QBFSAT$_t$ [3]. Once again, we first outline our parameterized reduction, then follow with a proof of correctness.[4]

Given an instance of SHORT SEEDED PURSUIT-EVASION, our goal is to create a formula F which encodes the rules of the game and is satisfied if and only if there is a winning strategy for the cops. As before, this must be a parameterized reduction.

We first create two sets of variables to represent the positions of the cops and robber:

[4] A preliminary version of this proof appeared in [67].

- c_{dvj} is true if and only if cop d is at vertex v in round j.
- r_{vj} is true if and only if the robber is at vertex v in round j.

We use the quantifiers to capture the alternating nature of the pursuit problem. Since the cops must choose their own moves, we use the existential quantifiers for the cops' moves, and since a winning strategy for the cops must take into account any possible move by the robber we use the universal quantifiers for his moves. Thus, for the existential quantifiers where i is odd i, we set $S_i = \{c_{dv((i+1)/2)} | 1 \leq d \leq |C|, v \in V\}$ and $k_i = |C|$. For universal quantifiers where i is even, we set $S_i = \{r_{v(i/2)} | v \in V\}$ and $k_i = 1$.

The k_i parameters cause the correct number of cops to occupy vertices. However, this does not preclude the possibility of one cop occupying multiple vertices while others occupy none. We will prevent this by adding a rule for the cops to our formula F (specifically R_5 below).

We use formula F to encode the movement rules and the winning condition. We write the robber's constraint rules such that they are satisfied by any variable setting which corresponds to a game in which the robber either violated a rule or was captured – in other words, a cop-win. Thus the robber's rules are negated so that a violation results in a clause being true. These clauses are then arranged in disjunctive normal form so that if any clause is true the entire formula is true as well. For the cops we want a single violation to result in the entire formula evaluating to false, so their clauses are arranged in CNF.

We now restate the rules of SHORT SEEDED PURSUIT-EVASION, encode them as formulas, and prove that these encodings properly enforce them. Negations are represented with overlines (e.g., the negation of x is $\overline{x}$).

Rule 1: Exactly one cop starts at each $v \in C$.
Implementation: $R_1 = \bigwedge_{v \in C} c_{f(v)v0}$, where $f(v)$ as an arbitrary one-to-one function that maps vertices to the set $\{1, \ldots, |C|\}$.

Proof. By definition, $c_{f(v)v0}$ is true if and only if cop $f(v)$ is on vertex v on turn 0. Therefore, R_1 is true if and only if the cops started on all the vertices in C. □

Rule 2: The robber begins at vertex a.
Implementation: $R_2 = \overline{r}_{a0}$

Proof. By definition, r_{a0} is true if and only if the robber is on vertex a on turn 0. Thus R_2 is true if and only if the robber starts on some vertex other than a. □

Rule 3: The cops and robber alternate taking turns, starting with the cops.

Proof. The cops' turns correspond to existential quantifiers and the robber's turns correspond to universal quantifiers, which alternate as desired. For each existential quantifier i is odd and S_i is the complete set of variables for cop positions on the corresponding turn. For each universal quantifier i is even and S_i is the complete set of variables for robber positions on the corresponding turn. □

Rule 4: During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put. Multiple cops may occupy the same node at a time.
Implementation: $R_4 = \bigwedge_{\substack{(u,v)\notin E, u\neq v \\ 1\leq s\leq t \\ 1\leq i\leq |C|}} (\overline{c}_{iu(s-1)} \vee \overline{c}_{ivs})$

Proof. It may be easier to see the correspondence with R_4 if we state Rule 4 as its complement: namely that no cop may move to a non-adjacent vertex.

Multiple cops (say i and j) occupying the same node is accomplished by setting both c_ivs and c_jvs to true. In essence, we permit this behaviour by never disallowing it. □

The following rule is the robber's analogue to Rule 4.

Rule 5: On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.
Implementation: $R_5 = \bigvee_{\substack{(u,v)\notin E, u\neq v \\ 1\leq s\leq t}} (r_{u(s-1)} \wedge r_{vs})$

Now we add the winning conditions for the cops. For implementation, these are lumped in with the robber's rules and thus phrased in DNF.

Rule 6: If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.

For simplicity, we split this rule into two:

Rule 6a: The cops win if a cop enters the vertex occupied by the robber.
Implementation: $R_{6a} = \bigvee_{\substack{v\in V \\ 1\leq s\leq t \\ 1\leq i\leq |C|}} (r_{v(s-1)} \wedge c_{ivs})$

Proof. Since the cops move first, they can capture the robber by entering the vertex he was in at the end of the last round $(s-1)$. Thus, if the robber ends round $s-1$ on vertex v, the cops can capture him if and only if some cop i (for any i such that $1 \leq i \leq |C|$) enters vertex v in round s. This is true if and only if $r_{v(s-1)} \wedge c_{ivs}$ is true. Thus R_{6a} is true if and only is a cop moves into the robber's vertex. □

As an alternative to the situation in R_{6a}, the robber may move into a cop's position. This, of course, gets him caught and thus we must enforce this possibility as well.

Rule 6b: The cops win if the robber enters vertex occupied by a cop.

Implementation: $R_{6b} = \bigvee_{\substack{v \in V \\ 1<s\leq t \\ 1\leq i\leq |C|}} (r_{vs} \wedge c_{ivs})$

The proof of this rule is similar to that of R_{6a}, except that since the robber moved last we check his position in round s instead of $s-1$.

Rule 7: Both players have complete information.

Proof. This is due to the fact that the quantifiers are resolved sequentially in a left-to-right manner. That is, when it is time to resolve the ith quantifier, the variables corresponding to quantifiers 1 through $i-1$ have already been decided and are known. Thus complete information is encoded into the formula. □

Rule 8: You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Implementation: $R_8 = \bigwedge_{\substack{u,v \in V \\ u \neq v \\ 1\leq s\leq t \\ 1\leq i\leq |C|}} (\overline{c}_{ius} \vee \overline{c}_{ivs})$

Proof. The first part of this statement is enforced by the quantifier weights. For every universal quantifier numbered i (robber's turns), we defined the weight k_i to always be 1 and the set S_i to be the set of all $r_{v(i/2)}$. Thus, for any given round t, exactly one of the robber variables r_{vt} is true, indicating that the robber is on vertex v.

Similarly, for the existential quantifiers (cops' turns) we defined the weight to always be $|C|$. However, this is not sufficient to ensure that all $|C|$ cops are placed as it is possible that the variable assignment could set one cop in multiple positions while another is not given any position. Rule R_8 ensures that no cop occurs occurs twice: if a cop occupies two vertices, say u and v, on turn i then the clause containing

both c_{ius} and c_{ivs} is false and thus R_8 would be false. If no cop occupies two vertices on the same turn, then for any turn i and any pair of vertices u, v, one of c_{ius} and c_{ivs} is false. Thus, every clause of R_8 would be true, rendering the entire formula true. Therefore, R_8 is true if and only if no cop occupies two different vertices on the same turn. This means that there must be $|C|$ different cops placed, and thus every cop is placed once. □

Now we assemble the formulas for the cops and the robber:

Robber's Formula:	$F_R = (R_2 \vee R_5 \vee R_{6a} \vee R_{6b})$
Cops' Formula:	$F_C = (R_1 \wedge R_4 \wedge R_8)$

Combining these, we get the formula F for our PARAMETERIZED QBFSAT$_t$ instance: $F = F_R \wedge F_C$

Given the preceding sequence of proofs, we also have:

Corollary 4. *Our constructed* PARAMETERIZED QBFSAT$_t$ *instance encodes all the rules of* SHORT SEEDED PURSUIT EVASION.

Now we are ready to state the main result of this section.

Theorem 13. SHORT SEEDED PURSUIT-EVASION *is in AW[*].*

Proof. We show that F is true if and only if the cops won the game, either by capturing the robber in a legal game or because the robber violated a rule.

As we have already verified the rules individually above, the robber's formula ($F_R = R_2 \vee R_5 \vee R_{6a} \vee R_{6b}$) is true if at least one of the following is true: he chooses to start somewhere other than his assigned starting position (R_1), performs an illegal move (R_5), or is caught (R_{6a}, R_{6b}). Otherwise, if the robber moves legally without being captured, F_R evaluates to false for the corresponding variable setting. Similarly, the cops' formula ($F_C = R_1 \wedge R_4 \wedge R_8$) is false if and only if they break a rule. Thus formula F is satisfied if the cops move legally (F_C is true) and the robber is caught or he violates a rule (F_R is true).

Conversely, F is unsatisfied if the cops move illegally (F_C is false) or the robber moves legally and is not caught (F_R is false). Note that the formula is still unsatisfied if the cops violate a rule and the robber is captured since F_C would be false under those conditions. The formula is polynomial in size, and does not introduce the input size into the parameters, so our reduction is complete. □

We can also use this to derive a result for SHORT PURSUIT-EVASION, defined as per SHORT SEEDED PURSUIT-EVASION except that the players choose their starting positions with their first moves.

Lemma 48. SHORT PURSUIT-EVASION *is in AW[*].*

Proof. Since the only difference between SHORT PURSUIT-EVASION and SHORT SEEDED PURSUIT-EVASION is the ability to choose starting positions, we can take the previous reduction and turn it into a reduction to SHORT PURSUIT-EVASION by removing the rules regarding starting positions from the formula, namely R_1 and R_2. □

Finally, as we have shown that SHORT SEEDED PURSUIT-EVASION is in AW[*] (Theorem 13) and AW[*]-hard (Lemma 47), we have the following result:

Theorem 14. SHORT SEEDED PURSUIT-EVASION *is AW[*]-complete.*

7.5 Directed Pursuit-Evasion

In this section we consider DIRECTED PURSUIT-EVASION, an unseeded variant of the SEEDED PURSUIT-EVASION problem where the edges of the input graph are directed and the players choose their starting positions. In [34], Goldstein and Reingold showed this problem to be hard for EXP.

