Thesis Proposal: An Application of Game Characterizations to Propositional Proof Complexity

Alexander Hertel *

April 4, 2007

1 Introduction

At our last checkpoint together, we reviewed a number of results and laid out a research plan. The purpose of this paper is to present the research completed since then and to propose how these results can be combined with those from the previous checkpoint into a Ph.D thesis.

To review, our previous checkpoint described the following results:

1. The development of a Non-Hamiltonicity Proof System (NHPS), together with proofs of soundness, completeness, exponential lower bounds, and simulation results relating it to other proof systems.
2. Formalizing the concept of dangerous reductions; We were able to show that slightly different versions of the reduction from Hamiltonian Cycles to SAT give results with drastically different proof complexities.
3. Prover / Delayer Game Upper bounds for Tree Resolution proof size.
4. The proof complexity of Intuitionistic Propositional Logic.

Since then, our Dangerous Reductions research [HHU07] was accepted at the SAT 2007 conference in Lisbon. In light of Pavel Hrubes' recent impressive Intuitionistic Logic lower bounds [Hru06], we are currently in the process of reworking our Intuitionistic Logic paper, which needs to be modified to take the new results into account. In unrelated research, [HHM07] was recently accepted by the Notre Dame Journal of Formal Logic.

The research completed since the last checkpoint falls in the intersection between graph theory, proof complexity, and computational complexity, and the rest of this paper is dedicated to presenting it. This research is related to work done by Prof. Pitassi and Philipp Hertel [HP07], and is organized in this paper as follows: in Section 2, we define a number of important concepts including pebbling and Resolution space. In Section 3, we describe some of the most important related previous work.

Next we prove our four main results, the first three of which are stated as a series of equivalence results between different forms of resolution space and pebbling, followed immediately by their implications for complexity theory.

In Section 4 we show that for any Lingas Circuit C, B-$Peb(C)$ (with sliding) is exactly equal to the Tree Resolution clause space of refuting the formula $Peb^2(C)$. Since pebbling Lingas Circuits is known to be $\mathcal{PSPACE}$-Complete, and since Tree Clause Space is known to be equal to Prover / Delayer Number, an immediate corollary to this equivalence is that both the Tree Clause Space problem as well as the

*The author gratefully acknowledges the support of NSERC and the University of Toronto Department of Computer Science.

Prover / Delayer Game are also $\mathcal{PSPACE}$-Complete. We are currently considering where to submit these results; CSL 2007 is one possibility.

In Section 5 show that for any binary DAG G, $B\text{-}Peb(G)$ (with sliding) is equal to within a constant of the Input Resolution total space of a slight modification of the formula $Peb^1(G)$. This has two immediate corollaries. The first is the $\mathcal{PSPACE}$-Completeness for various forms of the Input Resolution total space problem. These results suggest that a new respect for Input Resolution is in order, since it is normally considered to be trivial. The second corollary is an extreme (and optimal) total space / size tradeoff for Input Resolution. More specifically, we show that there exists an infinite family of formulas whose Input Resolution proofs with minimum required total space have size $2^{\Omega(n)}$. However, if only one more unit of total space is permitted, then the size drops to only $O(n)$, where n is the number of variables.

In Section 6 we prove a result showing that for any monotone circuit C, $BW\text{-}Peb(C)$ (without sliding) is exactly equal to the Resolution variable space of refuting $Peb^1(C)$. The $\mathcal{PSPACE}$-Hardness of the Resolution Variable Space problem follows as a corollary to this fact as well as the $\mathcal{PSPACE}$-Completeness of black-white pebbling recently proven in [HP07].

Finally, in Section 7, we prove the $\mathcal{EXPTIME}$-Completeness of the Resolution Width Problem. This result has been submitted to FOCS 2007.

We propose that together with the work presented during our last checkpoint, this new research constitutes the basis for a Ph.D thesis. A recurring theme in this proposed thesis is the application of game characterizations to propositional proof complexity. For example, the Prover / Delayer Game is used extensively with the NHPS research, comes up again in the context of dangerous reductions, and obviously again in the results concerning it. The results concerning Resolution space are heavily dependent on pebbling game characterizations of space, and the final result about Resolution width depends on the Existential k-Pebble Game. In fact, our only results which do not use game characterizations are those concerning Intuitionistic Logic, but those are related to the rest of the work in that they provide an important example of dangerous reductions. All of the results above are therefore thematically related, and it would not be hard to combine them into a coherent thesis.

2 Definitions

2.1 Resolution, Size, & Width

The notions of size and width can be formalized using a fairly simple definition of what constitutes a RES proof, whereas the notions of space require slightly more complicated definitions. We assume that the reader is already somewhat familiar with the basics of proof complexity and the RES proof system, and shall use [CK01] as our reference.

Definition 2.1 (RES Proof). *A clause C is a set of literals. We use the notation $C \vee D$ for the clause $C \cup D$, and write $C \vee l$ for $C \cup \{l\}$, where l is a literal. If $C \vee x$ and $D \vee \neg x$ are clauses, then the resolution rule allows us to derive the clause $C \vee D$, by resolving on the variable x. If F is a set of clauses, then the sequence of clauses $\pi = C_1, C_2, ..., C_k$ is a* RES *proof of C_k from F if each C_i in π appears in F (i.e. is an input, or initial clause) or follows from two previous clauses in π by the resolution rule.*

*If the graph underlying the structure of π is a tree (i.e. each clause in π is a premise for at most one application of the resolution rule), then the proof is said to be a Tree-Like Resolution (*T-RES*) proof. Otherwise it is said to be DAG-Like. A* RES *refutation of F is a* RES *proof from F in which $C_k = \emptyset$ (the empty clause).*

This allows us to define size and width:

Definition 2.2 (Resolution Size & Width). *If a* RES *proof π of formula F contains k clauses, then it is said to have size k. The width of a clause C refers to how many literals it contains, and is denoted $w(C)$. The width of a formula F is the width of the widest clause in F, and is denoted $w(F)$. The* *width of a* RES *refutation* π*, denoted* $w(\pi)$ *is equal to the width of its widest clause, and is denoted* $w(\pi)$*. Finally, the minimum width of any* RES *refutation of* F *is denoted* $w(F \vdash_{\mathsf{RES}} \emptyset)$*.*

2.2 Resolution Space

The definitions of the various different measures of space that we shall be discussing requires an alternative definition of RES proof which depends on the notion of configuration:

Definition 2.3 (Configuration-Style RES Proof). *A configuration* $\mathbb{C}$ *is a set of clauses. If* F *is a set of clauses, then the sequence of configurations* $\pi = \mathbb{C}_0, \mathbb{C}_1, ..., \mathbb{C}_k$ *is a* RES *proof of* C *from* F *if* $\mathbb{C}_0 = \emptyset$*,* $C \in \mathbb{C}_k$*, and for each* $i < k$*,* $\mathbb{C}_{i+1}$ *is obtained from* $\mathbb{C}_i$ *by one of the following rules:*

1. *deleting one or more of its clauses,*
2. *adding the resolvent of two clauses of* $\mathbb{C}_i$*,*
3. *adding one or more of the clauses of* F *(initial clauses).*

The proof π *is said to be Tree-Like if we replace rule* 2 *with the following:*

2. adding the resolvent of two clauses of $\mathbb{C}_i$ ***and deleting the parent clauses****.*

In addition, π *is said to be an* I-RES *proof if at least one input to every instance of the resolution rule is an input clause, and it is said to be an* I-RES-W$^-$ *proof if we add the following rule:*

4. replacing one of the clauses $C \in \mathbb{C}_i$ *with the clause* $C \cup \{\neg x\}$*, which was obtained by weakening* C *with an arbitrary negative literal* $\neg x$*.*

Once again, if $\emptyset \in C_k$*, then we refer to* π *as a refutation.*

This leads us to our different definitions of space. Intuitively, space is the amount of memory required in order to compute π. For each of these measures, it is important to distinguish between space for RES, T-RES, and I-RES. Note that our terminology differs from that introduced by Ben-Sasson in [BS02]. Our notions of 'Clause Space' are identical, but what Ben-Sasson refers to as 'Variable Space', we refer to as 'Total Space'. Our notion of 'Variable Space' is not explicitly defined in [BS02], but does appear implicitly in one of the results (See Corollary 3.4 below).

Definition 2.4 (Clause Space). *Let* F *be a set of clauses and* π *be a* RES *configuration-style proof of clause* C *from* F*. The Clause Space of a configuration* $\mathbb{C}$ *in* π*, denoted* $CS(\mathbb{C})$ *is the number of clauses in* $\mathbb{C}$*. The Clause Space of* π*, denoted* $CS(\pi)$ *is the maximum* $CS(\mathbb{C})$ *over all* $\mathbb{C} \in \pi$*. Finally, the Clause Space of resolving* C *from* F*, denoted* $CS(F \vdash_{\mathsf{RES}} C)$*, is the minimum* $CS(\pi)$ *over all* RES *proofs* π *of* C *from* F*.*

Definition 2.5 (Total Space). *Let* F *be a set of clauses and* π *be a* RES *configuration-style proof of clause* C *from* F*. The Total Space of a configuration* $\mathbb{C}$ *in* π*, denoted* $TS(\mathbb{C})$ *is defined as* $\sum_{C \in \mathbb{C}} w(C)$*. The Total Space of* π*, denoted* $TS(\pi)$ *is the maximum* $TS(\mathbb{C})$ *over all* $\mathbb{C} \in \pi$*. Finally, the Total Space of resolving* C *from* F*, denoted* $TS(F \vdash_{\mathsf{RES}} C)$*, is the minimum* $TS(\pi)$ *over all* RES *proofs* π *of* C *from* F*.*

Definition 2.6 (Variable Space). *Let* F *be a set of clauses and* π *be a* RES *configuration-style proof of clause* C *from* F*. The Variable Space of a configuration* $\mathbb{C}$ *in* π*, denoted* $VS(\mathbb{C})$ *is the number of distinct variables in* $\mathbb{C}$*. The Variable Space of* π*, denoted* $VS(\pi)$ *is the maximum* $VS(\mathbb{C})$ *over all* $\mathbb{C} \in \pi$*. Finally, the Variable Space of resolving* C *from* F*, denoted* $VS(F \vdash_{\mathsf{RES}} C)$*, is the minimum* $VS(\pi)$ *over all* RES *proofs* π *of* C *from* F*.*

In the case of T-RES, we abbreviate Clause Space, Total Space, and Variable Space as TCS, TTS, and TVS, respectively.

It is easy to see that for all F, $CS(F \vdash_{\mathsf{RES}} C) \leq TS(F \vdash_{\mathsf{RES}} C)$, and $VS(F \vdash_{\mathsf{RES}} C) \leq TS(F \vdash_{\mathsf{RES}} C)$. The same holds for the Tree-Like cases. $VS(F \vdash_{\mathsf{RES}} C)$ and $CS(F \vdash_{\mathsf{RES}} C)$ are incomparable, since each of these quantities can be made larger than the other by choosing an appropriate example for F.

2.3 Pebbling Circuits & Games

The investigation of RES space is closely associated with the well-known pebbling game and pebbling number of a DAG, originally explored in [Coo73, CS76] as a means of investigating bounds on storage requirements. There are several different versions of pebbling games, but the most popular ones are captured by the following generalized description:

Definition 2.7 (Pebbling Games on Monotone Circuits). *The generalized black-white pebbling game is a single-player game in which the goal is to 'pebble' a monotone circuit C in which each node is an AND gate, an OR gate, or a source. The leaves of C have in-degree 0 and are referred to as 'source' nodes. A single vertex in C is referred to as a 'target' or 'output' node, and has out-degree 0. All non-target nodes can have arbitrary out-degree, except of course in the case where C is a tree, in which case the maximum out-degree is 1. Similarly, all non-source nodes can have arbitrary in-degree. A circuit in which all non-source nodes have in-degree 2 is called a 'binary circuit'. The game involves two different types of pebbles (black and white), and has the following rules:*

1. *The game starts with no pebbles on the circuit.*
2. *A gate may have at most one pebble on it at any time.*
3. *At any point, the player may place any pebble onto a source node or remove any pebble from a source.*
4. *At any point, the player may remove any black pebble from any gate or place a white pebble on any empty gate.*
5. *For any unpebbled AND gate v, if all of a v's immediate predecessors have pebbles on them, then the player may place a black pebble on v. Alternatively, if present, the player may slide a black pebble from u to v, where u is a predecessor of v.*
6. *For any unpebbled OR gate v, if at least one of a v's immediate predecessors has a pebble on it, then the player may place a black pebble on v. Alternatively, if present, the player may slide a black pebble from u to v, where u is a predecessor of v.*
7. *For any AND gate v with a white pebble on it, if all of a v's immediate predecessors have pebbles on them, then the player may remove the white pebble from v. Alternatively, if all but one of v's predecessors are pebbled, then the player my slide the white pebble from v to the remaining unpebbled predecessor of v.*
8. *For any OR gate v with a white pebble on it, if at least one of v's immediate predecessors has a pebble on it, then the player may remove the white pebble from v. Alternatively, if at least one of v's predecessors is pebbled, then the player my slide the white pebble from v to any unpebbled predecessor of v.*
9. *The game ends once the circuit has a black pebble on the target node and there are no white pebbles present on any nodes.*

Most of the pebble games found in the literature can be viewed as restricted versions of this generalized game on monotone circuits. For example, if all nodes in C are AND gates, then we usually refer to it using the letter 'G' and call it a 'DAG'. The game resulting from this restriction is the standard 'Black-White Pebbling Game'. Similarly, we can further restrict the game by disallowing the use of white pebbles, resulting in the standard 'Black Pebbling Game'. The Black Pebbling Game and the Black-White Pebbling Game played on DAGs are the most common forms of the game found in the literature, although pebbling on monotone circuits has also been investigated. This leads us to the definitions of the 'Black-White pebbling number' and the 'black pebbling number' of a monotone circuit:

Definition 2.8 (Pebbling Numbers of Monotone Circuits). *Given a monotone circuit C, the 'Black-White pebbling number' of C, denoted $BW\text{-}Peb(C)$ is the minimum number of total pebbles that the player needs in order to pebble C's target node without violating any rules of the Black-White Pebbling Game.*

The 'black pebbling number' of C, denoted $B\text{-}Peb(C)$ is the minimum number of black pebbles that the player must be given in order to pebble C's target node without violating any rules of the Black Pebbling Game.

Note that each version of the pebbling game can either be defined with or without sliding; this is an important point because allowing sliding has a slight effect on the pebbling number (see Lemma 3.1 for details). Another issue is whether sliding is compulsory or optional, i.e. if one is allowed to place a pebble even when sliding is possible.

2.4 Pebbling Contradictions

A number of important groups of unsatisfiable formulas called the 'Pebbling Contradictions' are based on the various different forms of the pebbling game. A brief history of how these formulas have been used in the literature is given in [BS02].

Definition 2.9 (One-Colour Pebbling Contradictions for Monotone Circuits). *The One-Colour Pebbling Contradiction of a monotone circuit C, denoted $Peb^1(C)$, is an unsatisfiable formula constructed by taking C, and creating the following clauses in which each variable x is interpreted as meaning that vertex x has a pebble on it:*

1. *For each source node s, create the singleton clause $\{s\}$.*
2. *For each AND node y_0 of degree d with immediate predecessors $y_1, y_2, ..., y_d$, create the propagation clause $\{\neg y_1, \neg y_2, ..., \neg y_d, y_0\}$.*
3. *For each OR node y_0 of degree d with immediate predecessors $y_1, y_2, ..., y_d$, create the propagation clauses $\{\neg y_1, y_0\}, \{\neg y_2, y_0\}, ..., \{\neg y_d, y_0\}$.*
4. *Finally, for the target node t, create the singleton clause $\{\neg t\}$.*

We shall also make use of a more complicated type of contradiction, the two-colour pebbling contradictions:

Definition 2.10 (Two-Colour Pebbling Contradictions for Binary Monotone Circuits). *The Two-Colour Pebbling Contradiction of a binary monotone circuit C, denoted $Peb^2(C)$, is an unsatisfiable formula constructed by taking C, and creating the following clauses in which each variable x_i is interpreted as meaning that vertex x has a pebble on it:*

1. *For each source node s, create the singleton clause $\{s_0, s_1\}$.*
2. *For each AND node c with immediate predecessors a and b, create four propagation clauses $\{\neg a_0, \neg b_0, c_0, c_1\}$, $\{\neg a_0, \neg b_1, c_0, c_1\}$, $\{\neg a_1, \neg b_0, c_0, c_1\}$, and $\{\neg a_1, \neg b_1, c_0, c_1\}$.*

3. *For each OR node c with immediate predecessors a and b, create four propagation clauses $\{\neg a_0, c_0, c_1\}$, $\{\neg b_0, c_0, c_1\}$, $\{\neg a_1, c_0, c_1\}$, and $\{\neg b_1, c_0, c_1\}$.*
4. *Finally, for the target node t, create the two singleton clauses $\{\neg t_0\}$ and $\{\neg t_1\}$.*

It should be easy to see that for any monotone circuit C, both $Peb^1(C)$ and $Peb^2(C)$ are unsatisfiable; this is because the source node variables must all be true, and all of the propagation clauses send this truth towards the target. The target variable must therefore be true, but the negative singleton clause(s) for the target forces it to be false, ultimately creating a contradiction.

It is also worth noting that the clauses of $Peb^1(G)$ are all Horn clauses, meaning that each one contains at most one positive literal.

2.5 The Prover / Delayer Game

The Prover / Delayer Game, described in [PI00, BSIW04], is a combinatorial game between two players, the 'Prover', and the 'Delayer'. The game is played on an unsatisfiable CNF formula F. The point of the game is for the Prover to falsify some initial clause of F, thereby falsifying the formula. Since the formula is unsatisfiable, this is inevitable. Roughly speaking, the Delayer's goal is to delay the falsification of the formula for as long as possible.

The game proceeds in rounds. Each round starts with the Prover choosing a variable, and asking the Delayer what the value of that variable is. The Delayer can give one of three answers:

- True
- False
- You Choose

If the Delayer says 'You Choose', then the Prover gets to decide the value of that variable. In addition, every time 'You Choose' is said, the Delayer wins one point. This is the only way in which points can be scored.

The game finishes when some clause has been falsified. The real goal of the game is not actually to prove or delay; rather, the Delayer's aim is to win as many points as possible, while the Prover's aim is to make sure that the Delayer wins as few as possible. This leads us to the following definition:

Definition 2.11. *Let F be an unsatisfiable CNF formula. The Prover / Delayer number of F, denoted $PD(F)$ is the greatest number of points the Delayer can score on F with the Prover playing optimally.*

3 Results Related to Resolution Space

3.1 Pebbling

An important survey on pebbling is [Pip80]. In addition, the literature contains some useful results for manipulating pebbling strategies. The first is a connection between normal pebbling and pebbling with sliding. In [GLT80], the authors prove that any DAG G can be black pebbled using k pebbles with sliding if and only if it can be black-pebbled using $k + 1$ pebbles without sliding. We generalize this result to the black-white pebbling of monotone circuits:

Lemma 3.1. *Any monotone circuit C can be black-white pebbled using k pebbles with sliding if and only if it can be black-white pebbled using $k + 1$ pebbles without sliding.*

Proof: Let C be any arbitrary monotone circuit.