DIRECTED PURSUIT-EVASION
Input: directed graph $G = (V, E)$, positive integer k.
Rules:

- At the beginning of the game, each of the k cops chooses a vertex of G to be deployed to.
- Once the cops are deployed, the robber chooses a vertex in V to start on.
- For the remainder of the game the cops and robber alternate taking turns, starting with the cops.
- During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put. Multiple cops may occupy the same node at a time.

- On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.
- If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.
- Both players have complete information. That is, the cops know the positions of the robber and their fellow cops at all times, and vice-versa.
- You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Parameter: k
Question: Can the cops guarantee capture of the robber?

We show that this problem is AW[*]-hard by extending our previous reduction from Section 7.2. Specifically, we take the output instance from that reduction (from Unitary Monotone Parameterized QBFSAT$_2$ to Seeded Pursuit-Evasion) and create an instance of Directed Pursuit-Evasion.

Given a Seeded Pursuit-Evasion instance generated by our reduction, we add gadgets and direct the edges resulting in a Directed Pursuit-Evasion instance in which the players will lose if they do not move into the vertices corresponding to the starting position(s) of the Seeded Pursuit-Evasion instance. We avoid modifying the rest of the graph so that the remainder of the game plays out as before. The reduction creates G' from G as follows:

- Let s be the vertex in G' corresponding to the starting position in G for the cop in the assignment gadget as given by the Seeded Pursuit-Evasion instance (see Figure 7.2).
- Let r' be the vertex between r and s (again, see Figure 7.2). Let $U = V - C - r'$.
- For each vertex $v \in C$ (the set of starting positions for the cops), change all the undirected edges incident to v into out-edges.
- Create a set of new vertices D with $|D| = |C|$.
- Add directed edges to G' such that each vertex in D has one out-edge to a unique vertex in C (that is, a one-to-one mapping).

- Let s' be the vertex in D that is adjacent to s.
- Add directed edges from each vertex in $D - \{s'\}$ to every vertex in U.
- For each $v \in C - \{s\}$, create an escape subgraph called R_1^v. Let $R_1 = \{R_1^v\colon v \in C - \{s\}\}$.
- For each $v \in D - \{s'\}$, create an escape subgraph called and R_2^v. Let $R_2 = \{R_2^v\colon v \in D - \{s'\}\}$.
- For each R_1^v in R_1, pick a vertex $w \in R_1^v$ and add directed edges from r' and v to w.
- Add directed edges from s' to every vertex in each escape subgraph in R_1.
- For each R_2^v in R_2, pick a vertex $w \in R_2^v$ and add directed edges from s and r' to w.
- For each R_2^v in R_2, add directed edges from v to every vertex of R_2^v.

Figure 7.5 illustrates how the components of G' are connected.

Besides the graph, we also adjust the rest of the input from Seeded Pursuit-Evasion. While we do not input the set C, we set $k = |C|$. As the robber can choose his starting position, we omit vertex a from the input.

Theorem 15. Directed Pursuit-Evasion *is AW[*]-hard.*

Proof. This reduction forces the cops to start on the vertices of D: If when the cops deploy, there is some vertex $v \in D$ which is is not occupied by a cop, then the robber can evade the cops indefinitely simply by starting on v and never leaving that vertex. The cops cannot capture the robber if he uses this strategy since the vertices in D have no in-edges.

If the cops do start on the vertices of D, the robber must start at r' because every other vertex in the graph is adjacent to a starting position for a cop.

At this point, the cops must all move into C on the same turn or else the robber can escape. If the *chaser cop* (the one starting at s') moves to s while some other cop stays in D and thus leaves vertex $v \in C - \{s\}$ unoccupied, then R_1^v will be unguarded and the robber can escape into it. On the other hand, if the cop on $v \in D - \{s'\}$ (for arbitrary v) moves into C before the chaser cop moves to s, then the hypercube R_2^v will be left unguarded. To prevent the robber from escaping via one of these two

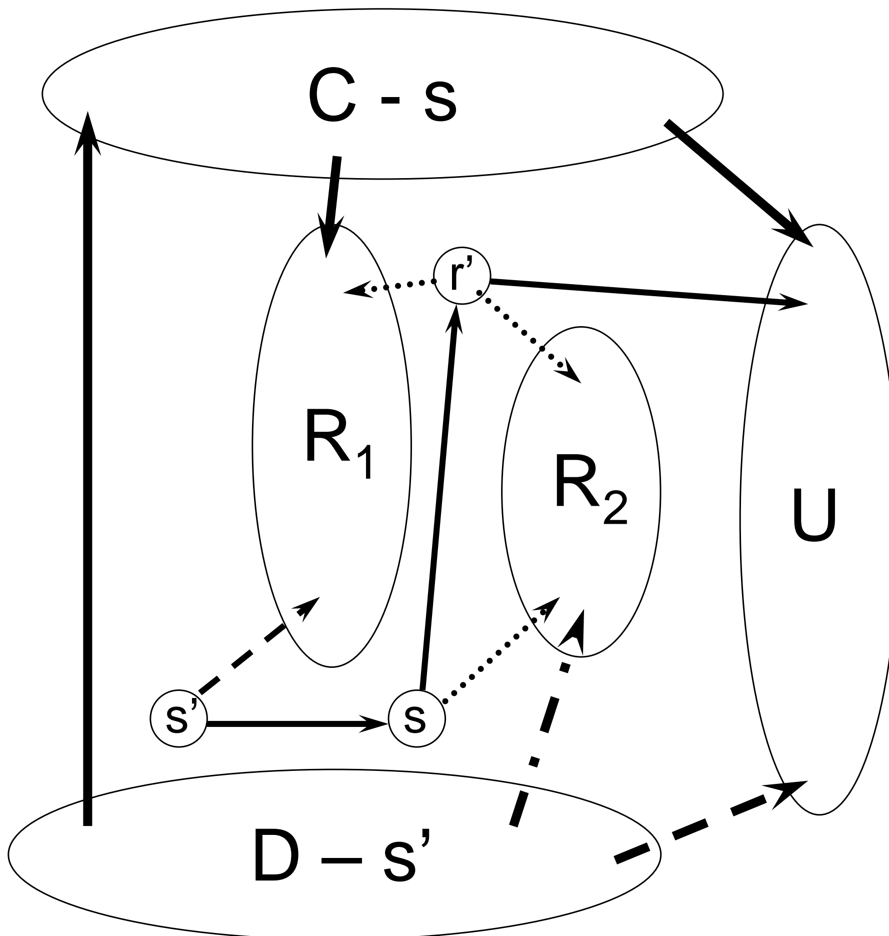

Figure 7.5: Connections in G'. Solid arrows indicate a directed one-to-one mapping from the vertices in the origin set to the vertices in the destination set. Dashed arrows indicate that every vertex in the origin set has an edge to every vertex in the destination set. Dotted arrows indicate that each vertex in the origin set has one edge to each escape subgraph in the destination set. Arrows that alternate between dashes and dots indicate that each vertex in the origin set has an edge to every vertex of its unique escape subgraph in the destination set.

situations, the cops must either take their positions in C all at once, or stay put. In the latter case, the robber can simply remain in his current position. In the former, the robber's only escape is to r.

Once the cops are on the vertices of C and the robber is on r, we have reached the starting position of the Seeded Pursuit-Evasion instance. The remainder of the game plays out as it did before (since the vertices in U have not gained any new out-edges). Thus we have a winning strategy in this instance if and only if we had one

in the SEEDED PURSUIT-EVASION instance. This gives us a parameterized reduction from UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$ to DIRECTED PURSUIT-EVASION since $k = |C|$ and $|C| = r + 1$, showing that DIRECTED PURSUIT-EVASION is AW[*]-hard. □

7.5.1 Short Directed Pursuit-Evasion

The last problem we consider is the short variant of DIRECTED PURSUIT-EVASION.

SHORT DIRECTED PURSUIT-EVASION
Input: directed graph $G = (V, E)$, positive integers k and t.
Rules:

- At the beginning of the game, each of the k cops chooses a vertex of G to be deployed to.
- Once the cops are deployed, the robber chooses a vertex in V to start on.
- For the remainder of the game the cops and robber alternate taking turns, starting with the cops.
- During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put. Multiple cops may occupy the same node at a time.
- On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.
- If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.
- Both players have complete information. That is, the cops know the positions of the robber and their fellow cops at all times, and vice-versa.
- You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Parameters: k, t
Question: Can the cops guarantee capture of the robber within t rounds?

We derive AW[*]-completeness for this problem using Theorem 15 and the techniques from Section 7.4.

Theorem 16. Short Directed Pursuit-Evasion *is AW[*]-complete.*

Proof. Lemma 47 shows that if an instance of Seeded Pursuit-Evasion admits a winning strategy for the cops, then the cops require at most $r+1$ rounds to win. Our reduction to Directed Pursuit-Evasion starts from Seeded Pursuit-Evasion. The gadgets added by the reduction force the cops to spend exactly two additional rounds getting into the starting positions for the Seeded Pursuit-Evasion instance (one round to move into the new starting vertices D, and one round to move from the new starting positions into the old positions C), after which the game is played out as before. Therefore, if the cops have a winning strategy in the Short Directed Pursuit-Evasion instance they require at most $r+3$ rounds to win. This gives us a parameterized reduction, making Short Directed Pursuit-Evasion AW[*]-hard.

To show membership in AW[*], we reuse the reduction from Theorem 13 in Section 7.4 from Parameterized QBFSAT$_t$ to Short Pursuit-Evasion by omitting the rules enforcing starting positions (R_1 and R_2). For R_4 and R_5 (the movement rules) note that these have already been written in a manner suitable for directed graphs. The remaining rules do not concern edges or starting positions, and thus remain unchanged. Thus, Short Directed Pursuit-Evasion is in AW[*]. □

7.6 Summary

Seeded Pursuit-Evasion	-	AW[*]-hard
Short Seeded Pursuit-Evasion	PSPACE-complete	AW[*]-complete
Directed Pursuit-Evasion	-	AW[*]-hard
Short Directed Pursuit-Evasion	PSPACE-complete	AW[*]-complete

Table 7.1: New Complexity Results for Pursuit Games

New complexity results presented in this chapter are summarized in Table 7.1. Goldstein and Reingold previously showed that Seeded Pursuit-Evasion and Directed Pursuit-Evasion were hard for EXP [34].

Open Problems

We can argue for SEEDED PURSUIT-EVASION and DIRECTED PURSUIT-EVASION that the number of positions in a game of either is $O(|V|^k)$. Standard game-theoretic algorithms can solve a game in time which is polynomial in the number of nodes in the game, so with such algorithms we can show these problems to be in XP. However, this leaves us with a gap between hardness and membership. What is the exact classification of these parameterized problems?

We know that SHORT PURSUIT-EVASION is in AW[*]. Can we adapt the techniques we used in this paper to find an AW[*]-hardness reduction for SHORT PURSUIT-EVASION as well? Or is the problem easier? Similarly, what about PURSUIT-EVASION? We know that the problem is W[2]-hard [26], but given that so many variants of the problem are hard for AW[*] it seems quite plausible that it is not in W[2] and thus we are interested in its exact characterization.

Chapter 8

Conclusions

We started this thesis by asking how difficult it is to play games, later refining this into the question of how difficult it is to determine whether there exists a winning strategy for a given combinatorial game. In Chapter 4 we found that, in the context of classical complexity theory, the problem of determining whether a given combinatorial game admits a winning strategy is complete for PSPACE or EXP. When moved on to parameterized complexity and considered short game problems in Chapter 5, we found evidence that short winning strategy problems are typically in AW[*]. In Chapter 6 we added more evidence by showing that Short Generalize Chess (A.26) is AW[*]-complete, and finally, in Chapter 7 we showed that two variant short pursuit game problems were also AW[*]-complete, and that when we considered using the number of cops as the only parameter in these games (removing the k-move constraint) these problems remained AW[*]-hard.

8.1 Contributions

The bulk of the contributions in this thesis come in the form of hardness and membership results. In addition to these technical results, which we list below, Chapter 4 of this thesis functions as a survey on the complexity of games in general. Although the topic is briefly reviewed in [73], and [39] covers it in detail, in Chapter 4 we offer a substantially different perspective on the topic.

Table 8.1 lists the new complexity results established in this thesis for specific computational problems. In addition to these results and the survey contributions mentioned earlier, we have two additional contributions: Firstly, we performed a

Problem	New Result	Section
Succinct Winning Strategy (A.36) with polynomial termination	PSPACE-complete	Section 4.3
Succinct Winning Strategy with an oracle for player two	PSPACE-complete	Section 4.4
Succinct Winning Strategy with polynomial termination and oracle	NP-complete	Section 4.4
Short Succinct Winning Strategy (A.32) with polynomial bound on available moves	in XP	Section 5.1
Short Succinct Winning Strategy	AW[P]-hard	Section 5.1
Short Alternating Hitting Set (A.23)	in AW[*]	Section 5.3
Short Generalized Hex (A.28)	in AW[*]	Section 5.4
Endgame Generalized Othello (A.10)	FPT	Section 5.7
Short Generalized Othello (A.29)	in AW[*]	Section 5.7
Endgame Generalized Checkers (A.9)	FPT	Section 5.6
Short Generalized Chess (A.26)	AW[*]-complete	Chapter 6
Seeded Pursuit-Evasion (A.22)	AW[*]-hard	Chapter 7
Directed Pursuit-Evasion (A.6)	AW[*]-hard	Chapter 7
Short Seeded Pursuit-Evasion (A.31)	AW[*]-complete	Chapter 7
Short Directed Pursuit-Evasion (A.30)	AW[*]-complete	Chapter 7

Table 8.1: New results presented for specific problems in this thesis.

complete analysis of the game brainstones as an illustration of standard minimax brute-force techniques. This is explained in Section 3.3 and pesudocode is given in Appendix C. Secondly, in Section 5.2 we provided a template argument (Lemma 9) for showing that classical hardness results for combinatorial game problems also apply to their short versions. Since short games all have polynomial termination by definition, it should be easy to show that they are in PSPACE with techniques such as we used in Lemma 4.

8.2 Open Problems

In the course of our investigations we have come across many new problems. We shall list them first before discussing each in detail.

1	How do we generalize brainstones, and what is the complexity of the resulting problem?
2	Is SHORT ALTERNATING HITTING SET (A.23) AW[*]-complete? (Known to be in AW[*].)
3	Is SHORT GENERALIZED OTHELLO (A.29) AW[*]-complete? (Known to be in AW[*].)
4	Is SHORT GENERALIZED HEX (A.28) AW[*]-complete? (Known to be in AW[*].)
5	What is the parameterized classification of SHORT GENERALIZED CHECKERS (A.25)?
6	What is the parameterized classification of SHORT GENERALIZED GO (A.27)?
7	Is SEEDED PURSUIT-EVASION (A.22) in AW[*]? (Known to be AW[*]-hard.)
8	Is DIRECTED PURSUIT-EVASION (A.6) in AW[*]? (Known to be AW[*]-hard.)
9	What is the exact classification of SHORT SUCCINCT WINNING STRATEGY (A.32) with the restriction that at most a polynomial number of moves are available from each position? (Known to be in XP, and AW[P]-hard without the polynomial restriction.)

Table 8.2: Open Problems

8.2.1 Brainstones

As we mentioned in Section 4.6, we are curious about the complexity of the winning strategy problem for brainstones, but we unsure exactly how to generalize the layout of the symbols on the board. Including the symbols on the board as part of the input allows us to circumvent the problem, and this may be a good starting point. It seems somewhat unlikely that the complexity of the problem is different depending on the board layout, but if this is indeed the case it may be quite interesting to quantify exactly how the complexity of the game depends on the board layout.

8.2.2 Short Games

Chapter 5 includes a couple of AW[*]-membership results for SHORT ALTERNATING HITTING SET (A.23), SHORT GENERALIZED HEX (A.28) , and SHORT GENERALIZED OTHELLO (A.29). Unfortunately, we lack matching hardness results.

Besides these games, we are interested in two additional short games for which we have no results, and some pursuit-evasion variants:

Checkers

Also, while we presented a result for the ENDGAME GENERALIZED CHECKERS (A.9) problem, we have not been able to categorize the short winning strategy problem for checkers without including the number of pieces as a parameter – SHORT GENERALIZED CHECKERS (A.25). Unfortunately, encoding the moves proves to be quite difficult. There are too many pieces on the board to simply record their positions in a “natural” encoding. However, if we try a scheme similar to the ones we used for othello and chess we run into problems when a player jumps several of his opponent’s pieces in a single turn; simply recording which piece moved and where it landed may not be enough to uniquely identify what path the piece followed and hence which of his opponent’s pieces should be removed (see Figure 8.1 for an example of a simple position where this is the case).

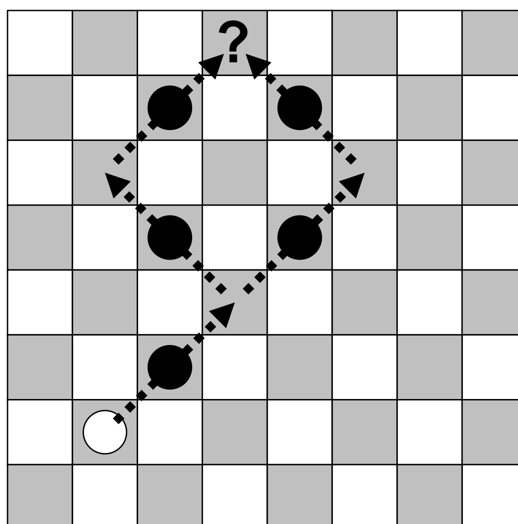

Figure 8.1: It is unclear which of the available paths the white checker took to reach the question mark.

Go

We conjecture that the short winning strategy problem for go is in AW[P]. The primary obstacle we encountered when looking for an AW[*]-hardness proof was the issue of solving implicit reachability problems. However, if we are willing to content ourselves with an AW[P] result then it is simple to construct a circuit to solve these

reachability problems, which in turn would enable us to construct circuits to determine a square's content on the basis of its history in a fashion similar to what we did for othello and chess. With this in place it our encoding scheme could be as simple as recording which square the new piece was placed in each turn. Assuming all this can be done, the one issue remaining to be solved is scoring. See the appendix (B.9) for some of the issues involved in this.

Pursuit-Evasion

Our investigation into parameterized pursuit-evasion raised several open questions. Firstly, though we got AW[*]-hardness results for Seeded Pursuit-Evasion (A.22) and Directed Pursuit-Evasion (A.6), the only membership results we can provide are for XP. We are curious as to the exact classification of these problems.

We are also interested in the complexity of Short Pursuit-Evasion (A.22), which is Short Seeded Pursuit-Evasion (A.31) without the starting position constraints. We know that this problem is in AW[*] because we can simply remove the starting position constraints from the membership reduction for Short Seeded Pursuit-Evasion (see Section 7.4.2). However, a hardness reduction currently eludes us.

8.2.3 Is AW[*] the Natural Home of Short Games?

Our interest in this problem arises from Downey and Fellows' conjecture that AW[*] is the "natural" home of short games [21]. In Section 5.1 we considered a generic parameterized hardness reduction for short games analogous to the reductions in Chapter 4. We remain interested in this line of investigation.