$\Rightarrow$ Suppose that C can be black-white pebbled using k pebbles with sliding. Replace each instance of the sliding rule from vertex x to vertex y with two moves; first place a pebble on y, and then remove the pebble from x. This substitution works regardless of whether the pebble is white or black, and leaves us with a pebbling strategy that involves no sliding and requires $k+1$ pebbles, as required.

$\Leftarrow$ Suppose that C can be black-white pebbled using $k+1$ pebbles without sliding, and let H be the pebbling strategy associated with this pebbling.

Let (B_i, w_i) be any time in H containing $k+1$ pebbles. Therefore, the move transitioning from step (B_{i-1}, w_{i-1}) to (B_i, w_i) was the placement of either a black or white pebble on a vertex x. Either this is the final move in the pebbling or it is not. If this is the final move, then it must be the placement of a black pebble. Then all of x's predecessors must be pebbled at step (B_{i-1}, w_{i-1}), and there are no white pebbles on C, so simply slide one of the black pebbles from one of x's predecessors to x instead of placing a new black pebble on x, thereby saving a pebble.

If this is not the final move in the pebbling, then the pebble being placed on x is either black or white, and the move transitioning from step (B_i, w_i) to step (B_{i+1}, w_{i+1}) must be the removal of a pebble from a vertex y. If the pebble being placed on x is white, then it does not require any predecessors, so simply remove the pebble from y before placing the white pebble on x, thereby saving a pebble.

If the pebble being placed on x is black, then y either is or is not an immediate predecessor of x. If y is not an immediate predecessor, then it is not needed for the placement of x, so simply remove y first, and then place the black pebble on x, thereby saving a pebble.

If y is an immediate predecessor of x, then the pebble being removed from y is either black or white. If it is black, then instead of placing a black pebble on x and then removing the black pebble from y, we simply slide the black pebble from y to x, thereby saving a pebble and still ending up in the same pebbling configuration (B_{i+1}, w_{i+1}).

If the pebble being removed from y is white, then instead of placing a black pebble on x and removing a white pebble from y, we first remove the white pebble from y, then place a black pebble on y, and then slide it from y to x, once again saving a pebble and ending up in the same pebbling configuration (B_{i+1}, w_{i+1}).

We simply perform this local transformation to save a pebble at every stage of H where $k+1$ pebbles are used. This shows that C can be pebbled using k pebbles with sliding, as required.

□

Another useful result is one which allows us to take the 'dual' of any black-white pebbling:

Lemma 3.2 ([Hei81]). *Given a Black-White pebbling strategy (without sliding) of a DAG G using at most k pebbles, if one reverses the strategy, converts all black pebbles to white, and all white pebbles to black, then the resulting strategy is also a valid Black-White pebbling strategy of G which uses at most k pebbles.*

3.2 Space & Games

The pebbling game is related closely to RES clause space. Let F be any arbitrary unsatisfiable formula, let π be a configuration-style RES refutation of F, and let G be the DAG underlying the structure of π. It is not hard to see that the clause space used in computing π is exactly equal to $B\text{-}Peb(G)$. More specifically, $CS(F \vdash_{\mathsf{RES}} \emptyset)$ is equal to the pebbling number of the DAG with the smallest pebbling number of all DAGs underlying valid RES refutations of F. This shows that both general clause space as well as tree clause space are related closely to the pebbling game.

A particularly relevant result relating Black-White pebbling number to space was proved by Ben-Sasson:

Theorem 3.3 ([BS02]). *For any monotone circuit C, $BW\text{-}Peb(C) \leq TS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset)$, where $BW\text{-}Peb(C)$ is defined without sliding.*

Implicit in the proof of this Theorem is the following Corollary:

Corollary 3.4 ([BS02]). *For any monotone circuit C, $BW\text{-}Peb(C) \leq VS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset)$, where $BW\text{-}Peb(C)$ is defined without sliding.*

Tree clause space is also related closely to the Prover / Delayer Game, described in Section 2.5. The results in [ET03] are the definitive work relating the Prover / Delayer number $PD(F)$ of an unsatisfiable CNF formula to its tree clause space $TCS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$ by showing the following:

Theorem 3.5 ([ET03]). *For any unsatisfiable CNF formula F, $TCS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) = PD(F) + 1$.*

In other words, the Prover / Delayer game perfectly captures the notion of clause space for T-RES.

In addition to doing some of the pioneering research in the area of Resolution space, Esteban & Torán also showed that an upper bound on tree clause space gives an upper bound on the size of T-RES proofs:

Theorem 3.6 ([ET01]). *Let F be an unsatisfiable formula on n distinct variables. If F has a* T-RES *refutation with tree clause space $s = TCS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$, then it has a* T-RES *refutation of size $\binom{n+s}{s}$.*

By substituting an upper bound for the expression $\binom{n+s}{s}$, it is easy to see that combining Theorems 3.5 and 3.6 shows that an upper bound on the number of points scored by the Prover immediately gives an upper bound on T-RES size:

Corollary 3.7. *If the Prover has a strategy limiting the Delayer to at most k points playing on formula F which contains $\leq n$ distinct variables, then F has a* T-RES *proof of size $\leq e^{k+1}(\frac{n}{k+1} + 1)^{k+1}$.*

The Prover / Delayer Game can also be used to give a lower bound on T-RES size; in [BSIW04], the authors prove that lower bounds on $PD(F)$ can be used to prove lower bounds on the size of T-RES proofs:

Theorem 3.8 ([BSIW04]). *If an unsatisfiable CNF formula F has a* DPLL *tree of size $\leq 2^k$, then the Prover has a strategy limiting the Delayer to $\leq k$ points.*

Proof: Let F be any arbitrary unsatisfiable formula with a DPLL tree of size $\leq 2^k$. The Prover uses this tree as a strategy to limit the Delayer to at most k points as follows: the prover starts by querying the variable corresponding to the root of the tree. If the Delayer says 'True' or 'False', then the Prover proceeds by querying the variable in the corresponding subtree of the root. If the Delayer says 'You Choose', then the Prover chooses the smaller subtree, and next queries on that variable. Therefore, even if the Delayer says 'You Choose' on every query, 2^k can only be divided in half at most k times, so the Delayer wins at most k points, as required.

$\square$

The contrapositive of this theorem gives us the following Corollary, giving a direct connection between the Prover / Delayer Game and T-RES size lower bounds:

Corollary 3.9. *If the Delayer has a strategy guaranteed to win $> k$ points on F, then every* T-RES *refutation of F has size $> 2^k$.*

In other words, upper and lower bounds on the Prover / Delayer Number of a formula not only immediately imply corresponding upper and lower bounds on tree clause space (since Prover / Delayer Number and tree clause space are basically synonymous), but they also imply upper and lower bounds for T-RES proof size.

3.3 Space & Width

An important paper relating space and width is [AD03]. In it the authors prove that for unsatisfiable k-CNF formulas, space is an upper bound on width.

Theorem 3.10 ([AD03]). *Let F be any unsatisfiable CNF formula. Then*

$$CS(F \vdash_{\mathsf{RES}} \emptyset) \geq w(F \vdash_{\mathsf{RES}} \emptyset) - w(F).$$

This is a very powerful result, because it can be used to derive space lower bounds for all formulas for which width lower bounds are known.

3.4 Width & Size

The relationship between width & size is very important because it shows that lower bounds for width imply lower bounds for size. This simplifies proofs for size lower bounds by allowing us to focus on lower bounds for width. The main result linking width and size is [BSW01], and is also explained well in [Urq06]:

Theorem 3.11 ([BSW01]). *Let F be a contradictory set of clauses with an underlying set of variables V, and let $S(F \vdash_{\mathsf{RES}} \emptyset)$ be the minimum size of any* RES *refutation of F. Then*

$$S(F \vdash_{\mathsf{RES}} \emptyset) = exp\left(\Omega\left(\frac{(w(F \vdash_{\mathsf{RES}} \emptyset) - w(F))^2}{|V|}\right)\right)$$

3.5 Tradeoff Results

Another important paper is [BS02], in which the author proves a number of interesting tradeoff results. For example, he provides families of formulas for which there are:

- linear size T-RES proofs and constant width T-RES proofs, but no T-RES proof that has both small width and small size.
- RES proofs with constant clause space, and RES proofs with constant width, but no RES proof that has both small width and small clause space.
- linear size RES proofs that also have constant width, but no RES proof that has both small clause space and small size.

These results rely on the pebbling contradiction formulas from Definition 2.9 above.

A more recent paper in the same vein is [Nor06]. In it, the author proves a tradeoff result for Resolution width and clause space. More specifically, he proves the existence of a family of unsatisfiable formulas with constant width but clause space which is bounded by $\Theta(lg(n))$, where n is the formula length.

3.6 Complexity of Pebbling

A number of complexity results involving pebbling are known. In [Lin78], the author proves the following interesting result about the black pebbling game on monotone circuits:

Theorem 3.12 ([Lin78]). *Given a monotone circuit C and an integer k, the problem of determining if C can be black-pebbled with k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

This result was later extended to DAGs in [GLT80]:

Theorem 3.13 ([GLT80]). *Given a DAG G and an integer k, the problem of determining if G can be black-pebbled with k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

More recently, the black-white pebbling game on monotone circuits was also shown to be $\mathcal{PSPACE}$-Complete [HP07]:

Theorem 3.14 ([HP07]). *Given a monotone circuit C and an integer k, the problem of determining if C can be black-white-pebbled with k pebbles (with sliding) is $\mathcal{PSPACE}$-Complete.*

By Lemma 3.1, we get the following Corollary:

Corollary 3.15. *The $\mathcal{PSPACE}$-Completeness of all three versions of the pebbling game holds regardless of whether or not sliding is allowed.*

All of these results are particularly interesting because they are $\mathcal{PSPACE}$-Completeness results for a game which has only one player, whereas most $\mathcal{PSPACE}$-Complete games have two.

4 Complexity of Tree Clause Space & Prover / Delayer Game

In this section we prove a number of results pertaining to pebbling, clause space, and the Prover / Delayer Game. In Section 4.1 we define a set of binary DAGs called the 'Increasing Binary DAGs' which can be solved in polynomial time. In Section 4.2 we follow up on this result by giving a Prover strategy for these pebbling graphs. Specifically, we show that for any Increasing Binary DAG G, when playing on the formula $Peb^2(G)$, the Prover has a strategy which limits the Delayer to at most $B\text{-}Peb(G)$ points.

Next, in Section 4.3 we define a set of monotone circuits called the 'Lingas Circuits' for which determining the pebbling number is $\mathcal{PSPACE}$-Complete. Section 4.4 contains a proof showing that for any monotone circuit C, the Delayer has a strategy which is guaranteed to score at least $B\text{-}Peb(C)$ points when playing on the formula $Peb^2(C)$. In Section 4.5 we combine these results with those from the previous sections to show a three-way equivalence between Black Pebbling Number, Tree Clause Space, and Prover / Delayer Number on both the Increasing DAGs and the Lingas Circuits.

Finally, in Section 4.6 we use this equivalence to prove the $\mathcal{PSPACE}$-Completeness of the Tree Clause Space Problem as well as the Prover / Delayer Game.

4.1 An Easy Case of the Pebbling Game

Although Pebbling Games in general are $\mathcal{PSPACE}$-Complete (see Section 3.6), in this section we define an interesting set of binary DAGs whose pebbling numbers can be computed in polynomial time. To this end, we first need to define the concept of the pebbling number of an internal vertex in a graph.

Definition 4.1 (Pebbling Number of a Vertex). *In a DAG, the pebbling number of a vertex x is simply the pebbling number of the subgraph rooted at x, with x set to be the target node.*

Given a binary DAG G, each node can be labelled with its pebbling number according to the definition above. Clearly, each source node has a pebbling number of 1. Consider each vertex c with predecessors a and b. Without loss of generality, we assume that the pebbling number of a is always less than or equal to that of b. There are only three possibilities for their pebbling numbers:

1. If vertex a has a pebbling number of at most k - 1 and vertex b has pebbling number k, then vertex c also has pebbling number k. To see this, simply pebble b with k pebbles, leave one pebble on b and remove the rest (if any). Then pebble a with the remaining k - 1 pebbles. Since a has a pebbling number strictly less than k, even when the extra pebble on b is taken into account, this cannot take more than k pebbles. Finally, slide one of the pebbles from a or b up to c.

2. If vertices a and b both have pebbling number k, then it is possible for c to have a pebbling number of $k + 1$.

3. If vertices a and b both have pebbling number k, then it is possible for c to also have a pebbling number of k.

It is impossible for a vertex to have a pebbling number which is greater than that of both of its predecessors by 2 or more, and it is also impossible for the pebbling number of a vertex to be less than that of its predecessors.

This observation gives rise to an interesting class of binary DAGs called the 'Increasing DAGs'. Intuitively, this set consists of all binary DAGs in which the pebbling number of each node is strictly greater than that of at least one of its predecessors. More formally,

Definition 4.2 (Increasing Binary DAGs). $\mathcal{G_I} = \{(G, k) \mid$ *G is a binary DAG with pebbling number k such that there is no $c \in V(G)$ with predecessors a and b with a, b, and c all having the same pebbling number.*$\}$

Although in general the pebbling game on binary DAGs is $\mathcal{PSPACE}$-Complete (see Corollary 3.15 above), when we restrict ourselves to inputs from $\mathcal{G_I}$, the problem becomes much easier:

Theorem 4.3. *Given a binary DAG and integer $(G, k) \in \mathcal{G_I}$ the problem of determining if G can be pebbled using at most k pebbles (with sliding) is in $\mathcal{P}$.*

Proof: Since the pebbling number of each node in G is uniquely determined by the pebbling numbers of its predecessors, simply start at the source nodes and set each of their pebbling numbers to 1. Then find a vertex x for which the pebbling numbers of both predecessors have been determined (since G is a DAG, such a node always exists), and set x's pebbling number. Repeat until the pebbling number of the target node has been determined. If it is at most k, then accept, and otherwise reject. Clearly this can be done in polynomial time.

$\square$

The pyramid graphs from [CS76] are good examples of graphs in $\mathcal{G_I}$.

This theorem gives some insight into why the Pebbling Game is so difficult. A close inspection of the constructions from both [Lin78] and [GLT80] shows that the circuits resulting from the reductions contain a large number of vertices which have the same pebbling numbers as their children, so the pebbling number of each vertex is not uniquely determined by that of its children, and the obvious algorithm fails.

4.2 Prover Strategy for the $\mathcal{G_I}$ DAGs

In this section we show that for any DAG and integer pair $(G, k) \in \mathcal{G_I}$, when playing the Prover / Delayer Game on $Peb^2(G)$, the Prover has a strategy which limits the Delayer to scoring at most $k = B\text{-}Peb(G)$ points (with sliding).

For the purposes of this proof, we will use the DAG G as a 'scaffold' to keep track of the Prover's progress. As such, we will assume that G is depicted with the target node t at the top and the sources at the bottom. The high-level idea of the Prover's strategy is to label each node in G with its pebbling number, and work down from the target towards the sources. The Prover must only give up a point when moving to a child with a lower pebbling number. The Prover will therefore be able to limit the Delayer to at most $B\text{-}Peb(G)$ points, since that is the pebbling number of t.

Since these are $\mathcal{G_I}$ graphs, there are two cases to consider. Given any node c with predecessors a and b, it is possible for a to have pebbling number $< k$ and both b and c to have pebbling number k. Apart from the case totally symmetrical to this one, the only other case we must consider is the one in which a and b have pebbling number k and c has pebbling number $k + 1$.

In the base case, both a and b are source nodes which have a pebbling number of 1. In this case clearly c has pebbling number 2. We know that every path from the targets downwards must eventually lead to a node with two sources as predecessors because paths cannot be infinitely long and the graphs are acyclic, so any path which the Prover follows will allow the Delayer to win at most $B\text{-}Peb(G)$ points.

This brings us to the Prover's strategy:

Lemma 4.4. *For any binary DAG and integer pair $(G, k) \in \mathcal{G}_{\mathcal{I}}$, when playing on the formula $Peb^2(G)$, the Prover has a strategy limiting the Delayer to at most $k = B\text{-}Peb(G)$ (with sliding) points.*

Proof: After the nodes have been labelled with their pebbling numbers, the Prover's strategy proceeds as follows: start at the target t and immediately query t_0. Since $(\neg t_0)$ is an initial clause, the Delayer must reply 'False', or else the game would immediately be over with no points scored. The Prover then queries t_1, which the Delayer must set to False for the same reason. Now the Prover tries to 'move down' the graph by having both of the variables associated with one of the predecessors of t set to False. In general, suppose that we are currently on node c with predecessors a and b, and both c_0 and c_1 have been set to False. What the prover does depends on which of the above cases a and b fall under.

Case 1: Assume that a and b have different pebbling numbers. Without loss of generality, assume that c and b have pebbling number k, a has pebbling number $< k$, where $k \geq 2$, and that both c_0 and c_1 have already been set to False. The Prover begins by querying variable b_0. If the Delayer says 'You Choose', then the Prover sets it to True. Next the Prover queries a_0. If the Delayer says 'You Choose', then the Prover sets a_0 to True and the game is over with 2 points scored because an initial propagation clause has been falsified. Since $k \geq 2$, the Prover would be content with this outcome. The Delayer would not set a_0 to True, since that would be even worse. If the Delayer sets a_0 to False, then the Prover immediately queries a_1, which the Delayer would have to set to False for the same reasons.

This means that the Delayer's responding 'You Choose' to b_0 has resulted in both a_0 and a_1 being falsified with only 1 point being scored, so the Prover has moved the game into a subgraph rooted at a node with a lower pebbling number without giving up more than 1 point. The Delayer would not set b_0 to True, because the Prover was willing to do this. The only other possibility is that the Delayer sets b_0 to False, at which point the Prover immediately queries b_1, which the Delayer must set to False for the same reasons.

Therefore in the case where the a and b have different pebbling numbers, either both of a_0 and a_1 are set to False with 1 point being scored, which means that the game has moved into a subgraph with a lower pebbling number, or both b_0 and b_1 are set to False with no points being scored, which means that the game has moved into a subgraph of equal pebbling number. The Delayer will favor this latter strategy if the difference between the pebbling number of a and c is more than 1.

Case 2: Assume that a and b have the same pebbling number, call it k, that the pebbling number of c is $k + 1 \geq 3$ (i.e. it is not the base case), and that both c_0 and c_1 have already been set to False. Note that if one of the variables associated with a and one of the variables associated with b is set to True, then the game is over because an initial propagation clause has been falsified, so the Delayer must try to avoid this. The Prover's goal will be to set either both of a_0 and a_1 to False, or to set both of b_0 and b_1 to False, allowing for the game to be played on one of the subgraphs rooted at a predecessor of c.

The Prover starts by querying a_0. If the Delayer says 'You Choose', then the Prover sets a_0 to True, so one of the variables associated with a has been set to True. (This means that the Delayer would not set a_0 to True, because it is better to let the Prover do so while giving up a point). If the Delayer sets a_0 to False, then the Prover immediately queries a_1. If the Delayer says 'You Choose', then the Prover sets a_1 to False, so both of a_0 and a_1 have been set to False, which was the Prover's goal, with at most 1 point being scored. The Delayer would not set a_1 to True, because this would not win any points, and winning 1 point having a_0 set to True would have been a better strategy. Similarly, the Delayer would not set a_1 to False at this point, because it would be better to let the Prover do this while winning a point. Therefore, at this point either both a_0 and a_1 have been set to False with 1 point being scored, and we have moved the game to the subgraph rooted at a which has a lower pebbling number than c, or a_0 has been set to True with 1 point scored.