Short Succinct Winning Strategy

If we continue to represent game rules with succinct circuits as we did in Chapter 4, then we can work with the problem Short Succinct Winning Strategy (A.32). The advantage of this is that we have an AW[P]-hardness reduction for the problem (Lemma 8) and we need only established membership. Unfortunately, we have encountered some obstacles to producing a membership reduction for Short Succinct Winning Strategy. The primary obstacle such a reduction must overcome is that Parameterized QCSAT (A.18) applies weight control to its bitstrings, while there

is no overt notion of weight control in Short Succinct Winning Strategy. We see two possibilities by which this obstacle might be overcome.

The first possibility is to add the number of pieces as a parameter. The advantage of this approach is that you can continue to use the sort of natural encoding scheme which we used to encode positions in Chapters 5, 6, and 7. The obvious disadvantage to this approach is that you've added a parameter and would therefore also need redo the hardness reduction, although a proof for this parameterization would be interesting in its own right since you could say something about "endgame" problems in general; our endgame checkers problem is FPT, but can we say that about endgame problems in general? It seems plausible that short chess is be AW[*]-hard even if the number of pieces were to be included as a parameter. Perhaps more importantly, employing this approach makes some explicit assumptions about how the game encodes positions, and as such may actually be a disadvantage as well.

The second possibility is to add a polynomial bound to the number of moves from any given position, as we did in Lemma 7 (Section 5.1, page 55). This is a relatively safe assumption, since nearly every game in this thesis admits such a bound, and we have already shown that this problem is XP. Moreover, you can reverse-engineer a weight bound from the polynomial bound; if there are at most $c \cdot n^k$ moves from any given position, then you can represent every possibility with $c{\cdot}n{\cdot}k$ bits and a weight of k. In other words, you know your bitstrings have a sufficient number of configurations to represent every possible move each turn. It should not be too difficult to set up machinery which provides membership results given an artificial move representation scheme and a logical formula to decode a "natural" position representation from an initial position and sequence of moves. A more challenging question may be whether you can find such schemes for all games where the number of moves is bounded by a polynomial function of the board size.

Other Succinct Representations

Even if we are able to prove that Short Succinct Winning Strategy (A.32) or some restriction thereof is complete for AW[P], this is not necessarily the final word on short games. Given the number of AW[*] membership results we have, it is clear that many games do not require the computational power afforded by AW[P]. We were able to show this even for Short Generalized Hex (A.28) – where the winning condition implicitly requires more computational power than a first-order

formula can provide – by offloading work onto the quantifiers in a parameterized manner (Theorem 6). It seems quite plausible that a technique for showing that the short winning strategy problem for games given as circuits is AW[P]-complete could be adapted to show that the same problem for games represented by first-order logic formulas is AW[SAT]-complete, and that games represented with t-normalized first-order logic formulas are AW[*]-complete. Thus, the question is which of these three possible succinct representations is sufficient to encode combinatorial games.

Bibliography

[1] FIDE handbook (online version): Chess rules. http://www.fide.com/official/handbook.asp?level=EE101.

[2] World Othello Federation: Othello rules. http://www.worldothellofederation.com/rules/rulesenglish.htm.

[3] K. A. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-parameter tractability and completeness IV: on completeness for W[P] and PSPACE analogues. *Annals Of Pure And Applied Logic*, 73:235–276, 1995.

[4] K. R. Abrahamson, R. G. Downey, and M. R. Fellows. Fixed-Parameter Intractability II (Extended Abstract). In *STACS '93: Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, pages 374–385, London, UK, 1993. Springer-Verlag.

[5] A. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Math*, 8:1–11, 1984.

[6] M. Albert, R. Nowakowski, and D. Wolfe. *Lessons in Play: An Introduction to Combinatorial Game Theory.* A. K. Peters, Ltd., 2007.

[7] V. Allis. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, University of Limburg, Maastricht, The Netherlands, 1994.

[8] B. Alspach. Searching and sweeping graphs: A brief survey. *Le Matematiche*, 59:5–37, 2004.

[9] J. L. Balcázar. The complexity of searching implicit graphs. *Artif. Intell.*, 86(1):171–188, 1996.

[10] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*, volume 2. Academic Press, 1982.

[11] M. Champion. Re: How many atoms make up the universe? http://www.madsci.org/posts/archives/oct98/905633072.As.r.html, 1998 (retrieved 2009).

[12] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.

[13] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In R. Kralovic and P. Urzyczyn, editors, *MFCS*, volume 4162 of *Lecture Notes in Computer Science*, pages 238–249. Springer, 2006.

[14] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hille, editor, *Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science*, pages 24–30. Elsevier/North-Holland, 1964.

[15] J. H. Conway. *On Numbers and Games.* AK Peters, Ltd., December 2000.

[16] S. A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[17] E. D. Demaine and R. A. Hearn. Constraint logic: A uniform framework for modeling computation as games. In *IEEE Conference on Computational Complexity*, pages 149–162. IEEE Computer Society, 2008.

[18] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness. *Congressus Numerantium*, 87:161–187, 1992.

[19] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal of Comput.*, 24:873–921, 1995.

[20] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141:109–131, 1995.

[21] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[22] J. Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards*, 69:67–72, 1965.

[23] S. Even and R. E. Tarjan. A combinatorial problem which is complete in polynomial space. *J. ACM*, 23(4):710–719, 1976.

[24] J. Feinstein. Amenor wins world 6×6 championships! *British Othello Federation Newsletter*, page 69, July 1993. http://www.maths.nott.ac.uk/othello/Jul93/Amenor.html.

[25] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[26] F. Fomin, P. Golovach, and J. Kratochvíl. On tractability of cops and robbers game. In G. Ausiello, J. Karhumäki, G. Mauri, and C.-H. L. Ong, editors, *IFIP TCS*, volume 273 of *IFIP*, pages 171–185. Springer, 2008.

[27] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

[28] A. S. Fraenkel. Combinatorial games: Selected bibliography with a succinct gourmet introduction. http://www.combinatorics.org/Surveys/index.html.

[29] A. S. Fraenkel, M. R. Garey, D. S. Johnson, T. J. Shaefer, and Y. Yesha. The complexity of checkers on an $N \times N$ board - preliminary report. In *Proc. 19th Ann. Symp. of Foundations of Computer Science, Long Beach, CA*, pages 55–64. IEEE Computer Society, 1978.

[30] A. S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. In *Proceedings of the 8th Colloquium on Automata, Languages and Programming*, pages 278–293, London, UK, 1981. Springer-Verlag.

[31] H. Galperin and A. Wigderson. Succinct representations of graphs. *Inf. Control*, 56(3):183–198, 1983.

[32] M. Gardner. Mathematical games. *Scientific American*, pages 145–150, 1957.

[33] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[34] A. S. Goldstein and E. M. Reingold. The complexity of pursuit on a graph. *Theoretical Computer Science*, 143:93–112, 1995.

[35] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press US, 1995.

[36] G. Hahn. Cops, robbers and graphs. *Tatra Mountains Mathematical Publications*, 36:1–14, 2007.

[37] G. Hahn and G. MacGillivray. A note on k-cop, ℓ-robber games on graphs. *Discrete Mathematics*, 306(19-20):2492–2497, 2006.

[38] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[39] R. A. Hearn. *Games, Puzzles, and Computation*. PhD thesis, Massachusetts Institute of Technology, 2006.

[40] N. Immerman. *Descriptive Complexity*. Springer-Verlag, 1999.

[41] S. Iwata and T. Kasai. The othello game on an $n \times n$ board is PSPACE-complete. *Theoretical Computer Science*, 123:329–340, 1994.

[42] N. D. Jones and W. T. Laaser. Complete problems for deterministic polynomial time. In *STOC '74: Proceedings of the sixth annual ACM Symposium on Theory of Computing*, pages 40–46, New York, NY, USA, 1974. ACM.

[43] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[44] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.

[45] D. Lichtenstein and M. Sipser. Go is Pspace hard. *SIAM Journal of Comput.*, pages 48–54, 1978.

[46] A. Lozano and J. L. Balczar. *The complexity of graph problems for succinctly represented graphs*, pages 277–286. Springer, 1990.

[47] P. Morris. *Introduction to Game Theory*. Springer, 1994.

[48] R. Neidermeier. *Invitation to Fixed Parameter Algorithms*. Oxford University Press, 2006.

[49] J. V. Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[50] R. J. Nowakowski, editor. *Games of No Chance*. Cambridge University Press, 1996.

[51] R. J. Nowakowski, editor. *More Games of No Chance*. Cambridge University Press, 2002.

[52] R. J. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Math*, 43:235–239, 1983.

[53] M. J. Osborne and P. S. Walker. A note on 'The Early History of the Theory of Strategic Games from Waldegrave to Borel' by Robert W. Dimand and Mary Ann Dimand. *History of Political Economy*, 28:8182, 1996.

[54] C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Inf. Control*, 71(3):181–185, 1986.

[55] A. Quilliot. Thèse de 3ème cycle. In *Université de Paris VI*, pages 131–145. 1978.

[56] A. Quilliot. A short note about pursuit games played on a graph with a given genus. *J. Comb. Theory, Ser. B*, 38(1):89–92, 1985.

[57] O. Reingold. Undirected ST-connectivity in log-space. In *STOC '05: Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing*, pages 376–385, New York, NY, USA, 2005. ACM.

[58] S. Reisch. Gobang ist PSPACE-vollstandig. *Acta Informatica*, 13:59–66, 1980.

[59] S. Reisch. Hex ist PSPACE-vollstandig. *Acta Informatica*, 15:167–191, 1981.

[60] D. J. Richards and T. P. Hart. The alpha-beta heuristic. Technical report, Massachusetts Institute of Technology, 1961. http://dspace.mit.edu/handle/1721.1/6098.

[61] J. M. Robson. The complexity of go. *Information Processing; Proceedings of the IFIP Congress*, pages 413–417, 1983.

[62] J. M. Robson. N by N checkers is Exptime complete. *SIAM Journal on Computing*, 2:252–267, 1984.

[63] W. Savitch. Relationship between deterministic and nondeterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[64] T. J. Schaefer. On the completity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16:185–225, 1978.

[65] J. Schaeffer, N. Burch, Y. Bjrnsson, A. Kishimoto, M. Mller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[66] A. Scott. Parameterized complexity of cliques and games. Master's thesis, University of Victoria, 2004.

[67] A. Scott. Short pursuit-evasion. In *Texts in Algorithmics 7: Algorithms and Complexity in Durham 2006*, pages 141–152, 2006.

[68] A. Scott and U. Stege. Parameterized chess. In *Parameterized and Exact Computation*, pages 172–189, 2008.

[69] P. D. Seymour and R. Thomas. Graph searching, and a min-max theorem for tree-width. Technical Report 89-1, The Center for Discrete Mathematics and Theoretical Computer Science, 1989.

[70] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(314), March 1950.

[71] R. E. Stearns, J. Hartmanis, and P. M. L. II. Hierarchies of memory limited computations. In *FOCS*, pages 179–190. IEEE, 1965.

[72] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[73] L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal of Computing*, 8(2):151–174, 1979.

[74] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9, New York, NY, USA, 1973. ACM.

[75] A. Tay. Guide to the use of computer chess endgame tablebases. http://horizonchess.com/FAQ/Winboard/egtb.html, 2006 (retrieved 2009).

[76] J. Tromp and G. Farnebäck. Combinatorics of go. In H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2006.

[77] H. J. van den Herik Jos W.H.M. Uiterwijk and J. van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134:277311, 2002.

Appendix A

Problem Definitions

A.1 Alternating Reachability

ALTERNATING REACHABILITY
Input: A directed graph $G = (V, E)$, a partition $V = X \cup Y$ of the vertices, and designated vertices s, t.
Question: Is $apath(s,t)$ true? $apath$ is defined as follows. Vertices in X are "existential" while those in Y are "universal". Such a graph is called an *alternating* graph or an AND/OR graph. The predicate $apath(s,t)$ holds if and only if

1. $s = t$, or
2. s is existential and there is a $z \in V$ with $(s,z) \in E$ and $apath(z,t)$, or
3. s is universal and for all $z \in V$ with $(s,z) \in E$, $apath(z,t)$ holds.

Note: We sometimes describe an instance of ALTERNATING REACHABILITY as a tuple (V, E, X, Y, s, t).

This problem is P-complete [40]. The bipartite variant, BIPARTITE ALTERNATING REACHABILITY, where the graph is bipartite with X and Y being the partite sets, is also P-complete [35].

A.2 r-Alternating Weighted CNF Satisfiability

r-Alternating Weighted CNF Satisfiability
Instance: A sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $s_1 \cup \ldots \cup s_r$, where F is in conjunctive normal form; integers $k_1, \ldots, k_r$.
Question: Does there exist a size-k_1 subset t_1 of s_1 such that for every size-k_2 subset t_2 of s_2, there exists a size-k_3 subset t_3 of s_3 such that ... (alternating quantifiers) such that, when the variables in $t_1, \ldots, t_r$ are set to true and all other variables are set to false, formula F is true?
Parameters: $k_1, \ldots, k_r$.

This problem is complete for A[r] for all $r \geq 1$ [25].

A.3 r-Alternating Weighted t-Normalized Satisfiability

r-Alternating Weighted t-Normalized Satisfiability (AWSat$_{t,r}$)
Instance: A sequence $S_1, \ldots, S_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $S_1 \cup \ldots \cup S_r$, where F consists of $t+1$ alternating layers of conjunctions and disjunctions with negations applied only to variables; integers $k_1, \ldots, k_r$.
Question: Does there exist a size-k_1 subset s_1 of S_1 such that for every size-k_2 subset s_2 of S_2, there exists a size-k_3 subset s_3 of S_3 such that ... (alternating quantifiers) such that, when the variables in $s_1, \ldots, s_r$ are set to true and all other variables are set to false, formula F is true?
Parameters: $k_1, \ldots, k_r$.

This problem is complete for the A-matrix class A[t, r] [25].

A.4 Circuit Value

Circuit Value
Instance: A encoding of boolean circuit α, inputs $x_1, \ldots, x_n$, and a designated output y.

Question: On inputs $x_1, \ldots, x_n$, is output y of α true?

This problem is complete for P [44].

A.5 Clique

CLIQUE
Instance: A graph $G = (V, E)$, positive integer k.
Question: Is there a set of vertices $V' \subset V$ such that the subgraph induced by V' is a complete graph and $|V'| \leq k$?

This problem is NP-complete [43] and W[1]-complete for parameter k [20].

A.6 Directed Pursuit-Evasion

DIRECTED PURSUIT-EVASION
Input: directed graph $G = (V, E)$, positive integer k.
Rules:

- At the beginning of the game, each of the k cops chooses a vertex of G to be deployed to.
- Once the cops are deployed, the robber chooses a vertex in V to start on.
- For the remainder of the game the cops and robber alternate taking turns, starting with the cops.
- During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put. Multiple cops may occupy the same node at a time.
- On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.
- If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.

- Both players have complete information. That is, the cops know the positions of the robber and their fellow cops at all times, and vice-versa.
- You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Parameter: k
Question: Do the cops have a winning strategy in the game described by the input?

This problem is AW[*]-hard (Theorem 15).

A.7 Directed Reachability

DIRECTED REACHABILITY
Input: Directed graph $G = (V, E)$, distinguished vertices $s, t \in V$. *Question*: Is there a path from s to t in G?

This problem is NL-complete [40].

A.8 Dominating Set

DOMINATING SET
Instance: A graph $G = (V, E)$, positive integer k. *Question*: Is there a set of vertices $V' \subseteq V$ such that every vertex $v \in V$ is either in V' or is adjacent to a vertex in V', and $|V'| \leq k$?

This problem is NP-complete [33] and W[2]-complete for parameter k [21].

A.9 Endgame Generalized Checkers

ENDGAME GENERALIZED CHECKERS
Input: A checkers position with k pieces on the board, positive integer t.

Question: Does first player have a t-turn winning strategy from the given position in the following game?
Game Rules: As per checkers.
Parameters: k, t

This problem is in FPT (Lemma 14).

A.10 Endgame Generalized Othello

ENDGAME GENERALIZED OTHELLO
Input: An $n \times n$ othello board position with exactly k unoccupied squares remaining.
Question: Starting from the given position, does the next player to move have a strategy which guarantees a win within the next k moves?
Parameter: k

This problem is in FPT (Lemma 15).

A.11 Game

GAME
Input: A game $\mathbb{G} = (P_1, P_2, W_0, s, M)$, where P_1 and P_2 are disjoint sets (of positions for player 1 and 2 respectively), $W_0 \subseteq P_1 \cup P_2$ (the winning positions), $s \in P_1$ (the starting position), and $M \subseteq P_1 \times P_2 \cup P_2 \times P_1$ (the set of allowable moves).
Question: Does player one have a winning strategy in $\mathbb{G}$?

This problem is P-complete [42].

A.12 Generalized Geography

GENERALIZED GEOGRAPHY
Input: A directed graph $G = (V, A)$ and a starting position $s \in V$.

Question: Does player one have a winning strategy in the following game? Players alternate choosing a new arc from A. The first arc chosen must have its tail at s and each subsequently chosen arc must have its tail at the vertex that was the head of the previously chosen arc. The first player unable to choose such an arc loses.

This problem is PSPACE-complete even when restricted to planar bipartite graphs with no in- or out-degree that exceeds 2 and a maximum total degree of 3 [45].

A.13 Generalized Othello

GENERALIZED OTHELLO
Input: An $n \times n$ othello board position.
Question: Starting from the given othello position, does the next player to move have a strategy which guarantees a win?

This problem is PSPACE-complete [41].

A.14 Hitting Set

HITTING SET
Instance: A collection C of elements, a set S of subsets of C, a positive integer k. *Question*: Is there a set S' such that $|S'| \leq k$ and for every $T \in S$, $T \cap S' \neq \emptyset$?

This problem is NP-complete [43] and W[2]-complete for parameter k [21].

A.15 Independent Set

INDEPENDENT SET
Instance: A graph $G = (V, E)$, positive integer k. *Question*: Is there a set of vertices $V' \subseteq V$ such that no two vertices in V' are adjacent to the same edge and $|V'| \geq k$?

This problem is NP-complete [33] and W[1]-complete for parameter k [20].

A.16 Parameterized Quantified Boolean Formula Satisfiability

PARAMETERIZED QUANTIFIED BOOLEAN FORMULA SATISFIABILITY (PARAMETERIZED QBFSAT)
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F involving the variables $s_1 \cup \ldots s_r$; integers $k_1, \ldots, k_r$.
Question: Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2, there exists a size k_3 subset t_3 of s_3 such that... (alternating qualifiers) such that, when the variables in $t_1 \cup \ldots t_r$ are made true and all other variables are made false, formula F is true?
Parameter: $r, k_1, \ldots, k_r$

This problem is complete for AW[SAT] [21].

A.17 Parameterized Quantified Boolean t-Normalized Formula Satisfiability

PARAMETERIZED QUANTIFIED BOOLEAN t-NORMALIZED FORMULA SATISFIABILITY (PARAMETERIZED QBFSAT$_t$)
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $s_1 \cup \ldots \cup s_r$, where F consists of $t+1$ alternating layers of conjunctions and disjunctions with negations applied only to variables (t is a fixed constant); integers $k_1, \ldots, k_r$.
Question: Is it the case that there exists a size-k_1 subset t_1 of s_1 such that for every size-k_2 subset t_2 of s_2, there exists a size-k_3 subset t_3 of s_3 such that ... (alternating quantifiers) such that, when the variables in $t_1, \ldots, t_r$ are set to true and all other variables are set to false, formula F is true?
Parameter: r, $k_1, \ldots, k_r$.