In the latter case, the Prover's next move is to query b_0. If the Delayer says 'You Choose', then the Prover sets it to True, which means that the game is over, since a_0 and b_0 are both True and both of c's associated variables are False, thereby falsifying an initial propagation clause. The pebbling number of c is at least 3 and the Delayer won only 2 points, which means that the Prover has done even better than is stated in the Theorem, so the Delayer cannot say 'You Choose' on b_0. Similarly, the Delayer would not set b_0 to True, because that would be even worse. Therefore the Delayer has no choice but to set b_0 to False, at which point the Prover queries b_1, which must also be set to False for the same reasons.

Therefore the Prover can have either a_0 and a_1 set to False or b_0 and b_1 set to False, which moves the game into a subgraph of c with a lower pebbling number while only giving up 1 point.

Base Case: Assume that both a and b are source nodes and that both c_0 and c_1 have already been set to False. Therefore c has pebbling number 2 and both a and b have pebbling number 1. The Prover first queries a_0. If the Delayer says 'You Choose', then the Prover sets it to True. (This means that the Delayer would not set a_0 to True, because it is better to let the Prover do so while winning a point). If the Delayer sets a_0 to False, then the Prover immediately queries a_1. If the Delayer replies 'You Choose', then the Prover sets a_1 to False, and the game is over with only 1 point scored. The Delayer must therefore set a_1 to True. But this means that both a_0 and a_1 were set without winning any points, so setting a_0 to False was a mistake; saying 'You Choose' was the correct strategy.

Since a_0 has been set to True with 1 point scored, the Prover next queries b_0. If the Delayer says 'You Choose', then the Prover sets b_0 to True and the game is over. (This means that the Delayer would not set b_0 to True, because it is better to let the Prover do so while winning a point). If the Delayer sets b_0 to False, then the Prover queries b_1. At this point, the Delayer must say 'You Choose', and any answer the Prover gives will falsify an initial clause.

Therefore the Delayer wins at most 2 points in the Base Case.

In effect, if the Prover starts at the target node and follows this strategy of 'moving down' the graph, every time the pebbling number decreases, the Delayer wins at most one point. This will eventually lead to a node with pebbling number 2 (the base case), in which the Delayer wins at most 2 more points, showing that the Prover can limit the Delayer to at most $B\text{-}Peb(G)$ points, as required. ☐

4.3 Prover Strategy for the Lingas Circuits

In Lingas' 1978 paper [Lin78], he shows that the Black Pebbling Game on monotone circuits is $\mathcal{PSPACE}$-Complete by reducing from the 3-QBF problem with alternating quantifiers. He does this by providing a reduction which takes as input a 3-QBF formula F with alternating quantifiers and outputs a binary circuit C and integer k such that F is True if and only if C has a pebbling number of at most k.

The reduction proceeds by taking F and building a number of 'widgets'. The example of the result of this reduction from [Lin78] can be seen below in Figure 4.2. The construction includes one widget for each quantifier, one for each literal, one for each clause, and one pyramid graph which acts as a conjunction between the clauses. Each clause widget is incident on the widgets corresponding to the literals contained in the clause. The literal widgets are shown using shorthand notation; they are the pyramid graphs from [CS76], and a more detailed diagram can be found below in Figure 4.3. The numbers on each shorthand version indicates the pebbling number of the apex of the pyramid.

The pebbling number output by the reduction is $k = 2U + E + M$, where U is the number of universal quantifiers, E is the number of existential quantifiers, and M is the number of clauses in F. In the case of our example, $k = 2 \times 2 + 3 + 4 = 11$.

Intuitively, the reduction works by forcing the pebbling to go through each truth assignment which sets F to True. Specifically, for each of the universal quantifiers, the pebbling must travel up both of its sides, requiring that the entire circuit below that point be re-pebbled, thereby ensuring that F is True. If F is false, then the circuit requires $k+1$ pebbles and cannot be pebbled using only k.

We shall refer to the circuits corresponding to true QBF formulas as the 'Lingas Circuits'. Although the main result in [Lin78] is that pebbling monotone circuits in general is $\mathcal{PSPACE}$-Complete, the reduction specifically shows that the problem of pebbling the Lingas circuits is $\mathcal{PSPACE}$-Complete. Formally, we define the Lingas Circuits as follows:

Definition 4.5 (Lingas Circuits). $\mathcal{C}_{\mathcal{L}} = \{(C, k) \mid \exists$ *a true* 3-QBF *formula* F *with alternating quantifiers such that applying Lingas' reduction to* F *yields* $(C, k)\}$

We shall make use of the $\mathcal{PSPACE}$-Completeness of the Lingas Circuits to prove the $\mathcal{PSPACE}$-Completeness of both the Prover / Delayer Game as well as the Tree Clause Space Problem. To this end we must first prove that when playing the Peb^2 formula of a Lingas Circuit, the Prover has a strategy limiting the Delayer to at most B-$Peb(C)$ points (for more details on the Peb^2 formulas, please refer to Definition 2.10). The strategy is quite simple: the Prover starts at the target node, and forces the Delayer to admit that both variables associated with it must be False. Then the Prover proceeds to propagate this 'Double False' setting downwards. We shall show that it is possible to traverse each existential widget while only giving up at most one point, and that it is possible to traverse each universal widget while only giving up at most two points. Finally, we shall show that it is possible to traverse the conjunctive pyramid to derive the final contradictions while giving up at most M points, where M is the number of clauses in the formula underlying the circuit.

Each of these steps shall be proven by giving a decision tree showing the order in which the Prover queries variables. Each branch in this tree shall terminate with the Delayer scoring no more than B-$Peb(C)$ points. A contradiction can be obtained (thereby ending the game) in one of four ways:

1. By falsifying a target clause; for target node t, either t_0 is set to True or t_1 is set to True.
2. By falsifying a propagation clause associated with an AND gate; for some AND node c with predecessors a and b, both of c's variables are set to False, and one of a's variables as well as one of b's variables is set to True.
3. By falsifying a propagation clause associated with an OR gate; for some OR node c with predecessors a and b, both of c's variables are set to False, and one of a's variables or one of b's variables is set to True.
4. By falsifying a source clause; for some source node s, both of s's variables are set to False.

The Prover's strategy is somewhat interesting in that whenever the Delayer responds 'You Choose' to a query, the Prover shall always respond with 'True', and never with 'False'. Note that although technically the Delayer has the choice of three different responses after each variable is queried, our decision tree only needs to be binary. This is because if the Prover is willing to give the Delayer one point by responding to the Delayer's 'You Choose' answer, then there is no sense in exploring the path in which the Delayer gives the same answer without winning a point, since that path can only be better for the Prover.

Theorem 4.6. *For any binary circuit and integer pair* $(C, k) \in \mathcal{C}_{\mathcal{L}}$*, when playing on the formula* $Peb^2(C)$*, the Prover has a strategy limiting the Delayer to at most* $k = B$-$Peb(C)$ *(with sliding) points.*

Proof: The Prover's strategy proceeds as follows: the first query is on t_0, one of the variables associated with the target node t. If the Delayer says 'You Choose', then the Prover sets it to True, and the game is over with just one point scored. If the Delayer says 'True', then the game is over with no points scored, so the Delayer must set it to False. Next the Prover queries t_1, which is set to False for the same

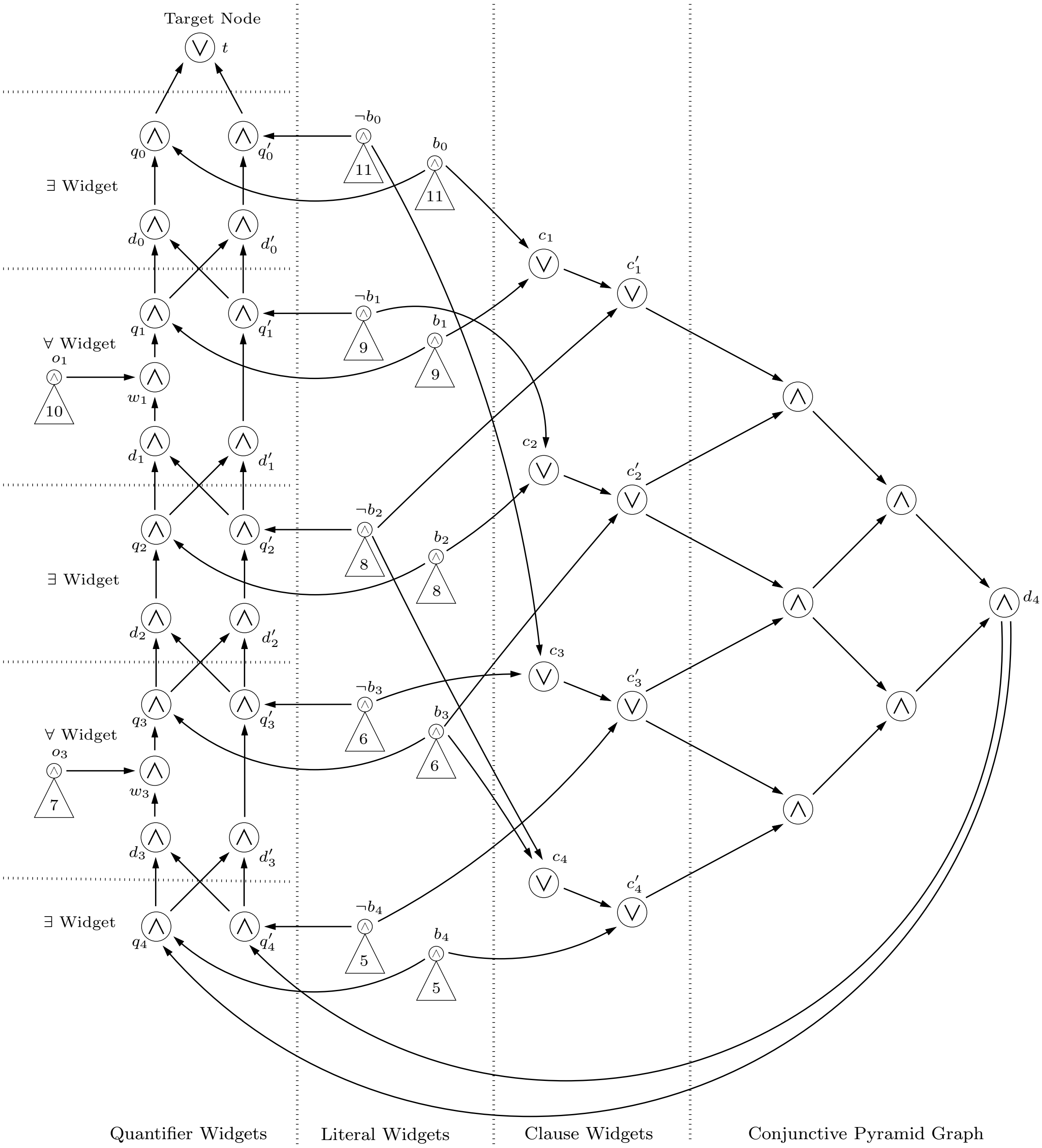

Figure 4.2: The Monotone Circuit Resulting from Applying Lingas' Reduction to the True 3-QBF Formula $F = \exists x_0 \forall x_1 \exists x_2 \forall x_3 \exists x_4\ (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_0 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$. In This Example $k = 11$.

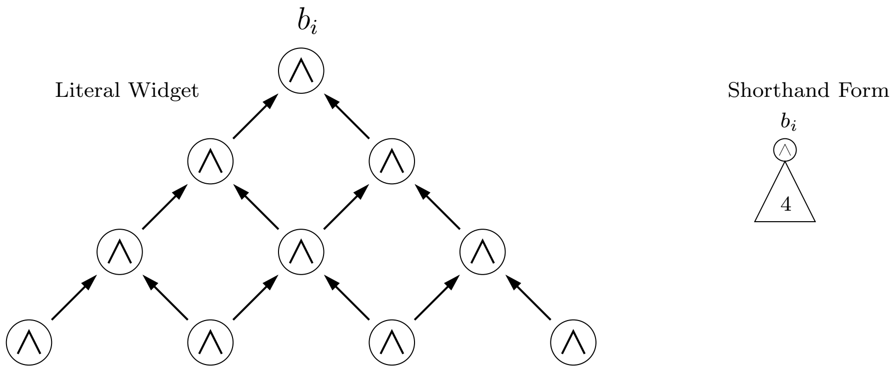

Figure 4.3: An Example of a Literal Widget from Lingas' Construction

reasons. The Delayer has therefore scored no points, so we enter the first quantifier widget with k points remaining.

Now the Prover inductively traverses the quantifier widgets in order, propagating the 'Double False' setting downwards towards the conjunctive pyramid. We will show that the Prover has a strategy which gives up at most one point while traversing an existential widget, and at most two points when traversing a universal widget.

We shall first show the Prover's strategy for traversing the existential widget, shown below in Figure 4.4. We assume that we have j points remaining before the Delayer reaches k points, and that both variables associated with the node d_{i-1} have been set to False or that both variables associated with the node d'_{i-1} have been set to False. The decision tree showing the Prover's strategy is shown below, beside the widget.

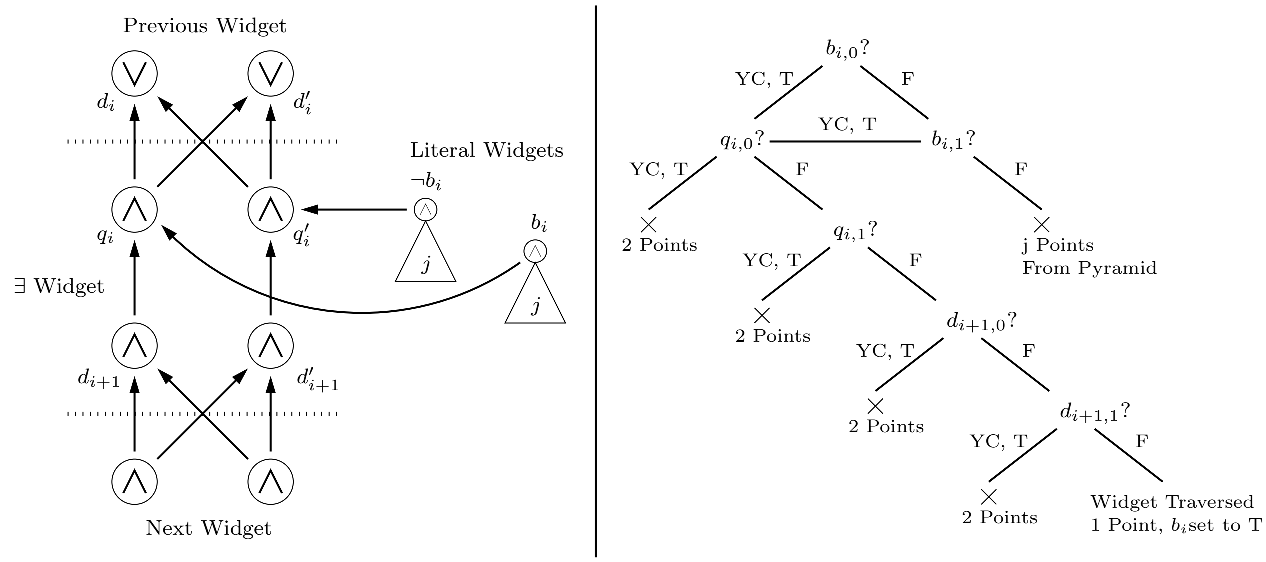

Figure 4.4: The Existential Widget from Lingas' Construction Together with the Decision Tree Showing the Prover's Strategy for Traversing it.

Note that edges are labelled with the different possibilities at each step during the Prover / Delayer game; 'YC, T' represents the case when the Delayer says 'You Choose', followed by the Prover setting the variable to True, and 'F' represents the case when the Delayer sets the variable to False. Duplicate subtrees are indicated by having multiple parents. In addition, leaves are labelled in the Prover / Delayer Game outcomes which they lead to. Leaves labelled with numbers represent situations in which the game is over since an initial clause has been falsified, and the number indicates how many points were scored. Although we said that the Prover may give up at most 1 point while traversing this widget, some leaves end with the Delayer scoring 2 points. This is not a problem because even if this is the final quantifier widget, the formula has at least one clause, so we have at least one extra point's leeway.

The leaf corresponding to both of b_i's variables being set to False corresponds to the game entering the pyramid graph associated with variable b_i which has pebbling number j. Since every pyramid graph belongs to the $\mathcal{G}_{\mathcal{I}}$ family, the Delayer can score at most j points on it by Lemma 4.4.

The remaining leaf is the one in which the widget has been traversed with only 1 point scored. Note that this decision tree corresponds to the strategy in which the Prover traversed the widget while setting one of the variables associated with b_i to True and both of the variables associated with d_{i+1} to False, thereby leading to the next widget. The case in which the Prover traverses the widget while setting one of the variables associated with $\neg b_i$ to True and both of the variables associated with d'_{i+1} to False is completely symmetrical. In other words, the Prover has complete control over which of the literal widgets is set to True during the traversal. This will be important because it allows the Prover to set the literal widgets in such a way so as to make the formula underlying the circuit true.

We now give the Prover's strategy for traversing the universal widget, shown below in Figure 4.5. We assume that we have j points remaining before the Delayer reaches k points, and that either both variables associated with the node d_i have been set to False or that both variables associated with the node d'_i have been set to False. The decision tree showing the Prover's strategy for the universal widget is shown below in Figure 4.6.

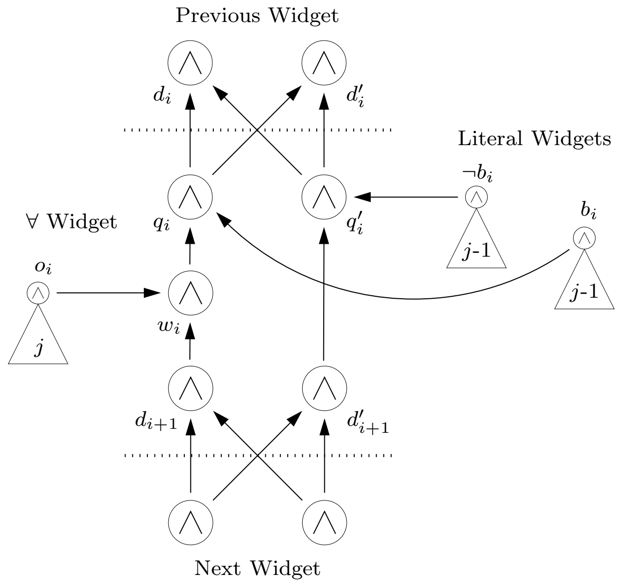

Figure 4.5: The Universal Widget from Lingas' Construction.