This problem is complete for AW[*] [3].

A.18 Parameterized Quantified Circuit Satisfiability

PARAMETERIZED QUANTIFIED CIRCUIT SATISFIABILITY
(PARAMETERIZED QCSAT)
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a decision circuit C with the variables $s_1 \cup \ldots s_r$ as inputs; integers $k_1, \ldots, k_r$.
Question: Is it the case that there exists a size k_1 subset t_1 of s_1 such that for every size k_2 subset t_2 of s_2, there exists a size k_3 subset t_3 of s_3 such that... (alternating qualifiers) such that, when the inputs in $t1 \cup \ldots tr$ are set to 1 and all other inputs are set to 0, circuit C outputs 1?
Parameter: $r, k_1, \ldots, k_r$

This problem is complete for AW[P] [21].

A.19 Quantified Boolean Formula

QUANTIFIED BOOLEAN FORMULA
Input: Set $U = u_1, u_2, \ldots, u_n$ of boolean variables, well-formed quantified Boolean formula $F = (Q_1 u_1)(Q_2 u_2) \cdots (Q_n u_n) E$, where E is a Boolean expression and each Q_i is either $\forall$ or $\exists$.
Question: Is F true?

This problem is PSPACE-complete [74].

A.20 Restricted Alternating Hitting Set

RESTRICTED ALTERNATING HITTING SET
Instance: A collection C of subsets of a set B with $|c| \leq k_1$ for all $c \in C$; an integer k_2.
Question: Does player one have a forced win in no more than k_2 moves in the following game played on C and B? Players alternate choosing a new element of B until, for each $c \in C$, some member of c has been chosen.

The player whose choice causes this to happen loses.
Parameter: k_1, k_2

This problem is in FPT [3].

A.21 Satisfiability

SATISFIABILITY
Input: A set of boolean variables U, a set of clauses C over the variables of U.
Question: Does there exist a setting for the variables of U which satisfies C?

This problem is NP-complete [16].

A.22 Seeded Pursuit-Evasion

SEEDED PURSUIT-EVASION
Input:

- A simple, undirected graph $G = (V, E)$
- A set $C \subseteq V$ of starting positions for the cops
- Starting position $a \in V, a \notin C$, for the robber

Rules:

1. C defines the starting positions of the cops. Exactly one cop starts at each $v \in C$.
2. The robber begins at vertex a.
3. The cops and robber alternate taking turns, starting with the cops.

4. During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put.[1] Multiple cops may occupy the same node at a time.

5. On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.

6. If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.

7. Both players have complete information. That is, the cops know the positions of the robber and their fellow cops at all times, and vice-versa.

8. You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Parameter: $|C|$
Question: Can the cops guarantee capture of the robber?

This problem is AW[*]-hard (Theorem 11).

A.23 Short Alternating Hitting Set

SHORT ALTERNATING HITTING SET (SAHS)
Input: A set of elements E, a set S of subsets over E, integer k.
Question: Does player one have a strategy to force a win within the next k moves in the following game?
Game Rules: Players one and two alternate choosing (unchosen) elements of E, which they add S'. S' starts the game empty. The player who chooses the last element such that S' becomes a hitting set for S wins.
Parameter: k

This problem is in AW[*] (Theorem 5).

[1]Note that we say the cops and robber *may* move, but they can also stay at the vertex they currently occupy. This is equivalent to the variant where all pieces *must* move and every vertex has an edge to itself, since in that variant a piece can stay at its current vertex by moving along the reflexive edge.

A.24 Short Directed Pursuit-Evasion

SHORT DIRECTED PURSUIT-EVASION
Input: directed graph $G = (V, E)$, positive integers k and t.
Rules:

- At the beginning of the game, each of the k cops chooses a vertex of G to be deployed to.
- Once the cops are deployed, the robber chooses a vertex in V to start on.
- For the remainder of the game the cops and robber alternate taking turns, starting with the cops.
- During the cops' turn, each cop either moves to a vertex adjacent to his current position, or stays put. Multiple cops may occupy the same node at a time.
- On his turn, the robber either moves to an unoccupied vertex adjacent to his current position, or stays put.
- If, at any time, the robber occupies the same vertex as a cop, the robber is captured and the cops win.
- Both players have complete information. That is, the cops know the positions of the robber and their fellow cops at all times, and vice-versa.
- You cannot add or subtract pieces (cops or robbers) from the game at any time. Further, any given piece occupies exactly one vertex at a time.

Parameters: k, t
Question: Can the cops guarantee capture of the robber within t rounds?

This problem is AW[*]-complete (Theorem 16).

A.25 Short Generalized Checkers

Short Generalized Checkers
Input: An $(n \times n)$-chessboard position[2], a positive integer k.
Parameter: k
Question: Starting from the given position, does player **I** have a strategy to force a win within the next k moves?

Generalized Geography is PSPACE-complete, and thus by Lemma 9 so is this problem. The parameterized complexity of this problem remains unknown. Endgame Generalized Checkers (A.9) – the endgame version of this problem – is in FPT.

A.26 Short Generalized Chess

Short Generalized Chess
Input: An $(n \times n)$-chessboard position[3], a positive integer k.
Parameter: k
Question: Starting from the given position, does player **I** have a strategy to force a win within the next k moves?

This problem is AW[*]-complete (Chapter 6, also [68]).

A.27 Short Generalized Geography

Short Generalized Geography
Input: A directed graph $G = (V, E)$, a starting position $s \in V$, and an integer t.
Question: Does player one have a winning strategy that takes at most t

[2] A *chessboard position* includes the position of every piece (including captured pieces), the turn number, and a flag for each king and rook indicating whether that piece has moved yet (essential information to castling, Rule 16).

[3] A *chessboard position* includes the position of every piece (including captured pieces), the turn number, and a flag for each king and rook indicating whether that piece has moved yet (essential information to castling, Rule 16).

turns to execute? Players alternate choosing a new arc from E. The first arc chosen must have its tail at s and each subsequently chosen arc must have its tail at the vertex that was the head of the previously chosen arc. The first player unable to choose such an arc loses.

This problem is in FPT (Corollary 2).

A.28 Short Generalized Hex

SHORT GENERALIZED HEX
Input: A graph $G = (V, E)$, vertices $\alpha, \omega \in V$, integer k.
Question: Does player one have a k-turn winning strategy in the following game?
Game Rules: Two players alternating place marked pebbles on unpebbled vertices. Player one wins if he can create a path from α to ω with his pebbles. Player two wins if player one is unable to form such a path.
Parameter: k

This problem in in AW[*] (Theorem 6).

A.29 Short Generalized Othello

SHORT GENERALIZED OTHELLO (SGO)
Input: An $n \times n$ othello board position, integer k.
Question: Starting from the given position, does the next player to move have a strategy to force a win within the next k moves?
Parameter: k

This problem is in AW[*] (Theorem 7).

A.30 Short Directed Pursuit-Evasion

SHORT DIRECTED PURSUIT-EVASION
Input:

- A simple, undirected graph $G = (V, E)$

- Starting position $a \in V$ for the robber.
- positive integer t.

Rules: *As for* Directed Pursuit-Evasion.
Parameters: t, $|C|$
Question: Can the cops guarantee capture of the robber within t moves?

This problem is AW[*]-complete (Theorem 16).

A.31 Short Seeded Pursuit-Evasion

Short Seeded Pursuit-Evasion
Input:

- A simple, undirected graph $G = (V, E)$
- A set $C \subseteq V$ of starting positions for the cops.
- Starting position $a \in V$ for the robber.
- positive integer t.

Rules: *As for* Seeded Pursuit-Evasion.
Parameters: t, $|C|$
Question: Can the cops guarantee capture of the robber within t moves?

This problem is AW[*]-complete (Theorem 14).

A.32 Short Succinct Winning Strategy

Short Succinct Winning Strategy
Input: a positive integer t, a circuit M which takes in two positions as bit strings p_1, p_2 and outputs true if moving from p_1 to p_2 is legal, a starting position p represented as a bitstring, circuits W, L, and D which take as input a bitstring representing some position q and return true if the player to move from q has won, lost, or drawn the game respectively.

Question: starting from position p, does player one have a winning strategy that takes at most t turns to execute in the two-player game described by circuits M, W, L, and D?
Parameter: t.

This is the k-move version of SUCCINCT WINNING STRATEGY, problem A.A.36. This problem is hard for AW[P] (Lemma 8), and in XP if the number of moves available from any given position is polynomial (Lemma 7).

A.33 Succinct Alternating Reachability

SUCCINCT ALTERNATING REACHABILITY
Input: A circuit E describing a bipartite alternating graph which takes in two positions as bit strings p_1, p_2 and outputs true if there is an edge from p_1 to p_2 in the graph, a starting position s represented as a bitstring, a destination vertex t.
Question: Is it possible to reach t from s in the graph described by E?

Note: We sometimes describe an instance of SUCCINCT ALTERNATING REACHABILITY as a tuple (E, s, t).

This problem is EXP-complete [9].

A.34 Succinct Circuit Value

SUCCINCT CIRCUIT VALUE
Instance: A circuit W which takes two inputs a and b and returns whether there is a wire from a to b, and a circuit S which takes an input a and returns whether a is an AND gate, an OR gate, a NOT gate, a true input, a false input, or the output.
Question: Does the output of circuit described by circuits W and S evaluate to true?

This problem is EXP-complete [54].