Just as with the existential widget, leaves in the decision tree are labelled with the number of points scored in falsifying an initial clause corresponding to that path. Once again, some paths lead to the Delayer scoring 3 points, but this is ok even if this is the final quantifier widget, since the formula

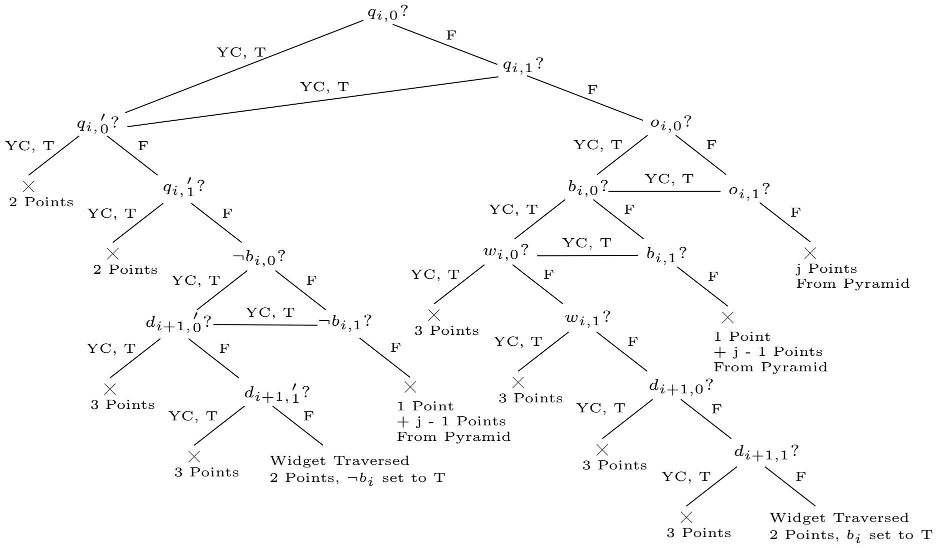

Figure 4.6: The Decision Tree Showing the Prover's Strategy for Traversing the Universal Widget.

underlying this circuit has at least one clause, thereby giving us leeway of at least one point. Also as before, the pyramid graphs corresponding to the literal widgets as well as the o_i widget belong to the $\mathcal{G}_{\mathcal{I}}$ family, so by Lemma 4.4, the Delayer can score at most $j-1$, $j-1$, and j points on them, respectively.

The remaining two leaves are ones in which the widget has been traversed with only 2 points scored. In one case, one of the variables associated with b_i is set to True and both variables associated with d_{i+1} are set to False, and in the other case one of the variables associated with $\neg b_i$ is set to True, and both of the variables associated with d'_{i+1} are set to False, thereby leading to the next widget.

We now show how the Prover traverses the conjunctive pyramid graph. The apex of the conjunctive pyramid is part of the final quantifier widget, so both of its variables have been set to False. Since the formula underlying the Lingas Circuit has M clauses and we have successfully traversed all of the clause widgets while giving up only the minimum number of points, we still have M points remaining. The conjunctive pyramid required to join M clauses has $M-1$ AND gates on its base, and therefore has pebbling number $M-1$.

Let us assume for a moment that the leaves of the conjunctive pyramid have no children. Since the pyramid belongs to the $\mathcal{G}_{\mathcal{I}}$ family, by Lemma 4.4 the Prover has a strategy limiting the Delayer to at most $M-1$ points. However, since each of the conjunctive pyramid's leaves does have two children, we must explore the decision tree rooted at the possibility that the two variables associated with a leaf l are both set to False. Since the Delayer did not say 'You Choose' on either of l's variables, the number of points scored in setting them both to False is at most $M-2$. This is the worst-case scenario branching down the conjunctive pyramid, so the number of points scored on all other paths to leaves is strictly less than $M-2$.

We therefore have at least 2 points left with which to derive a contradiction within the clause widgets. We shall show how to force a contradiction given a conjunction pyramid leaf l which has had both of its associated variables set to False. The Prover's strategy for doing this is shown below in Figure 4.7.

A key observation to make is that it is possible for the Prover to traverse the quantifier widgets such that the literal widgets which are attached to the existential widgets are set to True in any arbitrary way. Since the formula which the overall circuit is based on is a true QBF formula F, it is therefore possible to set them so that each clause widget is incident on at least one literal widget which has been set to

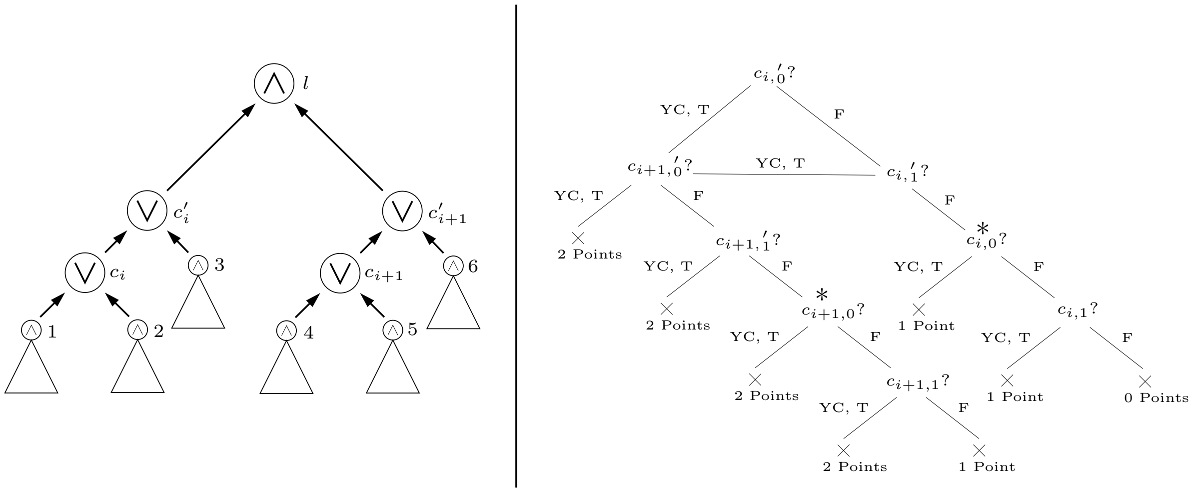

Figure 4.7: Two Clause Widgets Attached to a Leaf from the Conjunctive Pyramid in Lingas' Construction, Together with the Decision Tree Showing the Prover's Strategy for Finishing the Game.

True. This ensures that the Prover's strategy given by the decision tree in Figure 4.7 will always work. For example, it is possible for the Prover to traverse the existential widgets and set the truth values on the literal widgets in such a way that at least one of the variables associated with the literal widgets 1, 2, and 3 is True, and at least one of the variables associated with the literal widgets 4, 5, and 6 is True, thereby guaranteeing that the final contradictions needed can be obtained without exceeding 2 points. Note that the subtrees in the Prover's strategy labelled with $*$ may or may not be necessary, depending on which literal widgets were set to True.

Since each existential widget can be traversed while only giving up 1 point, each universal widget can be traversed while only giving up 2 points, the conjunctive pyramid can be traversed while only giving up $M - 2$ points, and then a final contradiction can always be derived within the clause widgets while giving up at most 2 points, it follows that all of the above decision trees can be combined to show that the Prover has a strategy limiting the Delayer to at most $B\text{-}Peb(C)$ points (defined with sliding), as required.

□

One interesting point of note is that the Prover / Delayer tree constructed above encodes a brute-force proof that F is a true QBF formula. This is particularly apparent when looking at the decision tree associated with the universal widget; it has two leaves at which we attach the decision tree associated with the next (existential) widget. In other words, the trivial decision tree proof system for the Prover / Delayer Game which we have been using seems to simulate the trivial brute force QBF proof system.

4.4 Delayer Strategy for All Binary Monotone Circuits

In this section we show that for any binary monotone circuit C, when playing the Prover / Delayer game on $Peb^2(C)$, the Delayer has a strategy which will always win at least $B\text{-}Peb(C)$ points. In [BSIW04] the authors give a Delayer strategy which wins at least $B\text{-}Peb(G) - 3$ points, where $B\text{-}Peb(G)$ is defined without sliding. We improve this argument to show that the Delayer has a strategy which is guaranteed to win at least $B\text{-}Peb(G)$ points, defined with sliding.

In the Delayer's strategy in [BSIW04], the Delayer maintains a DAG G, with certain nodes marked as source nodes, and other nodes marked as target nodes. In the improved strategy, which applies to the more general case of monotone circuits, the Delayer also maintains such a marked monotone circuit C, but in addition, certain source nodes have pebbles on them. Thus at each stage of the game, the Delayer maintains a structure of the form $\langle C, S, T, P \rangle$, where C is a monotone circuit, S and T are disjoint subsets of C, and $P \subseteq S$. By $\mathrm{Peb}(C, S, T, P)$, we mean the pebbling number of the marked, pebbled circuit $\langle C, S, T, P \rangle$, where the pebbles on the nodes in P can be used as 'free pebbles', and in addition we allow sliding, which seems to simplify the proof.

The Delayer's strategy proceeds as follows: initially, at the start of the game, P is empty (there are no free pebbles on the circuit), S is the set of source nodes of the circuit C, and T contains the unique target node of C. At the start of each round in the game, the Prover queries a variable $x(v)_i$, associated with a vertex v in the circuit C. The Delayer's strategy consists of two stages. In the first stage, the Delayer updates the sets S and T; in the second stage, the Delayer answers the Prover's query, and updates P.

The first stage proceeds as follows: assume that the marked, pebbled circuit at the start of the round is $\langle C, S, T, P \rangle$; in the first stage, the Delayer updates S and T to S' and T'.

Case 1a: If $v \in S \cup T$, then set $S' := S$, and $T' := T$.

Case 1b: If $v \notin S \cup T$, and $\mathrm{Peb}(C, S, T, P) = \mathrm{Peb}(C, S, T \cup \{v\}, P)$, then set $S' := S$ and $T' := T \cup \{v\}$.

Case 1c: If $v \notin S \cup T$, and $\mathrm{Peb}(C, S, T, P) > \mathrm{Peb}(C, S, T \cup \{v\}, P)$, then set $S' := S \cup \{v\}$ and $T' := T$.

The second stage of the Delayer's strategy proceeds as follows. In this stage, the Delayer updates P to P', and responds to the Prover's query.

Case 2a: If $v \in S'$, and v has no pebble on it, then respond $*$ ('you choose'), and place a pebble on v (that is to say, $P' := P \cup \{v\}$). If v has a pebble on it, then assign the queried variable the value 1.

Case 2b: If $v \in T'$, then assign the queried variable the value 0, and set $P' := P$.

Lemma 4.7. *When the game terminates, Peb$(C, S, T, P) = 0$.*

Proof: When the game terminates, an initial clause is falsified. Because of the strategy followed by the Delayer in the second stage of each round, this clause cannot be a clause associated with one of the initial source nodes of C, nor can it be the clause associated with the target node of C. Consequently, it must be a pebbling axiom associated with a vertex v and its immediate predecessors $u_1, \ldots, u_k$ of v is an $\wedge$ gate, or just one of its immediate predecessors if v is an $\vee$ gate. Let us assume that v is an $\wedge$ gate. Then both variables associated with v must be set to 0, from which it follows that $v \in T$, by Case 2a of the Delayer's strategy. If u_i is one of the immediate predecessors of v, one of the two variables associated with u_i must be set to 1. By Case 2b of the Delayer's strategy, u_i cannot be in T, so $u_i \in S$. It follows from this that there must be a pebble on u_i. Hence, we can pebble $\langle C, S, T, P \rangle$ by sliding the pebble on u_1 to v, showing that $\mathrm{Peb}(C, S, T, P) = 0$. The second case, where v is an $\vee$ gate, proceeds by essentially the same argument. □

Lemma 4.8. *If $\langle C, S, T, P \rangle$ is a marked, pebbled circuit, and v a vertex in C, then*

$$\mathit{Peb}(C, S, T, P) \leq \max\{\mathit{Peb}(C, S, T \cup \{v\}, P), \mathit{Peb}(C, S \cup \{v\}, T, P \cup \{v\}) + 1\}.$$

Proof: We employ the following strategy to pebble T from S, using only the free pebbles in P. First, pebble $T \cup \{v\}$ from S, using $\mathrm{Peb}(G, S, T \cup \{v\}, P)$ pebbles. If the result is a pebble in T, then we are through, otherwise the final configuration has a pebble on v. Keeping this pebble on v, remove the other pebbles, then pebble T from $S \cup \{v\}$; this final step uses a total of $\mathrm{Peb}(C, S \cup \{v\}, T, P \cup \{v\}) + 1$ pebbles. □

Theorem 4.9. *If at the beginning of a round in the game, the marked, pebbled circuit is $\langle C, S, T, P \rangle$, then the Delayer can score Peb(C, S, T, P) more points in the game.*

Proof: We argue by induction on the depth of the game tree. Lemma 4.7 settles the base case. Assume that the theorem holds for a subtree in which the marked, pebbled circuit is $\langle C, S', T', P' \rangle$; we wish to show that it holds also for its immediate supertree, associated with the marked, pebbled circuit $\langle C, S, T, P \rangle$. If $\mathrm{Peb}(C, S, T, P) = \mathrm{Peb}(C, S', T', P')$, then there is nothing to prove, so we can assume that $\mathrm{Peb}(C, S, T, P) > \mathrm{Peb}(C, S', T', P')$. By Lemma 4.8, $\mathrm{Peb}(C, S, T, P) = \mathrm{Peb}(C, S', T', P') + 1$.

Now consider the round of the game in which the Delayer's initial circuit is $\langle C, S, T, P \rangle$, and the final circuit is $\langle C, S', T', P' \rangle$, and the variable queried was $x(v)_i$. If v were in T', or if v had a pebble on it at the start of the round, then $\mathrm{Peb}(C, S, T, P) = \mathrm{Peb}(C, S', T', P')$. It follows from this that v is in S', but does not have a pebble on it at the start of the round, so the Delayer scores a point during this round. This proves that the condition of the Theorem holds for the supertree, as well. $\square$

Corollary 4.10. *For any binary monotone circuit C, when playing on the formula $Peb^2(C)$, the Delayer has a strategy which wins at least $B\text{-}Peb(C)$ points (with sliding).*

4.5 Black Pebbling, Prover/Delayer Game, & Tree Clause Space Equivalence

Since Corollary 4.10 applies to all binary monotone circuits, we can combine it with the upper bound on points from Lemma 4.4 to get an exact equivalence between black pebbling number and Prover / Delayer number for the $\mathcal{G}_{\mathcal{I}}$ DAGs. However, when we further include the equivalence between $TCS(F \vdash_{\textsf{T-RES}} \emptyset) - 1$ and $PD(F)$ from Theorem 3.5, we get the following three-way equivalence for the poly-time solvable pebbling graphs:

Corollary 4.11. *For any binary DAG $G \in \mathcal{G}_{\mathcal{I}}$, if the pebbling game is defined using sliding, then $PD(Peb^2(G)) = B\text{-}Peb(G) = TCS(Peb^2(G) \vdash_{\textsf{T-RES}} \emptyset) - 1$.*

More importantly, the Delayer's lower bound on points for binary monotone circuits also includes the Lingas Circuits, for which we already know that the Prover can limit the Delayer to at most $B\text{-}Peb(C)$ points because of the result in Lemma 4.6, thereby showing that for these circuits, black pebbling number and Prover / Delayer number are equivalent. When we further combine this with Theorem 3.5, we get the following three-way equivalence:

Corollary 4.12. *For any binary circuit $C \in \mathcal{C}_{\mathcal{L}}$, if the pebbling game is defined using sliding, then $PD(Peb^2(C)) = B\text{-}Peb(C) = TCS(Peb^2(C) \vdash_{\textsf{T-RES}} \emptyset) - 1$.*

This is interesting because the Lingas Circuits are a $\mathcal{PSPACE}$-Complete set. In the next section we shall make use of this fact to prove the $\mathcal{PSPACE}$-Completeness of both the Prover / Delayer Game as well as the Tree Clause Space problem.

4.6 $\mathcal{PSPACE}$-Completeness of Tree Clause Space & Prover / Delayer Game

This section contains an immediate corollary to the results in the previous section, namely the $\mathcal{PSPACE}$-Completeness of both the Tree Resolution Clause Space Problem as well as the Prover / Delayer game. The Tree Resolution Clause Space problem ($TCSP$) is defined as follows: given a formula F and integer k, does F have a T-RES refutation with clause space at most k? The Prover Delayer Game Problem ($PDGAME$) is defined as follows: given a formula F and integer k, is the Prover / Delayer number of F at most k?

More formally, the languages associated with these problems are defined as follows:

Definition 4.13 ($TCSP$)**.** $TCSP = \{(F, k) |$ *F is a formula for which there exists a* T-RES *refutation with clause space at most k*$\}$

Definition 4.14 ($PDGAME$)**.** $PDGAME = \{(F, k) |$ *F is a formula for which the Prover / Delayer Number is at most k*$\}$

Given Corollary 4.12 from the previous section, proving the $\mathcal{PSPACE}$-Completeness of $TCSP$ is a fairly straightforward reduction from the $\mathcal{PSPACE}$-Complete black pebbling problem of Lingas Circuits with sliding. However, before proving this result we must first give $\mathcal{PSPACE}$ algorithms for $TCSP$ and $PDGAME$.

4.6.1 $\mathcal{PSPACE}$ Algorithms for $TCSP$ and $PDGAME$

We first show that $TCSP \in \mathcal{PSPACE}$, which is not immediately obvious. In order to do this, we shall make use of the following Lemma:

Lemma 4.15. *Every unsatisfiable CNF formula F has a* T-RES *refutation with clause space at most $n+1$.*

Proof: Let F be any arbitrary unsatisfiable CNF formula. Simply take F and choose some ordering for its n variables. Build a complete DPLL tree for F, branching on this ordering. This tree can be viewed as a T-RES proof which can be pebbled with at most $n+1$ pebbles, since any binary tree can be pebbled with $h+1$ pebbles, where h is the height of the tree. These pebbles correspond to which clauses need to be kept in memory in order to verify the proof, showing that for any unsatisfiable F, $TCS(F \vdash_{\textsf{T-RES}} \emptyset) \leq n+1$, as required.

□

Lemma 4.16. $TCSP \in \mathcal{PSPACE}$

Proof: Given an input (F, k) we first determine if F is satisfiable. Since $SAT \in \mathcal{NP}$ and $\mathcal{NP} \subseteq \mathcal{PSPACE}$, this is not a problem. If F is satisfiable, then we reject. If it is unsatisfiable, then we look at k. If $k \geq n+1$, where n is the number of distinct variables in F, then by Lemma 4.15 we simply accept.

Otherwise F is unsatisfiable and $k \leq n$, so if F has a (configuration style) T-RES refutation π with $TCS(F \vdash_{\textsf{T-RES}} \emptyset) \leq k$, then each configuration contains at most k clauses. Since $k \leq n$ and since each clause contains at most n variables, each configuration in π requires only polynomial space. Although we don't have access to π, we verify it as follows: first we use a non-deterministic algorithm. Start with a configuration $\mathbb{C}_0 = \emptyset$. Guess configuration $\mathbb{C}_1$, check to ensure that it follows from $\mathbb{C}_0$ by a legal tree resolution step, and erase configuration $\mathbb{C}_0$. Next, guess configuration $\mathbb{C}_2$, check to make sure that it follows from $\mathbb{C}_1$, and erase configuration $\mathbb{C}_1$. Continue this way until $\mathbb{C}_k$ has been derived. Note that at any time, there are only two configurations in memory, but since each configuration takes only a polynomial amount of space, our computation is in $\mathcal{NPSPACE}$. Finally, we appeal to Savitch's Theorem [Sav70] to show that the problem of determining whether or not F has T-RES refutation π with $TCS(F \vdash_{\textsf{T-RES}} \emptyset) \leq k$ is in $\mathcal{PSPACE}$, thereby completing our $\mathcal{PSPACE}$ algorithm for TCS.