A.35 Succinct Directed Reachability

Succinct Directed Reachability
Input: A circuit E describing a directed graph which takes in two positions as bit strings p_1, p_2 and outputs true if there is an edge from p_1 to p_2 in the graph, a starting position s represented as a bitstring, a destination vertex t.
Question: Is it possible to reach t from s in the graph described by E?

This problem is PSPACE-complete [46].

A.36 Succinct Winning Strategy

Succinct Winning Strategy
Input: A circuit M which takes in two positions as bit strings p_1, p_2 and outputs true if moving from p_1 to p_2 is legal, a bitstring p representing a starting position, circuits W, L, and D which take as input a bitstring representing some position q and return true if the player to move from q has won, lost, or drawn the game respectively.
Question: Starting from position p, does player one have a winning strategy in the game described by circuits M, W, L, and D?

Note: We sometimes describe an instance of Succinct Winning Strategy as a tuple (M, p, W, L, D).

This problem is EXP-complete by reduction to and from Succinct Alternating Reachability (Lemma 3). It is PSPACE-complete if the game has polynomial termination (Lemma 4) or if player one is given an oracle for player two (Lemma 5). If the game has polynomial termination and player one is given an oracle for player two, the game is NP-complete (Lemma 6). This is the succinct version of Winning Strategy (A.44). The k-move version of this problem is Short Succinct Winning Strategy (A.32).

A.37 Traveling Salesman Problem

Traveling Salesman Problem
Input: A set C of m cities, a distance function $d(c_i, c_j) \in \mathbb{Z}^+$ for each pair

of cities c_i, $c_j \in C$, a positive integer B. *Question*: Is there a tour of C having length B or less, that is, a permutation γ of C such that:

$$\left(\sum_{1 \leq i < m} d\left(c_{\gamma(i)}, c_{\gamma(i+1)}\right) \right) + d\left(c_{\gamma(m)}, c_{\gamma(1)}\right) \leq B$$

This problem is NP-complete [33].

A.38 Undirected Reachability

UNDIRECTED REACHABILITY
Input: Directed graph $G = (V, E)$, distinguished vertices $s, t \in V$. *Question*: Is there a path from s to t in G?

This problem is L-complete [57].

A.39 Unitary Monotone Parameterized QBFSAT$_2$

UNITARY MONOTONE PARAMETERIZED QBFSAT$_2$
Instance: An integer r; a sequence $s_1, \ldots, s_r$ of pairwise disjoint sets of boolean variables; a boolean formula F over the variables $s_1 \cup \ldots \cup s_r$ in conjunctive normal form.
Question: Is it the case that there exists a variable t_1 in s_1 such that for every variable t_2 in s_2, there exists a variable t_3 in s_3 such that ... (alternating quantifiers) such that, when the variables $t_1, \ldots, t_r$ are set to true and all other variables are set to false, formula X is true?
Parameter: r.

This problem is AW[*]-complete, by reduction from UNITARY PARAMETERIZED QBFSAT$_t$ (Lemma 10).

A.40 Vertex Cover

VERTEX COVER
Input: A graph $G = (V, E)$, positive integer k.

Question: Does there exist a subset $V' \subset V$ such that $|V'| \leq k$ and for every edge in E at least one endpoint is in V'?

This problem is NP-complete [43] and in FPT for parameter k [18].

A.41 Weighted Circuit Satisfiability

Weighted Circuit Satisfiability
Instance: A boolean circuit C; a positive integer k.
Question: Does C have a satisfying assignment of Hamming weight k?
Parameter: k

This problem is complete for W[P] [3].

A.42 Weighted t-Normalized Satisfiability

Weighted t-Normalized Satisfiability
Input: A t-normalized boolean expression X, a positive integer k.
Parameter: k
Question: Does X have a satisfying truth assignment of weight k? That is, a truth assignment where precisely k of the variables which appear in X are true?

This problem is complete for W[$t-1$] for all $t \geq 2$ [19].

A.43 Weighted Satisfiability

Weighted Satisfiability
Instance: A boolean formula F; a positive integer k.
Question: Does F have a satisfying assignment of Hamming weight k?
Parameter: k

This problem is complete for W[SAT] [4].

A.44 Winning Strategy

WINNING STRATEGY
Input: A bipartite directed $G = (V_1 \cup V_2, E)$, position $p \in V_1$, and sets $W \subseteq V$, $L \subseteq V$, $D \subseteq V$ of winning, lost, and drawn positions respectively.
Question: Does player one have a strategy which guarantees victory in the game described by G, W, L, and D from position p?

Note: We sometimes describe an instance of WINNING STRATEGY as a tuple $(V_1, V_2, E, p, W, L, D)$.

This problem is P-complete by reductions to and from ALTERNATING REACHABILITY in this thesis (Lemma 2), and can be viewed as the problem GAME on a graph input (GAME is also P-complete). The succinct version of this problem is SUCCINCT WINNING STRATEGY (A.36).

Appendix B

Games

It should be noted that in most cases these are just basic outlines of the rules and are provided only to familiarize the reader with the games discussed within this thesis. For any game with organized play (othello, chess, go, etc.) there are typically large international organizations which publish far more authoritative rule sets.

B.1 Tic-Tac-Toe

Tic-tac-toe is played on a 9x9 squares grid. First and second player alternate placing symbols in unused squares (X for first player, O for second). The first player to position three of their symbols along a straight line (horizontally, diagonally, or vertically) wins the game. If the board is completely filled without either player winning, the game is a draw.

The game graph of Tic-tac-toe is so small that brute-force analysis on a computer takes virtually no time at all. Weakly solving the game is quite feasible with just a pencil and paper. Most players intuitively come to the realization that the result under best play is a draw.

Tic-tac-toe is also sometimes called naughts-and-crosses.

B.2 Go-moku

Go-moku can be played as tic-tac-toe with the following differences: the board is 19x19 and a player must form a line of 5 pieces to win.

Go-moku is actually meant to be played with a go set, where the players use black

and white stones instead of X and O symbols and the stones are placed on a go board. A go board is actually an 18x18 square grid, but the stones are placed on the intersections which results in a 19x19 playing field.

B.3 Connect Four

Connect Four is played on a 7x6 square grid (seven columns, six rows). Players alternate choosing a column and placing one of their colored discs at the bottom of that column (generally, one player is red and the other is black). Once a column is full (each has the capacity for 6 discs), no more discs may be added to it. The first player to get four discs of his color in a straight line (horizontally, vertically, or diagonally) wins. If the entire board is filled without either player fulfilling the winning condition, the game is a draw.

B.4 Geography

Geography is a game which may be played by two or more players. Each player in turn must name a country. The first country named can be any country. Each following country named must begin with the letter which the previous country ended with. For example, a line of play might go: France, England, Denmark, and so on. When a player is unable to name a country which meets the letter criteria they are eliminated from the game. The last player remaining wins.

In practice, the "vocabulary" of the game is often augmented. Rather that working strictly from a list of countries, players may provide any place name which may include cities, bodies of water, continents, and so on. The geography theme itself is optional; the same rules could be applied to any sufficiently-large list or names and/or words. For example: animals, Hollywood actors and actresses, or five-letter words.

B.5 Othello

Othello is played on an 8x8 square grid with disc pieces which are white on one side and black on the other.

The starting position is shown in Figure B.1. The two players alternate taking turns. Each player's turn consists of placing a disc on the board such that his colour is

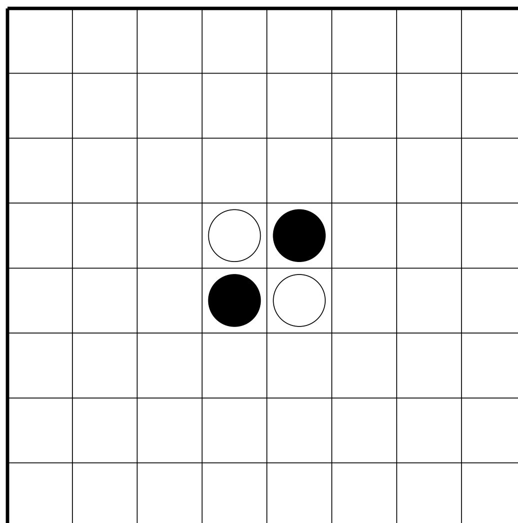

Figure B.1: The starting position for othello.

facing up (typically, black for player one and white for player two), such that there is a row of opposite-colored pieces between this newly-placed piece and a piece already on the board with the player's color. This row of pieces is then flipped, effectively joining the player who just moved. If such a move is impossible, the game ends.

When the game ends, each player counts the number of pieces of his color. The player with the greater number wins.

B.6 Hex

Hex is played on an unconventional board formed of hexagons (see Figure B.2). The classic hex board is 11x11 (diagonally). At the beginning of the game, first player choses two opposite sides of the board (the other two are assigned to second player). Each player's goal is to claim hexes in a manner which creates a path between his two sides of the board. This claiming of hexes takes place in a manner very similar to tic-tac-toe or go-moku – players alternate placing stones of their color on unoccupied hexes.

Creating a path from one side of the board to the opposing side makes it impossible to connect the other two opposing sides. Conversely, once all the hexes have been claimed there must be a path between one pair of opposing sides. Therefore there is always exactly one winner in a game of Hex.

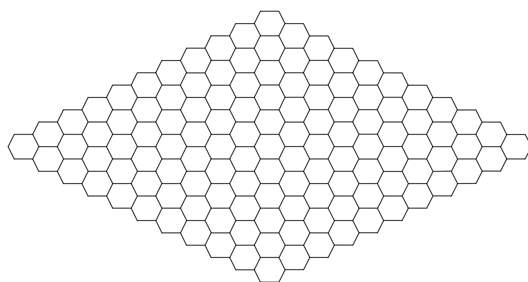

Figure B.2: An 11x11 hex board.

Hex was invented by Piet Hein in 1942, then rediscovered in 1948 by John Nash.