□

Since $TCSP \in \mathcal{PSPACE}$ and it is virtually identical to $PDGAME$, it is trivial to design a $\mathcal{PSPACE}$ algorithm for $PDGAME$:

Lemma 4.17. $PDGAME \in \mathcal{PSPACE}$

Proof: To show that $PDGAME \in \mathcal{PSPACE}$, simply take the input (F, k) and check if $TCS(F \vdash_{\textsf{T-RES}} \emptyset) - 1 \leq k$ using the $\mathcal{PSPACE}$ $TCSP$ algorithm from Lemma 4.16 as a subroutine and give the same answer. By Theorem 3.5, $TCS(F \vdash_{\textsf{T-RES}} \emptyset) = PD(F) + 1$, which immediately proves the correctness of this algorithm.

□

4.6.2 $\mathcal{PSPACE}$-Completeness of $TCSP$ and $PDGAME$

We are now ready to prove this section's main result:

Theorem 4.18. *$TCSP$ is $\mathcal{PSPACE}$-Complete.*

Proof: By Lemma 4.16, we know that $TCSP \in \mathcal{PSPACE}$. Next we show that $TCSP$ is $\mathcal{PSPACE}$-Hard by reducing from black pebbling with sliding on the Lingas Circuits. The reduction proceeds by simply taking the input (C, k), and outputting $(Peb^2(C), k-1)$. Clearly this is a polytime reduction, and its correctness is immediate from Corollary 4.12.

$\square$

Since by Theorem 3.5 we know that for any formula F, $PD(F) = TCS(F \vdash_{\text{T-RES}} \emptyset) - 1$, the $\mathcal{PSPACE}$-Completeness of $TCSP$ also gives us the $\mathcal{PSPACE}$-Completeness of $PDGAME$ as a corollary.

Corollary 4.19. *$PDGAME$ is $\mathcal{PSPACE}$-Complete.*

Proof: The proof is very similar to Theorem 4.18. By Lemma 4.17, we know that $PDGAME \in \mathcal{PSPACE}$. To show that $PDGAME$ is $\mathcal{PSPACE}$-Hard, simply reduce from $TCSP$: Given an input (F, k) for $TCSP$, output $(F, k+1)$. Clearly this is a polytime reduction, and its correctness is immediate from Theorem 3.5.

$\square$

5 The Complexity of Input Resolution Derivation Total Space

In this section we prove a number of results pertaining to black pebbling and the total space of a particularly simple form of Resolution called Input Resolution:

Definition 5.1. *An Input Resolution (*I-RES*) proof π is a* RES *proof in which at least one of the two inputs to every application of the resolution rule is an input clause. The first two clauses resolved on are called the 'Top Clauses' of π. The final result of the refutation, C_k is called the 'Goal Clause'.*

A slight variation of this is the proof system I-RES-W$^-$*, which is* I-RES *augmented with a restricted form of the weakening rule which allows for clauses to be weakened with negative literals. More specifically, if a clause C has been derived, then we may infer the clause $C \cup \{\neg x\}$ for any arbitrary variable x.*

In Section 5.1 we prove that for any DAG G, B-$Peb(G)$ is equivalent to the total space of I-RES-W$^-$ derivations of a certain variation of $Peb^1(G)$ formulas. Following that we show how to dispense with weakening and prove the equivalent result for I-RES. In Section 5.2 we put these equivalence results to use in order to prove the $\mathcal{PSPACE}$-Completeness of various forms of the I-RES Total Space Problem. Finally, in Section 5.3 we prove another corollary to one of the equivalence results, namely an optimal I-RES size / total space tradeoff.

5.1 The Equivalence of Black Pebbling & Input Total Space

We shall prove that for any DAG G, B-$Peb(G)$ (with sliding) is almost exactly equivalent to the minimum total space of any I-RES derivation from $Peb^1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$. These formulas, closely related to the $Peb^1(G)$ formulas described in Section 2.4, are defined as follows:

Definition 5.2 ($Peb^1(G)^*$). *In the case of binary DAGs without OR gates, in order to ensure that our resulting formula is a 3-CNF formula, we define $Peb^1(G)^*$ to be just like $Peb^1(G)$ except that we include dummy literals $\neg\alpha$ and $\neg\beta$ in each singleton clause so that each source clause $\{s\}$ becomes $\{s, \neg\alpha, \neg\beta\}$ and the target clause becomes $\{\neg t, \neg\alpha, \neg\beta\}$. Note that there are no positive instances of α or β, so a proof of $Peb^1(G) \vdash \emptyset$ will correspond exactly with a proof of $Peb^1(G)^* \vdash \{\neg\alpha, \neg\beta\}$.*

We shall first show that this equivalence holds for I-RES-W$^-$, and then we will show that this result holds for standard I-RES by demonstrating how to dispense with weakening.

In order to prove these results, we first need to define two very similar concepts. The first is a way of documenting the moves made during a pebbling game. We refer to such a record as a 'Pebbling History':

Definition 5.3 (Pebbling Histories & Strategies). *A pebbling history on a monotone circuit G with j moves is a sequence of pairs $H = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$ in which each (B_i, W_i) pair completely describes which nodes of G have pebbles on them at step i of the pebbling game; B_i is the set of nodes having black pebbles on them, and W_i is the set of nodes having white pebbles on them at time i. A step in the pebbling game is defined as being one move by the player: either the removal of a pebble, the placement of a pebble, or the sliding of a pebble (if sliding is allowed).*

When dealing with pure black or pure white pebbling histories, we allow $H = S_0, S_1, ..., S_j$ to be a sequence of sets in which each S_i is the set of nodes at time i having pebbles on them. This simplifies our notation, since pairs are not necessary in this case.

A 'Pebbling Strategy' is a history which has met the termination condition of the game.

The second concept which we need to define before proceeding is called an I-RES refutation 'Backbone', and it is very similar to the idea of a one-colour pebbling strategy:

Definition 5.4 (I-RES Backbone). *Let T be the tree underlying an I-RES refutation π of a formula F. The Backbone B of π is the linear portion of T starting at π's top clause C_0 and ending at the goal clause C_k, with all of the remaining clauses (which are all input clauses) removed from T. B can therefore be written as a sequence of clauses $B = B_0, B_1, ..., B_k$. Depending on our application we may consider B_0 to be the top clause and B_k to be the goal clause, or the other way around.*

5.1.1 The Equivalence of Black Pebbling & Input Total Space With Weakening

Our first lemma proves the forward direction of our equivalence and states that it is possible to take a pure black pebbling strategy of an arbitrary DAG G and from it build an $\mathsf{I\text{-}RES\text{-}W}^-$ derivation from $Peb^1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which the backbone perfectly encodes the strategy:

Lemma 5.5. *For any DAG G with target node t, if G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_{j-1}, S_j$ where $t \in S_j$, then $Peb^1(G)^*$ has an $\mathsf{I\text{-}RES\text{-}W}^-$ derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has a backbone $B = B_0, B_1, ..., B_{j-1}, B_j$ (where B_j is the top clause and B_0 is the goal clause) such that for all $0 \le i \le j$, $B_i = \bigcup_{v \in S_i}\{\neg v\} \cup \{\neg\alpha, \neg\beta\}$.*

Proof: Let G be any arbitrary DAG with target node t and let $S = S_0, S_1, S_2..., S_{j-1}, S_j$ be any pure black k-pebbling strategy (using sliding) of G. Note that $S_0 = \emptyset$ and $t \in S_j$. We will show that there is a corresponding $\mathsf{I\text{-}RES\text{-}W}^-$ derivation with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ by induction on the number of steps in the pebbling strategy S. It is useful to picture the pebbling strategy as a linear sequence of sets drawn with the first step at the bottom, and the last step at the top. We will construct the $\mathsf{I\text{-}RES\text{-}W}^-$ proof and backbone from the bottom (goal clause) to the top (top clause). Note that because the dummy literals $\neg\alpha$ and $\neg\beta$ are present in every clause of the proof backbone, each backbone clause has a width that is two greater than the corresponding step in the pebbling strategy.

Basis: Step 0 of the pebbling strategy contains no pebbles. Therefore $S_0 = \emptyset$. The corresponding clause in our backbone is the clause that we are trying to derive. Therefore $B_0 = \{\neg\alpha, \neg\beta\}$. Since $\bigcup_{v \in S_0}\{\neg v\} = \emptyset$, it is clear that $B_0 = \bigcup_{v \in S_0}\{\neg v\} \cup \{\neg\alpha, \neg\beta\}$.

Induction Hypothesis: Suppose that we have been able to translate our pebbling strategy up to and including step i to backbone clauses. More specifically, $B_i = \bigcup_{v \in S_i}\{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, holds for step S_i.

Induction Step: We now show how to translate step $i+1$ of our pebbling strategy into a corresponding backbone clause. More specifically, we need to show that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, holds for step $i+1$ in our pebbling strategy, S_{i+1}. Pebbling step S_{i+1} could have come from step S_i in one of exactly three ways:

1. By pebbling a source node s.
2. By removing a pebble from vertex u.
3. If nodes a and b are pebbled predecessors of c, by sliding one of the pebbles from a or b to c.

Case 1: In this case, $S_{i+1} = S_i \cup \{s\}$. Let $B_{i+1} = B_i \cup \{\neg s\}$. Then we can derive B_i from B_{i+1} by resolving B_{i+1} with the input clause $\{s, \neg\alpha, \neg\beta\}$, so this is a valid resolution step resolving on the variable s. By our induction hypothesis, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, but $B_{i+1} = B_i \cup \{\neg s\}$, so $B_{i+1} = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\} \cup \{\neg s\}$. Since $S_{i+1} = S_i \cup \{s\}$, we know that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, as required.

Case 2: In this case, $S_{i+1} = S_i - \{u\}$. Let $B_{i+1} = B_i - \{\neg u\}$. Then we can derive B_i from B_{i+1} by weakening B_{i+1} to introduce $\{\neg u\}$, so this is a valid resolution step. By our induction hypothesis, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, but $B_{i+1} = B_i - \{\neg u\}$, so $B_{i+1} = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\} - \{\neg u\}$. Since $S_{i+1} = S_i - \{u\}$, we know that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, as required.

Case 3: Without loss of generality, assume that we are sliding the pebble from a to c. In this case, $S_{i+1} = S_i - \{a\} \cup \{c\}$. Let $B_{i+1} = B_i - \{\neg a\} \cup \{\neg c\}$. Then we can derive B_i from B_{i+1} by resolving B_{i+1} with the input clause $\{\neg a, \neg b, c\}$ (resolving on the variable a), so this is a valid resolution step resolving on the variable c. By our induction hypothesis, $B_i = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, but $B_{i+1} = B_i - \{\neg a\} \cup \{\neg c\}$, so $B_{i+1} = \bigcup_{v \in S_i} \{\neg v\} \cup \{\neg\alpha, \neg\beta\} - \{\neg a\} \cup \{\neg c\}$. Since $S_{i+1} = S_i - \{a\} \cup \{c\}$, we know that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, as required.

Therefore, in all cases, there exists a valid next step in the backbone such that $B_{i+1} = \bigcup_{v \in S_{i+1}} \{\neg v\} \cup \{\neg\alpha, \neg\beta\}$, so by induction we can translate a pure black pebbling into an I-RES-W$^-$ proof in which the backbone perfectly encodes the pebbling strategy. In order to ensure that the I-RES-W$^-$ proof has top clause $\{\neg t, \neg\alpha, \neg\beta\}$, we may have to apply several weakenings in reverse. This is because the pebbling strategy may end with more than just the target node pebbled, in which case we would have to remove the literals corresponding to these superfluous pebbles in order to ensure that the top clause is indeed at the top of our I-RES-W$^-$ proof.

□

The next Lemma shows the opposite direction, namely that it is possible to take any arbitrary I-RES-W$^-$ derivation π and translate it into a pebbling strategy which uses at most as many pebbles as two less than π's backbone width:

Lemma 5.6. *For any DAG G with target node t, if $Peb^1(G)^*$ has an I-RES-W$^-$ derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has a backbone $B = B_0, B_1, ..., B_j$ (where B_0 is the top clause and B_j is the goal clause), then there exists a pure white pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_l$ where l is $O(j)$ with $S_0 = \{t\}$ and $S_l = \emptyset$ such that it is possible to translate each B_i into either one pebbling step S_q such that $S_q = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$, or to translate B_i into two consecutive pebbling steps S_{q_1}, S_{q_2} such that $|S_{q_1}| \leq |B_i - 2|$ and $S_{q_2} = \bigcup_{\neg v \in B_i} \{v\} - \{\neg\alpha, \neg\beta\}$.*

Proof: Let G be any arbitrary DAG with target node t and let π be an I-RES-W$^-$ derivation with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has a backbone of width $k+2$. We will show how to take this backbone and use it to create a pure white pebbling strategy which uses k pebbles. It is useful to picture π with the top clause at the top and the goal clause at the bottom. We shall move down the backbone of π and show how to build a corresponding pure white pebbling strategy. Note that every clause in the backbone contains the dummy literals $\neg\alpha$ and $\neg\beta$ and that this accounts for the '+ 2' in $k+2$.

Basis: Consider the top clause of π, $B_0 = \{\neg t, \neg\alpha, \neg\beta\}$; we create the equivalent first step of a pebbling by placing a white pebble on node t to create pebbling step $S_0 = \{t\}$. Therefore $S_0 = \bigcup_{\neg v \in B_0}\{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Induction Hypothesis: Suppose that we have been able to translate our backbone into a pebbling strategy up to and including backbone clause B_i and that B_i was translated to S_p such that $S_p = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\}$.

Induction Step: We will now show how to translate backbone clause B_{i+1} to create either one or two corresponding pebbling steps. Backbone clause B_{i+1} could have come from B_i in one of exactly three ways:

1. By resolving B_i with a source clause $\{u, \neg\alpha, \neg\beta\}$ (resolving on variable u).
2. By resolving B_i with a propagation clause $\{\neg a, \neg b, c\}$ (resolving on variable c).
3. By weakening B_i to introduce a negative literal $\neg u$.

Case 1: By our induction hypothesis, we translated backbone clause B_i into pebbling step S_p such that $S_p = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\}$. We now resolve B_i with the input clause $\{u, \neg\alpha, \neg\beta\}$ on variable u to create $B_{i+1} = B_i - \{\neg u\}$. Our corresponding pebbling move is to take S_p and create S_q by removing the white pebble on node u. Therefore $S_q = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\} - \{\neg u\}$, but $B_{i+1} = B_i - \{\neg u\}$, so $S_q = \bigcup_{\neg v \in B_{i+1}}\{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Case 2: By our induction hypothesis, we translated backbone clause B_i into pebbling step S_p such that $S_p = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\}$. We now resolve B_i with a propagation clause $\{\neg a, \neg b, c\}$ on variable c to create $B_{i+1} = B_i \cup \{\neg a, \neg b\} - \{\neg c\}$. We make two corresponding pebbling moves. The first is to place a white pebble on node a. Therefore $S_{p_1} = S_p \cup \{a\}$, so the size of our pebbling state has increased by 1. Similarly, $B_{i+1} = B_i \cup \{\neg a, \neg b\} - \{\neg c\}$, and neither a nor b were pebbled in S_p (which has corresponding backbone clause $S_p = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\}$ of size $|B_i| - 2$), which means that the variables a and b do not occur in B_i, so $|B_{i+1}| = |B_i| + 2$. Therefore $|S_{p_1}| \leq |B_{i+1}| - 2$.

The second and final pebbling move is to slide the white pebble on c to b. Therefore $S_{p_2} = S_p \cup \{a, b\} - \{c\}$. But $S_p = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\}$, so $S_{p_2} = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\} \cup \{a, b\} - \{c\}$, but $B_{i+1} = B_i \cup \{\neg a, \neg b\} - \{\neg c\}$, so $S_{p_2} = \bigcup_{\neg v \in B_{i+1}}\{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Case 3: By our induction hypothesis, we translated backbone clause B_i into pebbling step S_p such that $S_p = \bigcup_{\neg v \in B_i}\{v\} - \{\neg\alpha, \neg\beta\}$. But $B_{i+1} = B_i - \{\neg u\}$, so $S_q = \bigcup_{\neg v \in B_{i+1}}\{v\} - \{\neg\alpha, \neg\beta\}$, as required.

Therefore, in all three cases, there exists a valid next step in the pebbling such that $S_q = \bigcup_{\neg v \in B_{i+1}}\{v\} - \{\neg\alpha, \neg\beta\}$, so by induction we can translate an $\mathsf{I\text{-}RES\text{-}W}^-$ backbone B into a pure white pebbling strategy S such that S perfectly encodes B.

$\square$

The following Corollary combines the previous two Lemmas in order to prove an equivalence between the black pebbling number of DAG G and the $\mathsf{I\text{-}RES\text{-}W}^-$ total space of $Peb^1(G)^*$.

Corollary 5.7. *For any DAG G with target node t, G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2..., S_j$ with j steps where $t \in S_j$ and j is $O(l)$ if and only if $Peb^1(G)^*$ has an* $\mathsf{I\text{-}RES\text{-}W}^-$ *derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and the backbone of π contains l clauses where l is $O(j)$.*

Proof: Let G be any arbitrary DAG with target node t.

$\Rightarrow$ Suppose that G has a pure black k-pebbling history (using sliding) $S = S_0, S_1, S_2..., S_j$ with j steps where $t \in S_j$ and j is $O(q)$. By Lemma 5.5, it is possible to translate this pebbling strategy into an $\mathsf{I\text{-}RES\text{-}W}^-$ derivation π of $Peb^1(G)^*$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ such that the backbone of π contains j clauses, each of which has width at most $k+2$. However, since $Peb^1(G)^*$ is a 3-CNF formula, each of the input clauses used to resolve with clauses in the backbone has width 3. Since input refutations only require there to be two clauses in memory at any time, π therefore has total space $k+5$ and size j.

$\Leftarrow$ Suppose that $Peb^1(G)^*$ has an $\mathsf{I\text{-}RES\text{-}W}^-$ derivation π of $\{\neg\alpha, \neg\beta\}$ with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and π's backbone contains l clauses where l is $O(j)$. Since $Peb^1(G)^*$ is a 3-CNF formula, π has a backbone with width $k+2$. By Lemma 5.6, it is possible to translate this backbone into a pure white pebbling strategy containing l steps which uses at most k pebbles. But by Lemma 3.2, it is possible to convert this pure white pebbling strategy into a pure black pebbling strategy with $O(l)$ steps which uses exactly the same number of pebbles. Therefore G has a pure black k-pebbling strategy (using sliding) with $O(l)$ steps.

□

5.1.2 The Equivalence of Black Pebbling & Input Derivation Total Space

We now show that the weakenings in any $\mathsf{I\text{-}RES\text{-}W}^-$ proof π can be removed to turn it into a weakening-free proof π' with the same backbone width.

Lemma 5.8. *For any unsatisfiable set of Horn clauses F, any $C \in F$, and any goal clause D, there exists an* $\mathsf{I\text{-}RES\text{-}W}^-$ *proof π from F with top clause C and goal clause D such that the backbone width of π is at most k and the size of π is $O(s)$ if and only if there exists an* $\mathsf{I\text{-}RES}$ *proof π' from F with top clause C and goal clause $D^{'} \subseteq D$ such that the backbone width of π' is at most k and the size of π' is $O(s)$.*

Proof: Let F be any unsatisfiable set of Horn clauses, let C be any arbitrary initial clause in F, and let D be any arbitrary goal clause.

$\Rightarrow$ Suppose that there exists an $\mathsf{I\text{-}RES\text{-}W}^-$ proof π from F with top clause C and goal clause D such that the backbone width of $\pi \leq k$ and the size of π is $O(s)$. In order to show that there exists an $\mathsf{I\text{-}RES}$ proof π' from F with top clause C and goal clause $D^{'} \subseteq D$ such that the backbone width of $\pi' \leq k$ and the size of π' is $O(s)$, we will show how to translate π into π' without losing any of these properties. We shall first translate π into π^* by removing all of the weakenings in π as well as all applications of the Resolution rule which no longer apply due to the fact that weakened variables (and all other variables derived by Resolving on them) are now missing. It is important to note that π^* is not necessarily a legitimate $\mathsf{I\text{-}RES}$ proof, because our translation method may cause two successive clauses in the backbone of π to be translated into successive duplicate clauses in the backbone of π^*.

We translate π into π^* via induction on the depth (the top clause having the lowest depth) of π according to the following inductive procedure / proof. We shall translate each clause C_i in the backbone of π into a clause C_i^* in the backbone of π^* such that $\forall i\ C_i \subseteq C_i^*$.

Basis: The top clause in π is C. Set the top clause in C_0^* π^* to be C as well. Therefore $C_0 \subseteq C_0^*$, as required.

Induction Hypothesis: Suppose that $C_i \subseteq C_i^*$.

Induction Step: We now show how to translate π into π^* such that $C_{i+1} \subseteq C_{i+1}^*$:

Case 1: Suppose that C_{i+1} came from C_i in π via weakening. In this case, simply omit this weakening step in π^*. Since $C_i \subseteq C_i^*$ by our induction hypothesis, and C_{i+1} has grown, whereas C_{i+1}^* has not, we can conclude that $C_{i+1} \subseteq C_{i+1}^*$.

Case 2a: Suppose that C_{i+1} came from C_i in π by resolving C_i with the input clause I on variable x, but this resolution step is not possible in π^* because $x \notin C_i^*$. In this case, simply omit this resolution step in π^*. We know by the induction hypothesis that $C_i \subseteq C_i^*$, and since $x \notin C_i^*$, we can conclude that $C_{i+1} \subseteq C_{i+1}^*$.

Case 2b: Suppose that C_{i+1} came from C_i in π by resolving C_i with the input clause I on variable x, and $x \in C_i^*$. In this case, simply perform the same resolution step in π^*; i.e. resolve C_i^* with I to get C_{i+1}^*. By the induction hypothesis that $C_i \subseteq C_i^*$, so $C_{i+1} \subseteq C_{i+1}^*$.

Therefore, in all cases, $C_{i+1} \subseteq C_{i+1}^*$, showing that by induction, $D^* \subseteq D$, where D is the goal clause of π and D^* is the goal clause of π^*. Finally, since π^* might contain duplicate clauses along its backbone, we simply remove all but one out of each group of duplicates to create π', a legitimate I-RES proof. Therefore there exists an I-RES proof π' from F with top clause C and goal clause $D^{'} \subseteq D$ such that the backbone width of $\pi' \leq k$ and the size of π' is $O(s)$, as required.

$\Leftarrow$ Suppose there exists an I-RES proof π' from F with top clause C and goal clause $D^{'} \subseteq D$ such that the backbone width of $\pi' \leq k$ and the size of π' is $O(s)$. Since every I-RES proof is an I-RES-W$^-$ proof, π' is also an I-RES-W$^-$ proof with goal clause $D^{'} \subseteq D$. We convert π' to π by weakening $D^{'}$ to get D. Therefore there exists an I-RES-W$^-$ proof π from F with top clause C and goal clause D such that the backbone width of $\pi \leq k$ and the size of π is $O(s)$, as required.

□

The following Corollary is identical to Corollary 5.7 except that instead of proving an equivalence between the black pebbling number of a DAG G and the I-RES-W$^-$ total space of $Peb^1(G)^*$, here we use our weakening removal Lemma from above to show that this equivalence also holds for I-RES without weakening.

Corollary 5.9. *For any DAG G with target node t, G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2 ..., S_j$ with j steps where $t \in S_j$ and j is $O(l)$ if and only if $Peb^1(G)^*$ has an* I-RES *derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π' has total space $k+5$ and size l where l is $O(j)$.*

Proof: Let G be any arbitrary DAG with target node t.

$\Rightarrow$ Suppose that G has a pure black k-pebbling strategy (using sliding) $S = S_0, S_1, S_2 ..., S_j$ with j steps where $t \in S_j$ and j is $O(l)$. By Corollary 5.7, $Peb^1(G)^*$ has an I-RES-W$^-$ derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and size l where l is $O(j)$. Because $Peb^1(G)^*$ is a 3-CNF formula, π's backbone has width $\leq k+2$. By Lemma 5.8 there exists an I-RES proof π' from F with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $D^{'} \subseteq \{\neg\alpha, \neg\beta\}$ such that the backbone width of $\pi' \leq k+2$ and the size of π' is $O(j)$. Since F does not include any positive instances of the variables α or β, we can conclude that $D^{'} = \{\neg\alpha, \neg\beta\}$. Therefore, since $Peb^1(G)^*$ is a 3-CNF formula, π' has total space $k+5$ and size $O(j)$, with the same top and goal clauses as π.

$\Leftarrow$ Suppose that $Peb^1(G)^*$ has an I-RES derivation π' with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π' has total space $k+5$ and size l where l is $O(j)$. But every I-RES proof is an I-RES-W$^-$ proof, so there exists an I-RES-W$^-$ derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ in which π has total space $k+5$ and size l, which is $O(l)$. By Corollary 5.7, G has a pure black k-pebbling strategy.

□

5.2 The $\mathcal{PSPACE}$-Completeness of Input Derivation Total Space

This section contains some immediate corollaries to the previous results. Here we prove the $\mathcal{PSPACE}$-Completeness of three slightly different versions of the I-RES derivation total space problem. These results are somewhat surprising because I-RES is usually considered to be a trivial type of Resolution. This suggests that a new-found respect for Input Resolution is in order.

The I-RES Derivation Total Space Problem (ITS) is defined as follows: the input instance is (F, $C \in F$, D, k), where F is an unsatisfiable set of clauses, C is any clause in F, D is a goal clause, and k is an integer. The problem is to determine if there exists an I-RES derivation from F with top clause C, goal clause D, and total space at most k.

We shall prove the $\mathcal{PSPACE}$-Completeness of three different versions of this problem:

1. The set F is restricted to containing only Horn clauses, and instead of asking whether there exists an I-RES derivation from F with top clause C, goal clause D, and total space at most k, we ask whether there exists an I-RES-W$^-$ derivation with negative weakening from F with top clause C, goal clause D, and total space at most k. This is the input derivation with negative weakening total space problem, and we shall refer to the language associated with it as $HWITS$, where the 'HW' stands for '**H**orn with Negative **W**eakening'.
2. This version of the problem is almost identical to the previous one; the set F is still restricted to containing only Horn clauses, but this time we are dealing with I-RES rather than I-RES-W$^-$. We shall refer to the language associated with this version of the problem as $HITS$.
3. Again, this version of the problem is almost identical to the previous one, only this time there is no restriction on F. We shall refer to the language associated with this version of the problem as ITS.

5.2.1 $\mathcal{PSPACE}$-Completeness of the Weakening Input Total Space Problem

The $\mathcal{PSPACE}$-Completeness of $HWITS$ follows from the equivalence between the black pebbling number of DAG G and the I-RES-W$^-$ total space of $Peb^1(G)^*$ given in Corollary 5.7 from the previous section.

Theorem 5.10. *$HWITS$ is $\mathcal{PSPACE}$-Complete.*

Proof: We first show that $HWITS \in \mathcal{PSPACE}$. We are given an input instance (F, $C \in F$, D, k) and asked to determine if F has an I-RES-W$^-$ proof π with top clause C and goal clause D such that π's total space is bounded above by k. Since we are dealing with input refutations, we know that if such a π exists, then each of its configurations contains no more than two clauses. Since each clause contains at most n literals, it is clear that every configuration in π takes only polynomial space.

Our algorithm proceeds as follows: start with a configuration $\mathbb{C}_0 = \{C\}$. Guess configuration $\mathbb{C}_1$, check to ensure that it follows from $\mathbb{C}_0$ by a legal input resolution or negative weakening step, and erase configuration $\mathbb{C}_0$. Next, guess configuration $\mathbb{C}_2$, check to make sure that it follows from $\mathbb{C}_1$, and erase configuration $\mathbb{C}_1$. Continue this way until the goal clause has been derived. Note that at any time, there are only two configurations in memory. But since we are dealing with input refutations, we know that each configuration contains no more than two clauses. Since each clause contains at most n literals, it is clear that our non-deterministic algorithm requires polynomial space. This shows that $HWITS \in \mathcal{NPSPACE}$. Finally, we appeal to Savitch's Theorem [Sav70] to show that $HWITS \in \mathcal{PSPACE}$.

Next we show that $HWITS$ is $\mathcal{PSPACE}$-Hard by giving a polynomial-time reduction from the black pebbling number Problem with sliding ($BLACK$-PEB) from [GLT80] which was shown to be $\mathcal{PSPACE}$-Complete. We are given an instance (G, k) where t is the target node in G and we wish to convert it to an instance $(F,\ C \in F,\ D,\ k)$ such that $(G, k) \in BLACK$-PEB if and only if $(F,\ C \in F,\ D,\ k) \in HWITS$. Our reduction proceeds as follows: take (G, k) and output $(Peb^1(G)^*,\ \{\neg t, \neg\alpha, \neg\beta\},\ \{\neg\alpha, \neg\beta\},\ k+5)$, which is clearly a polynomial-time reduction.

The proof of correctness for this reduction is given by Corollary 5.7.

□

5.2.2 $\mathcal{PSPACE}$-Completeness of the Horn Input Total Space Problem

The $\mathcal{PSPACE}$-Completeness of $HITS$ follows immediately from the equivalence between the black pebbling number of DAG G and the I-RES-W$^-$ total space of $Peb^1(G)^*$ given in Corollary 5.9 from the previous section.

Theorem 5.11. *$HITS$ is $\mathcal{PSPACE}$-Complete.*

Proof: To show that $HITS \in \mathcal{PSPACE}$, we simply use the same algorithm given in Theorem 5.10.

To show that $HITS$ is $\mathcal{PSPACE}$-Hard, we give a polytime reduction from $BLACK$-PEB. As in Theorem 5.10, take (G, k) and output $(Peb^1(G)^*,\ \{\neg t, \neg\alpha, \neg\beta\},\ \{\neg\alpha, \neg\beta\},\ k+5)$, which is clearly a polynomial-time reduction.

The proof of correctness for this reduction is given by Corollary 5.9.

□

5.2.3 $\mathcal{PSPACE}$-Completeness of the Input Total Space Problem

The $\mathcal{PSPACE}$-Completeness of ITS follows trivially from the $\mathcal{PSPACE}$-Completeness of $HITS$:

Theorem 5.12. *ITS is $\mathcal{PSPACE}$-Complete.*

Proof: To show that $ITS \in \mathcal{PSPACE}$, we simply use the same algorithm given in Theorem 5.10.

To show that ITS is $\mathcal{PSPACE}$-Hard, we reduce from $HITS$. Simply take the input for $HITS$ and check to see if F is a set of Horn clauses. If not, output a pre-chosen string which is not in ITS. If so, then do nothing. Clearly this is a polytime reduction, and its correctness is immediate since Horn formulas are just a special case of what an ITS solver can decide.

□

5.2.4 $\mathcal{PSPACE}$-Completeness of the Input Derivation Width Problem

One interesting point of note is that I-RES Total Space and Width are virtually identical. More specifically,

Corollary 5.13. *For every k-CNF formula, $w(F \vdash_{\text{I-RES}} \emptyset) = TS(F \vdash_{\text{I-RES}} \emptyset) - k$.*

The I-RES Derivation Width Problem is therefore also $\mathcal{PSPACE}$-Complete, because the $Peb^1(G)^*$ formulas are in 3-CNF.

5.3 Optimal Size / Total Space Tradeoffs For Input Resolution

In this section we describe another corollary to the results in Section 5.1. This result also relies closely on one of the most surprising facts concerning pebbling, namely that there exist infinite families of DAGs which take an exponential amount of time to be pebbled, but if one is willing to use just one or two more pebbles, then they can be pebbled in linear time. The earliest known example of this phenomenon is [Lin78] in which the author gives an infinite family of DAGs such that pebbling any of them with the minimum number of pebbles requires $2^{\Omega(n^{1/3})}$ time. However, if given only two more pebbles, the amount of time required drops exponentially to only $O(n)$, giving a massive pebble / pebbling time tradeoff.

This result was later improved by Gilbert, Lengauer, and Tarjan:

Lemma 5.14 ([GLT80]). *There exists an infinite family of DAGs, call it $\mathcal{G}$, such that pebbling any $G \in \mathcal{G}$ with the minimum number k of pebbles takes $\Omega(2^n)$ time, but if only one more pebble is used, then the amount of time required to pebble G drops exponentially to only $O(n)$, where n is the number of vertices in G.*

Such an exponential separation at the cost of only one pebble is extraordinary, but since Corollary 5.9 gives such an exact relationship between black pebbling and total space for I-RES derivations, it is possible to translate this amazing result from the world of pebbling over to Resolution, thereby giving a massive size / total space tradeoff for I-RES:

Corollary 5.15. *There exists an infinite family of formulas $\mathcal{F} = \{Peb^1(G)^* | G \in \mathcal{G}\}$ such that for every $F \in \mathcal{F}$, every I-RES derivation π of F with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ where π has the minimum required total space, has size $\Omega(2^n)$. However, if only one more unit of total space is permitted, then the size of π drops to only $O(n)$, where n is the number of variables in F.*

Proof: Let $\mathcal{G}$ be the family of DAGs from Lemma 5.14, and let F be any arbitrary formula from $\mathcal{F}$. Therefore $F = Peb^1(G)^*$ for some $G \in \mathcal{G}$. G can be pebbled with k but not with fewer than k pebbles. By Corollary 5.9, F has an I-RES derivation π with top clause $\{\neg t, \neg\alpha, \neg\beta\}$ and goal clause $\{\neg\alpha, \neg\beta\}$ such that π requires total space $k+5$, but does not have an I-RES derivation with total space less than $k+5$. We know that to pebble G with k pebbles requires $\Omega(2^n)$ time, so by Corollary 5.9, π requires size $\Omega(2^n)$. Similarly, we know that pebbling G with $k+1$ pebbles requires $O(n)$ time, so again by Corollary 5.9, there exists a proof π' with total space $k+5+1 = k+6$ and size $O(n)$, as required.

$\square$

Therefore, using just one extra unit of total space yields an exponential decrease in size, giving an optimal size / total space tradeoff.

6 The Complexity of Resolution Variable Space

In this section we prove results pertaining to black-white pebbling and Resolution variable space. In Section 6.1 we show that for any monotone circuit C, the minimum RES variable space of refuting $Peb^1(C)$ is exactly equal to $BW\text{-}Peb(C)$. Next, in Section 6.2 we put this equivalence result to use in order to prove the $\mathcal{PSPACE}$-Hardness of the RES Variable Space Problem.

6.1 The Equivalence of Black-White Pebbling & Resolution Variable Space

In this section we prove that for any monotone circuit C, the minimum RES variable space of refuting $Peb^1(C)$ is exactly equal to $BW\text{-}Peb(C)$ (without sliding). We do this by first showing that this result holds for RES-EM, a slight variation of RES which allows the use of some trivial axioms. We then show how to dispense with these axioms to prove that the result also holds for RES.

Before proving our equivalence theorem, we first need to define the notion of 'dependency set'. Intuitively, the dependency set of a vertex i is the set of all white pebbles that were needed in order to pebble i. More formally dependency sets are defined as follows:

Definition 6.1 (Dependency Set). *Given a pebbling strategy $S = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$ on a monotone circuit C, for each $0 \leq k \leq j$ and each $i \in B_k \cup W_k$ we define the Dependency Set of a vertex i, denoted $\Delta(i)$, below. If a vertex i has no pebble on it, then $\Delta(i)$ is undefined.*

1. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a black pebble was placed on AND gate i of degree d, with immediate predecessors $p_1, ..., p_d$, then set $\Delta(i) = \bigcup_{1 \leq w \leq d} \Delta(p_w)$.*
2. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a black pebble was placed on OR gate i, then set $\Delta(i) = \Delta(p_w)$ for some pebbled predecessor p_w of i.*
3. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a white pebble was placed on node i where i is either an AND gate or an OR gate, then set $\Delta(i) = \{i\}$.*
4. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a black pebble was removed from node i where i is either an AND gate or an OR gate, then $\Delta(i)$ becomes undefined, and there is no change in the other Dependency Sets.*
5. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a white pebble was removed from AND gate i of degree d, with immediate predecessors $p_1, ..., p_d$, then for all v such that $i \in \Delta(v)$, $\Delta(v) = (\Delta(v) - \{i\}) \bigcup_{1 \leq w \leq d} \Delta(p_w)$.*
6. *If the difference between (B_k, W_k) and (B_{k+1}, W_{k+1}) is that a white pebble was removed from OR gate i then for all v such that $i \in \Delta(v)$, $\Delta(v) = (\Delta(v) - \{i\}) \cup \Delta(p_w)$ for some pebbled predecessor p_w of i.*

Another definition that we need before proving the equivalence of variable space and black-white pebbling number is that of a special form of Resolution:

Definition 6.2 (RES-EM). *The proof system* RES-EM *is just like* RES *except that all n clauses representing the law of excluded middle of the form $(a \vee \neg a)$ are present as axioms and may be used at any time as if they were initial clauses.*

We now prove a Lemma giving the first half of our variable space / black-white pebbling number equivalence. The high-level idea of this proof is as follows: for a monotone circuit C, we take a black-white pebbling strategy $S = (B_0, W_0), (B_1, W_1), ..., (B_j, W_j)$, and for each (B_k, W_k), we create a RES-EM configuration M_k. We then build a RES refutation from $Peb^1(C)$ by induction on k such that $VS(M_k) \leq |B_k \cup W_k|$. This is done by showing how to derive each M_{k+1} from M_k using variable space bounded by $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. Since $(B_0, W_0) = (\emptyset, \emptyset)$, and since $(B_j, W_j) = (\{t\}, \emptyset)$, the resulting proof is a RES-EM refutation of $Peb^1(C)$ bounded by variable space $BW\text{-}Peb(C)$.