B.7 Checkers

Checkers is played on an 8x8 square grid. Each player controls one set of coloured discs (generally red for one player and black for the other). The pieces are initially deployed as in Figure B.3.

Players alternate taking moves. A move consists of moving a piece one square diagonally, or hopping over an opposing piece in a diagonally adjacent square (in effect, moving two squares). In the latter case, the piece hopped over is captured (removed from the board). A player may chain several captures together in one move, so long as they all performed by moving the same piece. Pieces must always be moved toward the opponent's side of the board unless they are kings. A piece becomes a king by reaching the opponent's side of the board.

A player loses when all his pieces are removed from the board.

Checkers is also known as draughts. Some variants are played on a 10x10 board. Some variant rules require that players make a capture so long as one is available.

B.8 Chess

Chess is played on an 8x8 square grid. Each player controls a coloured set of pieces (white for first player, black for second player). There are several different kinds of pieces, each of which can be moved in a different manner:

Pawn - Pawns may be moved one square forward, except when moving from their original position when they may be moved two squares instead. Unlike other pieces,

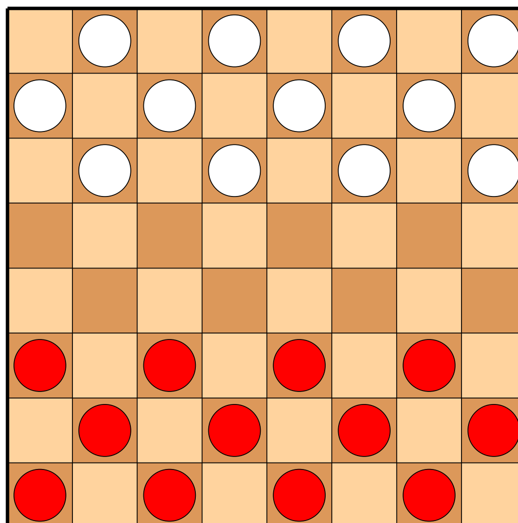

Figure B.3: The starting position for checkers.

pawns cannot capture with their standard move, but they can capture pieces by moving one square forward diagonally. They are only allowed to move diagonally when performing a capture.

Rook - A rook can move any number of squares horizontally or vertically, so long as there are no pieces between the start and end positions.

Knight - A knight can move in an L-shape: two squares horizontally or vertically, and then one square perpendicular to the first motion. Unlike other pieces, knights can move through occupied squares.

Bishop - A bishop can move any number of squares diagonally, so long as there are no pieces between the start and end positions.

Queen - A queen has the movement options of both a rook and a bishop.

King - A king can move exactly one square in any direction, and never into check. The only time the king can move more than one square is with a special move known as castling. When castling, the king moves two squares toward one of his rooks, while the rook moves to the square between the king's original and new positions. This special move can only be performed when both the king and the rook it is castling with not have previously moved this game, there are no pieces between them, and no square the king moves through is in check.

All pieces except pawns can capture an opposing piece by moving into the square it occupies. Pawns obey different rules when capturing, as explained with their move-

ment rules. A captured piece is removed from the board.

A player is in check if one of his opponent's pieces could capture his king. A player is in checkmate if he is in check and has no available move which would remove him from check. The first player to checkmate his opponent wins the game.

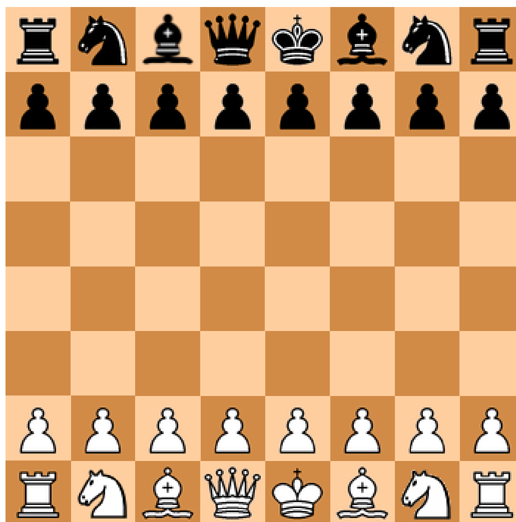

Figure B.4: The starting position for chess.

B.9 Go

Go is played on a square grid. It is an exception in that unlike other games played on square grids, pieces are placed on the intersections of the grid rather than within the squares of the grid. While the square grid is 18x18, the intersections form a 19x19 playing field.

Each player is assigned a piece color (white or black). A move consists of placing a single piece on an unoccupied grid intersection.

A piece may remain on the board only so long as it is adjacent to an empty intersection (diagonals to not count – adjacency is taken from the grid lines), or if there is some path through like-colored pieces which connects it to an empty intersection. Any piece which cannot meet either of these conditions is removed from the board and claimed by the opposing player.

Ending and scoring the game can be convoluted procedures. Generally, the game ends when both players agree that there are no more meaningful moves to be made (in

effect, they both believe that every intersection on the board is effectively "controlled" by some player and that this control cannot be stolen away). Scoring takes into account pieces captured, intersections controlled, and who went first. Very roughly, a player controls an intersection if the players agree that any opposing piece placed there would inevitably be captured. In practice, this process is generally more obvious than it sounds. Exact scoring details (the values of captured pieces, controlled intersections, and the penalty for going first) vary from region to region. The player with the higher score wins.

Appendix C

Brainstones

```
algorithm is_terminal(board b)
   // First check the horizontals and the verticals.
   for i := 0 to 3
      if b[i,0] = b[i,1] and b[i,0] = b[i,2] and b[i,0] = b[i,3] then
         if b[i,0] = 1 return 1 else return -1
      if b[0,i] = b[1,i] and b[0,i] = b[2,i] and b[0,i] = b[3,i] then
         if b[0,i] = 1 return 1 else return -1
   // Now check the diagonals.
   if b[0,0] = b[1,1] and b[0,0] = b[2,2] and b[0,0] = b[3,3] then
      if b[0,0] = 1 return 1 else return -1
   if b[3,0] = b[2,1] and b[3,0] = b[1,2] and b[3,0] = b[0,3] then
      if b[3,0] = 1 return 1 else return -1
   return 0
end algorithm
```

```
algorithm t(board b, integers x1, y1, x2, y2, x3, y3, x4, y4)
   if b[x4,y4] = 2 then return false
   return not (b[pair[x3,y3]] != 1 and (x1 != x3 or y1 != y3))
end algorithm
```

```
algorithm evaluate_board(board b, flags[] f)
```

```
i := trinary_to_binary(b)
if f[i].is_terminal then return false
result := false
all_win := true
can_move := false
// Check all possible moves from this position.
for x1 = 0 to 3
   for y1 = 0 to 3
      if b[x1,y1] != 0 then continue
      for x2 = 0 to 3
         for y2 = 0 to 3
            if b[x2,y2] != 2 and b[pair[x1,y1]] = 1 then continue
            for x3 = 0 to 3
               for y3 = 0 to 3
                  if b[x3,y3] = 0 then
                     for x4 = 0 to 3
                        for y4 = 0 to 3
                           if t(b, x1, y1, x2, y2, x3, y3, x4, y4)
                              then continue
                           // This is a valid move.
                           can_move := true
                           // Calculate hash value for this move.
                           m := b
                           m[x1,y1] := 1
                           if m[pair[x1,y1]] = 1 then m[x2,y2] = 0
                           m[x3,y3] := 1
                           if m[pair[x3,y3]] = 1 then m[x4,y4] = 0
                           j := trinary_to_binary(swap_1_and_2(m))
                           // Check flags for result of this move.
                           if f[j].is_loss then
                              // Have a winning move.
                              if not f[i].is_win then
                                 result := true
                                 f[i].is_win := true
                                 f[i].is_loss := false
```

```
                              return result
                           // Can we at least avoid losing?
                           if not f[j].is_win then
                              all_win := false
   // Check whether this was an undiscovered terminal node.
   if not can_move then
      f[i].is_terminal := true
      return true
   // Check whether all children are winning (this node loses).
   if all_win then
      if not f[i].is_loss then
         result := true
         f[i].is_loss := true
         f[i].is_win := false
      return result
   // Mark this node as a draw.
   if not f[i].is_loss and not f[i].is_win then return false
   f[i].is_loss = false
   f[i].is_win = false
   return true
end algorithm
```

```
algorithm solve_brainstones
   // Step #1 - Flag all the terminal positions.
   for i := 0 to 3^16 - 1
      board := binary_to_trinary(i)
      state := is_terminal(board)
      if state != 0 then flags[i].is_terminal := true
      if state = 1 then flags[i].is_win := true
      if state = -1 then flags[i].is_loss := true
   // Step #2 - Update positions until there are no more updates.
   do
      node_changed := false
      // Working backwards proliferates win/loss values faster.
```

```
    for i := 3^16 - 1 down to 0
      board := binary_to_trinary(i)
      if evaluate_node(board, flags) then node_changed := true
  while (node_changed)
  // Step #3 - Check if starting position is a win for first player.
  return flags[0].is_win
end algorithm
```

The function `binary_to_trinary(i)` takes an integer i and returns the board position corresponding to that position. Similarly, `trinary_to_binary(board)` takes a board position and returns the integer corresponding to that position. This correspondence is easy, since we can treat the board as a trinary number by asserting that an empty square is zero, a white stone is one, a black stone is two, and always using some consistent ordering of the squares on the board.

The function `swap_1_and_2(board)` changes all the 1s in board to 2s and vice-versa. This effectively means the stone colors are swapped. We do this as a time-saving shortcut; we always assume that white is the next player to move, so when presented with a position where black should move next we can simply swap the colors of the stones.

The pair array is a predefined constant. When given some coordinate pair, it returns the coordinate pair of the square whose symbol matches the symbol of the given coordinate pair.