This brings us to the final definition that we shall need before proceeding, that of a special configuration called M_k:

Definition 6.3. $M_k = \{(\Delta(i) \supset i) \mid i$ *is a pebbled vertex in pebbling step* $(B_k, W_k)\}$

Lemma 6.4. *For any monotone circuit C with target t, $VS(Peb^1(C) \vdash_{\text{RES-EM}} \emptyset) \leq BW\text{-}Peb(C)$, where $BW\text{-}Peb(C)$ is defined without sliding.*

Proof: Let C be any arbitrary monotone circuit with target t, and let $S = (B_0, W_0), ..., (B_j, W_j)$ be any pebbling strategy for C with $(B_0, W_0) = (\emptyset, \emptyset)$, and $(B_j, W_j) = (\{t\}, \emptyset)$. By induction on k, we show how to take each pebbling step (B_k, W_k) and build a RES refutation from $Peb^1(C)$ which includes the configurations $M_0, M_1, ..., M_j$ such that for all k, we can derive M_{k+1} from M_k within variable space $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. Of course, we write $(\Delta(i) \supset i)$ for notational purposes only; all clauses are in CNF.

Basis: $(B_0, W_0) = (\emptyset, \emptyset)$, so $M_0 = \emptyset$. Clearly the empty configuration can be derived from $Peb^1(C)$ via RES-EM within variable space $0 = |B_0 \cup W_0|$.

Induction Hypothesis: Suppose that for (B_k, W_k), $VS(M_k) \leq |B_k \cup W_k|$.

Induction Step: We now show that for (B_{k+1}, W_{k+1}), we can derive M_{k+1} from M_k all within variable space $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. We know that (B_{k+1}, W_{k+1}) could have come from (B_k, W_k) by placing or removing a single pebble. There are 6 cases to consider:

1. By placing a black pebble on AND gate i.
2. By placing a black pebble on OR gate i.
3. By placing a white pebble on node i, where i is either an AND or an OR gate.
4. By removing a black pebble from node i, where i is either an AND or an OR gate.
5. By removing a white pebble from AND gate i.
6. By removing a white pebble from OR gate i.

Case 1: Since we placed a black pebble on AND gate i, $(B_{k+1}, W_{k+1}) = (B_k \cup \{i\}, W_k)$, which means that $M_{k+1} = M_k \cup \{(\Delta(i) \supset i)\}$, where $\Delta(i) = \bigcup_{1 \leq w \leq d} \Delta(p_w)$. By our induction hypothesis, we know that for (B_k, W_k), $VS(M_k) \leq |B_k \cup W_k|$, so we need to show how to derive M_{k+1} from M_k within variable space $max\{|B_k \cup W_k|, |B_{k+1} \cup W_{k+1}|\}$. This is done as follows: we know that node i has predecessors $p_1, ..., p_d$, all of which are pebbled. Therefore, $(\Delta(p_w) \supset p_w) \in M_k$ for every predecessor p_w of i. We must show how to derive $(\Delta(i) \supset i)$. Since i has $p_1, ..., p_d$ as predecessors, $Peb^1(C)$ contains the clause $(p_1, \wedge, ..., \wedge\ p_d \supset i)$ as an initial clause. Download this clause into memory. Since the variables $p_1, ..., p_d$ are already all present in M_k, this adds at most one to our total variable space. Next, resolve $(p_1, \wedge, ..., \wedge\ p_d \supset i)$ with $(\Delta(p_1) \supset p_1)$ on variable p_1 to get $(\Delta(p_1), \wedge, ..., \wedge\ p_d \supset i)$. Repeat this for every $(\Delta(p_w) \supset p_w)$ to get $\{\bigcup_{1 \leq w \leq d} \Delta(p_w) \supset i\}$. But since $\Delta(i) = \bigcup_{1 \leq w \leq d} \Delta(p_w)$, we have derived $(\Delta(i) \supset i)$, all within variable space $|B_{k+1} \cup W_{k+1}|$.

Case 2: This is a simpler form of the previous case. Since we placed a black pebble on OR gate i, $(B_{k+1}, W_{k+1}) = (B_k \cup \{i\}, W_k)$, which means that $M_{k+1} = M_k \cup \{(\Delta(i) \supset i)\}$, where $\Delta(i) = \Delta(p_w)$ for some pebbled predecessor p_w. Therefore, $(\Delta(p_w) \supset p_w) \in M_k$, and $Peb^1(C)$ contains the initial clause $(p_w \supset i)$. Download this clause into memory, thereby adding at most one to the total variable space. Now simply resolve $(\Delta(p_w) \supset p_w)$ with $(p_w \supset i)$ to get $(\Delta(p_w) \supset p_w)$. But $\Delta(i) = \Delta(p_w)$, so we have derived $(\Delta(i) \supset i)$, all within variable space $|B_{k+1} \cup W_{k+1}|$.

Case 3: Since we placed a white pebble on node i which is either an AND gate or an OR gate, $(B_{k+1}, W_{k+1}) = (B_k, W_k \cup \{i\})$, which means that $M_{k+1} = M_k \cup \{i \supset i\}$. By our induction hypothesis, we know that for (B_k, W_k), $VS(M_k) \leq |B_k \cup W_k|$, so we need to show how to derive $(i \supset i)$. Since we are using the system RES-EM, this is an axiom, so simply download it. Clearly this does not exceed variable space $|B_{k+1} \cup W_{k+1}|$, since W_{k+1} has a pebble on i.

Case 4: Since we removed a black pebble from node i which is either an AND or an OR gate, $(B_{k+1}, W_{k+1}) = (B_k - \{i\}, W_k)$, which means that $M_{k+1} = M_k - \{(\Delta(i) \supset i)\}$. Therefore, in order to derive M_{k+1} from M_k, simply drop the clause $(\Delta(i) \supset i)$, which can clearly be done without increasing the variable space.

Case 5: This case is more complicated than the others, because i may be in the dependency sets of many other vertices. Since we removed a white pebble from vertex i, $(B_{k+1}, W_{k+1}) = (B_k, W_k - \{i\})$. In order to derive M_{k+1} from M_k, we must therefore drop the clause $(\Delta(i) \supset i)$ from M_k, and also show

how to remove the literal $\neg i$ from all other clauses that contain it. Because removing a white pebble requires all predecessors to be pebbled, there are two sub-cases to consider:

Case 5a: Suppose that i has no predecessors; i.e. i is a leaf. In this case we simply drop the clause $(i \supset i)$ from M_k, and resolve every remaining clause in M_k which contains $\neg i$ as part of its dependency set with the initial clause (i). Clearly this removes the literal $\neg i$ from all clauses containing it, and does not increase the variable space, since i is already present in M_k.

Case 5b: Suppose that i has in-degree d and has predecessors $p_1, ..., p_d$, of which 0 or more have black pebbles on them in (B_k, W_k). Therefore M_k contains the clause $(\Delta(i) \supset i)$, and for each predecessor p_w which has a black pebble on it, M_k contains the clause $(\Delta(p_w) \supset p_w)$. Furthermore, $Peb^1(C)$ contains the clause $(p_1, \wedge, ..., \wedge\ p_d \supset i)$ as an initial clause. In order to derive M_{k+1} from M_k, first drop the clause $(\Delta(i) \supset i)$. Next, we need to show how to turn each clause $(\Delta(v) \supset v)$ where $i \in \Delta(v)$ and turn it into the clause $((\Delta(v) - \{i\}) \bigcup_{1 \le w \le d} \Delta(p_w) \supset v)$.

To do this, first download initial clause $(p_1, \wedge, ..., \wedge\ p_d \supset i)$ into memory. All of its variables are already present in M_k, so this does not change the variable space. Then, for each clause $(\Delta(v) \supset v)$ where $i \in \Delta(v)$, do the following: Resolve $(\Delta(v) \supset v)$, with the initial clause $(p_1, \wedge, ..., \wedge\ p_d \supset i)$ to get $((\Delta(v) - \{i\})\ \cup\ \{p_1, ..., p_d\} \supset v)$. Finally, for each p_w which is a predecessor of i, if p_w has a black pebble on it in (B_k, W_k), then resolve with $(\Delta(p_w) \supset p_w)$. For each p_w with a white pebble on it, $\Delta(p_w) = \{p_w\}$, so this process results in the derivation of the clause $((\Delta(v) - \{i\}) \bigcup_{1 \le w \le d} \Delta(p_w) \supset v)$, all without increasing the variable space.

Case 6: This case is a slightly simpler form of the previous case. Once again, there are two sub-cases to consider:

Case 6a: This case is identical to 5a.

Case 6b: Suppose that i has in-degree d with predecessors $p_1, ..., p_d$. Let p_w be one of i's pebbled predecessors. Therefore M_k contains the clauses $(\Delta(i) \supset i)$ and $(\Delta(p_w) \supset p_w)$. To derive M_{k+1} from M_k, first drop the clause $(\Delta(i) \supset i)$. Next, download the initial clause $(p_w \supset i)$ into memory. All of its variables are already present in M_k, so this does not change the variable space. Then, for each clause $(\Delta(v) \supset v)$ where $i \in \Delta(v)$, turn $(\Delta(v) \supset v)$ into $((\Delta(v) - \{i\})\ \cup\ \{p_w\}) \supset v)$ by taking $(\Delta(v) \supset v)$ and resolving on variable i with $(p_w \supset i)$. None of these steps increase the variable space.

Therefore, in all cases, each M_{k+1} can be derived in RES-EM from M_k by induction such that the whole proof uses a variable space of at most $BW\text{-}Peb(C)$. This yields the configuration $M_j = \{(t)\}$ which corresponds to the final pebbling step $(B_j, W_j) = (\{t\}, \emptyset)$. In order to derive the empty clause, simply resolve with the initial clause $\{\neg t\}$, which does not change the variable space. Hence we can conclude that for any circuit C with target t, $VS(Peb^1(C) \vdash_{\mathsf{RES\text{-}EM}} \emptyset) \le BW\text{-}Peb(C)$, as required.

□

The following Lemma proves that RES and RES-EM are equivalent with respect to variable space:

Lemma 6.5. *For any formula F and clause C, $VS(F \vdash_{\mathsf{RES\text{-}EM}} C) = VS(F \vdash_{\mathsf{RES}} C)$.*

Proof: Let F be any arbitrary CNF formula and C be any clause.

$\Rightarrow$ Let π be a RES-EM derivation of C from F. Since resolving with law of excluded middle axioms $(a \vee \neg a)$ results in exactly the same clause that we started with, we may remove all such resolution steps to give us a RES derivation π' of C. Furthermore, since a must already be present for us to be able to resolve with $(a \vee \neg a)$, removing these steps does not change the variable space of π. Therefore $VS(F \vdash_{\mathsf{RES\text{-}EM}} C) \le VS(F \vdash_{\mathsf{RES}} C)$.

$\Leftarrow$ Let π be a RES derivation of C from F. Since every RES derivation is a RES-EM derivation, π is also a RES-EM derivation of C from F. Therefore $VS(F \vdash_{\mathsf{RES}} C) \le VS(F \vdash_{\mathsf{RES\text{-}EM}} C)$.

□

This allows us to prove our main result in this section, namely that the black-white pebbling number of monotone circuits and the RES variable space of their pebbling contradictions are equivalent:

Theorem 6.6. *For any monotone circuit C, $BW\text{-}Peb(C) = VS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset)$, where $BW\text{-}Peb(C)$ is defined without sliding.*

Proof: Let C be any arbitrary monotone circuit.

$\Rightarrow$ From Corollary 3.4 above, $BW\text{-}Peb(C) \leq VS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset)$.
$\Leftarrow$ Suppose that $VS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset) = k$. By Lemma 6.5, $VS(Peb^1(C) \vdash_{\mathsf{RES\text{-}EM}} \emptyset) = k$. But by Lemma 6.4, $VS(Peb^1(C) \vdash_{\mathsf{RES\text{-}EM}} \emptyset) \leq BW\text{-}Peb(C)$, so $k \leq BW\text{-}Peb(C)$. Therefore $VS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset) \leq BW\text{-}Peb(C)$.

□

This result together with Lemma 3.1 immediately gives us the corresponding result with sliding as a corollary:

Corollary 6.7. *For any monotone circuit C, $BW\text{-}Peb(C) = VS(Peb^1(C) \vdash_{\mathsf{RES}} \emptyset) - 1$, where $BW\text{-}Peb(C)$ is defined with sliding.*

6.2 The $\mathcal{PSPACE}$-Hardness of Resolution Variable Space

This section contains an immediate corollary to the results in the previous section, namely the $\mathcal{PSPACE}$-Hardness of the Resolution Variable Space Problem. More formally, the Resolution Variable Space Problem is defined as follows: given a formula F and integer k, does F have a RES refutation with variable space k? We shall refer to the language associated with this problem as RVS. The proof is a reduction from the $\mathcal{PSPACE}$-Complete black-white pebbling problem with sliding from [HP07].

Theorem 6.8. *RVS is $\mathcal{PSPACE}$-Hard.*

Proof: The reduction proceeds by simply taking the input (C, k), and outputting $(Peb^1(C), k-1)$. Clearly this is a polytime reduction, and the proof of correctness is immediate from Corollary 6.7. □

7 The EXPTIME-Completeness of Resolution Width

7.1 Introduction

The importance of *width* as a resource in resolution theorem proving has been emphasized in a remarkable paper by Ben-Sasson and Wigderson [BSW01]. In that article they prove a general theorem, of which one consequence is that if Σ is a contradictory set of clauses in 3-CNF, then the minimal size of a resolution refutation of Σ is exponential in the minimal width of such a refutation. In addition, they propose a simple dynamic programming procedure for automated theorem proving – one which simply searches for small width proofs. Both of these results motivate the problem of deciding whether or not a set of clauses has a proof of a given width.

The width problem for resolution is as follows: given a set of clauses Σ, and integer k as input, determine whether or not there is a resolution refutation of Σ of width bounded above by k. This problem was conjecture to be $\mathcal{EXPTIME}$-complete by Moshe Vardi. the present paper confirms Vardi's conjecture. The proof proceeds by a reduction from the existential k-pebble game, recently proved $\mathcal{EXPTIME}$-complete by Kolaitis and Panttaja [KP03].

7.2 Characterization of Resolution Width

In this section, we give a proof of the result of Atserias and Dalmau characterizing the width of resolution refutations [AD03]. The characterization is in terms of a two-player game which we shall call the *k-width game*, and it is played as follows: the players are the *Prover* and the *Adversary*.[1] The game is played as follows:

The players are given a set of clauses Σ, on a set V of variables, and an integer parameter $k > 0$. The players together construct a succession of assignments to the variables V. Initially, the assignment is empty; at every round of the game, all assignments involve at most k variables. Each round of the game proceeds as follows. First, the Prover can delete zero or more of the variable assignments from a previous round. Second, the Prover queries an unassigned variable, and the Adversary (re)assigns a value to it.

The Prover wins if the current assignment falsifies an initial clause in Σ. The Adversary wins if an earlier assignment is repeated during the play of the game.

Clearly every play of the game must eventually terminate with a win for the Prover or for the Adversary (Atserias and Dalmau define their game so that when the game continues infinitely, the Adversary wins). It follows that either the Prover or the Adversary must have a winning strategy.

Definition 7.1. *If Σ is a set of clauses on a set V of variables, then a non-empty family $\mathcal{F}$ of V-assignments is an* extendible k-family for Σ *if it satisfies the following conditions:*

1. *No assignment in $\mathcal{F}$ falsifies a clause in Σ;*
2. *If $\alpha \in \mathcal{F}$, and $\beta \subseteq \alpha$, then $\beta \in \mathcal{F}$;*
3. *If $\alpha \in \mathcal{F}$, $|\alpha| < k$, and $x \in V$, then there is a $\beta \in \mathcal{F}$, so that $\alpha \subseteq \beta$, and $\beta(x)$ is defined.*

The next theorem (Atserias and Dalmau [AD03]) shows that a resolution refutation of width k constitutes a winning strategy for the Prover, while an extendible $k+1$-family provides a winning strategy for the Adversary in the $k+1$-width resolution game.

Theorem 7.2. *Let Σ be a contradictory set of clauses, and $k \geq w(\Sigma)$. Then the following are equivalent:*

1. *There is no resolution refutation of Σ of width k;*
2. *There is an extendible $k+1$-family for Σ;*
3. *The Adversary wins the $k+1$-width game based on Σ.*

Proof: First, let us suppose that there is no resolution refutation of Σ of width k. Define $\mathcal{C}$ to be the set of all clauses having a resolution proof from Σ of width at most k; by assumption, $\Sigma \subseteq \mathcal{C}$. Let $\mathcal{F}$ be the set of all assignments of size at most $k+1$ that do not falsify any clause in $\mathcal{C}$. We claim that $\mathcal{F}$ is an extendible $k+1$-family for Σ. First, $\mathcal{F}$ is non-empty, because it contains the empty assignment (since $\mathcal{C}$ does not contain the empty clause). Second, $\mathcal{F}$ satisfies the first two conditions of Definition 7.1, by construction. To prove the fourth condition, let $\alpha \in \mathcal{F}$, and $|\alpha| \leq k$, $x \in V$, but there is no extension β of α in $\mathcal{H}$ with $\beta(x)$ defined. It follows that there is a clause $D \in \mathcal{C}$ that is falsified if we extend α by setting x to 0. Then $D = E \vee x$ for some E, since otherwise α would falsify D. Similarly, there is a clause $F \vee \overline{x}$ in $\mathcal{C}$ that is falsified by the extension of α that sets x to 1. Then α must falsify $E \vee F$; but $E \vee F$ is in $\mathcal{C}$, contradicting our assumption.

Second, let us suppose that there is an extendible $k+1$-family $\mathcal{F}$ for Σ. Then the Adversary can play the $k+1$-width game on Σ by responding to the Prover's queries with the appropriate assignment from the family, starting with the empty assignment. Since no assignment in the family falsifies an initial clause, this strategy must eventually end in a win for the Adversary, no matter how the Prover plays.

[1] Atserias and Dalmau, following the tradition of finite model theory, call their players the *Spoiler* and the *Duplicator*, but our terminology seems clearer in the present context

Finally, let us suppose that there is a resolution refutation of Σ of width k. Then the refutation provides the Prover with a winning strategy in the $k+1$-width game based on Σ. Starting from the empty clause at the root, the Prover follows a path in the refutation to one of the leaves in the refutation. At each round, the current assignment (after appropriate deletions), is a minimal assignment falsifying a clause in the path. The variable queried is the variable resolved upon to derive the current clause, and the next clause in the path is one of the premisses of the clause from the previous round. Since the refutation has width k, every assignment has size bounded by $k+1$, so this strategy must result in a win for the Prover. $\square$

Corollary 7.3. *The k-width resolution problem is in $\mathcal{EXPTIME}$.*

Proof: On a given play of the game, it is possible to keep track of the number of current assignments that have appeared up to a given round, and so determine if a repetition has occurred. Since there are at most $N = \binom{n}{k}3^k$ possible assignments, where n is the number of variables in the clause set Σ employed in the game, when the count reaches $N+1$, a repetition must have occurred.

Consequently, the description of the game given above shows that there is an alternating Turing machine operating in polynomial space that determines whether the Prover or the Adversary wins a given instance of the game. Hence, the problem is in $\mathcal{EXPTIME}$. $\square$

7.3 The Existential k-Pebble Game

The k-width resolution problem is a special case of the existential pebble game described in this section, as Atserias and Dalmau show in [AD03].

The *existential k-pebble game* [KV95], or *$(\exists, k)$-pebble game* for short, is played on two finite relational structures $\mathcal{A}$ and $\mathcal{B}$ of the same similarity type. A *partial homomorphism* φ between $\mathcal{A}$ and $\mathcal{B}$ is a mapping from a substructure of $\mathcal{A}$ to a substructure of $\mathcal{B}$ that preserves all of the relations in the structures; that is to say, for every relation $R^{\mathcal{A}}$ in $\mathcal{A}$, if $a_1, \ldots, a_m$ are in the domain of φ, and $R^{\mathcal{A}}(a_1, \ldots, a_m)$, then $R^{\mathcal{B}}(\varphi(a_1), \ldots, \varphi(a_m))$.

The $(\exists, k)$-pebble game, where $k \geq 2$ is a positive integer, is played by two players, the *Spoiler* and *Duplicator*, on the relational structures $\mathcal{A}$ and $\mathcal{B}$ (in the terminology of §7.2, the Prover plays the role of the Spoiler, the Adversary the role of the Duplicator). Each player has a set of k pebbles, numbered $1, \ldots, k$; we shall write $\{p_1, \ldots, p_k\}$ for the set of pebbles used by the Spoiler, and $\{q_1, \ldots, q_k\}$ for the set of pebbles used by the Duplicator. In each round of the game, the Spoiler can make one of two different moves: either removing a pebble p_i from a pebbled element of $\mathcal{A}$, or placing a free pebble p_j on an element of the domain of $\mathcal{A}$. To each move of the Spoiler, the Duplicator must respond, either by removing the corresponding numbered pebble q_i from an element of $\mathcal{B}$, or placing the pebble q_j on an element of $\mathcal{B}$.

The pebbles placed by the two players at any stage of the game define a relation R between the domains of $\mathcal{A}$ and $\mathcal{B}$; if a is an element of $\mathcal{A}$, and b of $\mathcal{B}$, then Rab holds if and only if there is a pebble p_i on a, and a corresponding pebble q_i on b. The Spoiler wins a play of the game at a given round if the relation R defined by the pebbling at that round is not a partial homomorphism between $\mathcal{A}$ and $\mathcal{B}$. The Duplicator wins if there is a repetition of an earlier position. As before, the game must terminate after a finite number of moves in a win for the Spoiler or the Duplicator, and so either the Spoiler or the Duplicator must have a winning strategy.

As in the case of the resolution game of §7.2, we can give a combinatorial characterization of a winning strategy for the Duplicator.

Definition 7.4. *Let $\mathcal{A}$ and $\mathcal{B}$ be two finite relational structures of the same similarity type. A non-empty family $\mathcal{H}$ of partial homomorphisms between $\mathcal{A}$ and $\mathcal{B}$ is an* extendible k-family *if it satisfies the following two conditions:*

1. $\mathcal{H}$ is closed under subfunctions: *If $h \in \mathcal{H}$, and $g \subseteq h$, then $g \in \mathcal{H}$.*

2. $\mathcal{H}$ has the k-extension property: *If $f \in \mathcal{H}$, $|f| < k$ and a is an element of $\mathcal{A}$, then there is an element b of $\mathcal{B}$ so that $f \cup \{\langle a, b\rangle\}$ is in $\mathcal{H}$.*

The characterization in the following theorem is the general result of which Theorem 7.2 is a special case.

Theorem 7.5. *Let $\mathcal{A}$ and $\mathcal{B}$ be two finite relational structures of the same similarity type, and k a positive integer. Then the following two statements are equivalent:*

1. *The Duplicator has a winning strategy for the $(\exists, k)$-pebble game on the structures $\mathcal{A}$ and $\mathcal{B}$.*

2. *There is an extendible k-family of partial homomorphisms for $\mathcal{A}$ and $\mathcal{B}$.*

Proof: See Kolaitis and Vardi [KV95, §4]. □

In the reduction of the next section, it is helpful to assume that the strategy for the Spoiler in the $(\exists, k)$-pebble game is of a restricted sort.

Lemma 7.6. *If there is a winning strategy for the Spoiler in the $(\exists, k)$-pebble game, then there is a strategy in which the Spoiler never places more than one pebble on any individual element of $\mathcal{A}$.*

Proof: If the Spoiler places a pebble p_j on an element of $\mathcal{A}$, where there is already a pebble p_i placed earlier in the game, then the Duplicator can respond by placing q_j on the same element of $\mathcal{B}$ as q_i. Any other response is an obvious blunder, since the relation defined from the resulting position is not a homomorphism. Consequently, such moves can give no advantage to the Spoiler, and so the Spoiler might as well play as if the Duplicator never makes such obvious blunders, and thus never place two pebbles on the same element of $\mathcal{A}$. □

7.4 Complexity of the k-Width Problem

In this section, we prove our main result by reducing the problem of determining the winner in $(\exists, k)$-pebble game to the k-width problem for resolution. The former problem was proved $\mathcal{EXPTIME}$-complete by Kolaitis and Panttaja [KP03]. In fact, they prove the stronger result, that a special case of this problem is $\mathcal{EXPTIME}$-complete, a fact that is useful in our reduction.

A *coloured graph* is a relational structure $\mathcal{A}$ of the form $\langle A, E, C_1, \ldots, C_m\rangle$, where E is a symmetric, irreflexive relation on A, and $C_1, \ldots, C_m$ are subsets of A. Using this notion, we can state the main result of Kolaitis and Panttaja.

Theorem 7.7. *The problem of determining whether the Duplicator has a winning strategy in the $(\exists, k)$-pebble game on the structures $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ are coloured graphs of the same similarity type, is $\mathcal{EXPTIME}$-complete under logspace reducibility.*

Proof: See Kolaitis and Panttaja [KP03]. □

We now give a reduction of the $(\exists, k)$-pebble game problem to the width problem for resolution, by translating the first problem into a set of clauses in 3-CNF.

Definition 7.8. *Let $\mathcal{A} = \langle A, E, C_1, \ldots, C_m\rangle$ and $\mathcal{B} = \langle B, F, D_1, \ldots, D_m\rangle$ be coloured graphs, with $A = \{a_1, \ldots, a_p\}$ and $B = \{b_1, \ldots, b_q\}$. For every i, $1 \leq i \leq p$, $\Sigma(\mathcal{A}, \mathcal{B})$ contains q variables P^i_j, for each j, $1 \leq j \leq q$, and in addition, $q - 1$ auxiliary variables Q^i_j, for $1 \leq j < q$. The clauses constituting $\Sigma(\mathcal{A}, \mathcal{B})$ are as follows:*

1. *Q^i_1, for $1 \leq i \leq p$.*

2. *$Q^i_j \leftrightarrow (P^i_j \vee Q^i_{j+1})$, for $1 \leq j < q - 1$, and $Q^i_{q-1} \leftrightarrow (P^i_{q-1} \vee P^i_q)$.*

3. *$\neg P^i_j$, where $a_i \in C_r$, $b_j \notin D_r$, for some r.*

4. $\neg P_j^i \vee \neg P_t^s$*, where* $E(a_i, a_s)$*, but not* $F(b_j, b_t)$*.*

5. $\neg P_j^i \vee \neg P_k^i$*, where* $1 \leq j < k \leq q$*.*

The set of clauses $\Sigma(\mathcal{A}, \mathcal{B})$ is satisfiable if and only if there is a homomorphism from $\mathcal{A}$ to $\mathcal{B}$, and is logspace constructible from $\mathcal{A}$ and $\mathcal{B}$. If α is an assignment to the variables of $\Sigma(\mathcal{A}, \mathcal{B})$, we define $\mathrm{Dom}(\alpha)$ to be the set of all i for which $\alpha(P_j^i)$ is defined, for some j.

If $R \subseteq A \times B$, then we write $\alpha[R]$ for the assignment defined by: $\alpha[R](P_j^i) = 1$ if and only if $R(a_i, b_j)$. Furthermore, if f is a mapping from a subset of A to B, then we define the assignment $\beta[f]$ determined by f as follows. If $i \in \mathrm{Dom}(f)$, then $\beta[f](P_j^i) = 1$ if $f(a_i) = b_j$, and $\beta[f](P_j^i) = 0$ otherwise, while $\beta[f](Q_k^i) = 1$ if $f(a_i) = b_j$, for $k \leq j$, and $\beta[f](Q_j^i) = 0$ otherwise. It is easily checked that if f is a partial homomorphism from $\mathcal{A}$ to $\mathcal{B}$, $\beta[f]$ does not falsify any clause in $\Sigma(\mathcal{A}, \mathcal{B})$.

Lemma 7.9. *If* $\mathcal{A}$ *and* $\mathcal{B}$ *are coloured graphs, and* $k \geq 3$*, then the Spoiler has a winning strategy for the* $(\exists, k)$*-pebble game on* $\mathcal{A}$ *and* $\mathcal{B}$ *if and only if the Prover has a winning strategy for the* $k + 2$*-width game on* $\Sigma(\mathcal{A}, \mathcal{B})$*.*

Proof: $(\Rightarrow)$ First, assume that the Spoiler has a winning strategy for the $(\exists, k)$-pebble game on $\mathcal{A}$ and $\mathcal{B}$. By Lemma 7.6, we can assume that the strategy for the Prover never involves doubly pebbled elements; this implies that every relation R_t produced by the Spoiler's strategy is a map from a subset of A to B. Then the Prover has a winning strategy for the $k + 2$-width game that follows the Spoiler's strategy. At each stage in the strategy, the current assignment α maintained by the Prover contains a set of at most k variables of the form P_j^i, all of them set to 1, and representing a partial map from A to B. In addition, α assigns values to at most two extra variables, each of these being either a variable P_q^i, or an auxiliary variable of the form Q_j^i.

The Prover's strategy consists of successively forcing the Adversary to assign the value 1 to variables of the form P_j^i, in such a way as to produce a series of assignments of the form $\alpha[R_0], \alpha[R_1], \ldots, \alpha[R_t], \ldots$, where $R_0, R_1, \ldots, R_t, \ldots$ are the relations $R_t \subseteq A \times B$ produced by following the Spoiler's strategy in the $(\exists, k)$-pebble game on $\mathcal{A}$ and $\mathcal{B}$.

Let us suppose that the Prover's current assignment is of the form $\alpha[R_t]$. If R_{t+1} is produced from R_t by removing pebbles, then the Prover simply deletes the appropriate variable assignment from $\alpha[R_t]$. So, let us suppose that the Spoiler places a free pebble on an element a_i of A, so that $|R_t| < k$. The Prover must now force the Adversary to set at least one variable P_j^i to 1.

The Prover begins by querying Q_1^i, which the Adversary is forced to set to 1; the Prover then queries P_q^i. If the Adversary assigns this last variable the value 1, then the Prover has succeeded. Otherwise, the Prover performs a binary search in the sequence of variables $\sigma = Q_1^i, \ldots, Q_{q-1}^i, P_q^i$. At each stage in the search, the Prover retains two variables from σ in the current assignment, the first set to 1, the second set to 0. The next variable queried is chosen so as to cut the interval between the two variables in σ as nearly in half as possible. This search procedure must terminate with two consecutive variables in σ set to 1 and 0, respectively, say, Q_j^i and Q_{j+1}^i. Then the Adversary must assign 1 to the variable P_j^i on the next query, otherwise a clause of type 2 is falsified. Once the Prover has succeeded in forcing a response of this kind from the Adversary, the assignments to the extra variables are deleted, and the strategy continues with the new assignment of the form $\alpha[R_{t+1}]$, with $i \in \mathrm{Dom}(\alpha[R_{t+1}])$.

Since the strategy described above corresponds to a winning strategy for the Spoiler, any play using this strategy terminates in a win for the Prover, since it must end when one of the assignments $\alpha[R_t]$ falsifies a clause of type 3, 4 or 5 in Definition 7.8 (the Spoiler wins the $(\exists, k)$-pebble game when R_t is not a partial homomorphism from $\mathcal{A}$ to $\mathcal{B}$).

$(\Leftarrow)$ Second, assume that the Duplicator has a winning strategy for the $(\exists, k)$-pebble game on $\mathcal{A}$ and $\mathcal{B}$. We describe a winning strategy for the Adversary in the k-width game, based on the Duplicator's winning strategy. By Theorem 7.5, there is an extendible k-family $\mathcal{H}$ of homomorphisms for $\mathcal{A}$ and $\mathcal{B}$. At each round in a play of the k-width game, the Adversary has a partial homomorphism belonging to $\mathcal{H}$. Initially, this homomorphism is empty; the homomorphism is updated as the play proceeds. We now describe the Adversary's update procedure.

The Adversary plays so as to maintain the following properties invariant throughout the game:

1. f is a partial homomorphism in $\mathcal{H}$, and $|f| \leq k$.

2. $\{i : a_i \in \mathrm{Dom}(f)\} \subseteq \mathrm{Dom}(\alpha)$.

Let α be the assignment at the start of a given round of the k-width game, and $f \in \mathcal{H}$ the partial homomorphism that the Adversary has available at the end of the previous round. Initially, the Prover removes some variable assignments from α. If this results in the removal of some $i \in \mathrm{Dom}(\alpha)$, where $a_i \in \mathrm{Dom}(f)$, then f is restricted appropriately; that is to say, the new assignment f' is defined by $f' := f\restriction\{a_i : i \in \mathrm{Dom}(\alpha)\}$.

We now need to describe the variable querying part of a round. Let α be the current assignment maintained by the Prover, and f the partial homomorphism maintained by the Adversary. When the Prover queries an unset variable of the form P^i_j or Q^i_j, three cases arise:

1. If $a_i \in \mathrm{Dom}(f)$, then the Adversary answers in accordance with the assignment $\beta[f]$.

2. If $a_i \notin \mathrm{Dom}(f)$, but $|f| < k$, then the Adversary extends f to a new partial homomorphism $g \in \mathcal{H}$, with $a_i \in \mathrm{Dom}(g)$, and then replies according to $\beta[g]$.

3. If $a_i \notin \mathrm{Dom}(f)$, but $|f| = k$, then the Adversary sets any variable P^i_j to 0, and any variable Q^i_j to 1.

We need to prove that this strategy on the part of the Adversary succeeds. This amounts to showing that whenever the Adversary answers a query, that the resulting assignment never falsifies an initial clause in $\Sigma(\mathcal{A}, \mathcal{B})$. In the first two cases, this is clearly true, by the definition of an extendible k-family. Only the third case causes difficulties. Let α' be the extension of α after the Adversary's reply. In this case, there are at most two variables assigned values by α' that are not assigned values by $\beta[f]$. Since all such variables P^i_j are set to 0 by the Adversary, no clause of types 3,4 or 5 can be falsified by α', and since all such variables Q^i_j are set to 1, no clause of type 1 can be falsified. Finally, since at most two such variables are assigned values, no clause of type 2 can be falsified by α', completing the verification that the strategy used by the Adversary always succeeds in the $k+2$-width game using the set of clauses $\Sigma(\mathcal{A}, \mathcal{B})$. $\square$

Theorem 7.10. *The k-width problem for resolution is $\mathcal{EXPTIME}$-complete under logspace reducibility.*

Proof: Theorem 7.7 of Kolaitis and Panttaja shows that the problem of determining the winner in an instance of the $(\exists, k)$-pebble game on coloured graphs $\mathcal{A}$ and $\mathcal{B}$ of the same similarity type is $\mathcal{EXPTIME}$-complete under logspace reducibility. Theorem 7.2 and Lemma 7.9 provide a logspace reduction of this problem to the $k+1$-width problem for $\Sigma(\mathcal{A}, \mathcal{B})$. $\square$

Corollary 7.11. *The problem of determining the winner in an instance of the $(\exists, k)$-pebble game on relational structures $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ is a two-element structure, is $\mathcal{EXPTIME}$-complete.*

Albert Atserias has remarked that the preceding theorem also settles the complexity of the fixed template version of the existential k-pebble game. Feder and Vardi [FV98] (see also [Ats04]) observe that the satisfiability problem for formulas in r-CNF can be encoded as a constraint satisfaction problem in the form where the target structure $\mathcal{B}$, or 'template' is fixed, while only the source structure, or 'instance' varies.

Corollary 7.12. *The problem of determining whether the Duplicator has a winning strategy in the $(\exists, k)$-pebble game on structures $\mathcal{A}$ and $\mathcal{B}$, where $\mathcal{B}$ is a fixed template on a two-element universe, is $\mathcal{EXPTIME}$-Complete under logspace reducibility.*

Acknowledgements

We would like to express our thanks to Steve Cook, who originally told us of the resolution width problem, and to Moshe Vardi for his stimulating conjecture, as well as his comments on an earlier draft. We would also like to thank Albert Atserias for his observation about the fixed template problem.

References

[AD03] A. Atserias and V. Dalmau. A Combinatorial Characterization of Resolution Width. *Proc. of the 18th IEEE Conference on Computational Complexity*, 2003.

[Ats04] Albert Atserias. On Sufficient Conditions for Unsatisfiability of Random Formulas. *Journal of the Association for Computing Machinery*, 51:281–311, 2004. Preliminary version, 17th IEEE Symposium on Logic in Computer Science (LICS), pages 275-284, 2002.

[BS02] E. Ben-Sasson. Size Space Tradeoffs For Resolution. *Proceedings of the 34th ACM Symposium on the Theory of Computing*, pages 457 – 464, 2002.

[BSIW04] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near Optimal Separation of Tree-like and General Resolution. *Combinatorica*, Vol. 24, Issue 4:585 – 604, 2004.

[BSW01] E. Ben-Sasson and A. Wigderson. Short Proofs are Narrow -Resolution Made simple. *Journal of the Association for Computing Machinery*, 48:149–169, 2001. Preliminary version: Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 517-526.

[CK01] P. Clote and E. Kranakis. *Boolean Functions and Computation Models*. Springer-Verlag, Berlin, 2001.

[Coo73] S. A. Cook. An Observation on Time-Storage Tradeoff. *Proceedings of the 5th Annual ACM Symposiumon Theory of Computing*, 1973.

[CS76] S. Cook and R. Sethi. Storage Requirements for Deterministic Polynomial Time Recognizable Languages. *J. of Computer & System Sciences*, pages 25 – 37, 1976.

[ET01] J. Esteban and J. Torán. Space Bounds for Resolution. *Information and Computation*, 171:84 – 97, 2001.

[ET03] J. Esteban and J. Torán. A Combinatorial Characterization of Treelike Resolution Space. *Information Processing Letters*, 87:295–300, 2003.

[FV98] Tomás Feder and Moshe Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *Siam Journal on Computing*, 28:57–104, 1998.

[GLT80] J. R. Gilbert, T. Lengauer, and R. E. Tarjan. The Pebbling Problem is Complete in Polynomial Space. *SIAM Journal of Computing*, Vol. 9, Issue 3:513 – 524, 1980.

[Hei81] F. Meyer Auf Der Heide. A Comparison of Two Variations of a Pebble Game on Graphs. *Theoretical Computer Science*, 13:315 – 322, 1981.

[HHM07] A. Hertel, P. Hertel, and C. Morgan. A Sound and Complete Proof Theory for Propositional Logical Contingencies. *Notre Dame Journal of Formal Logic (Accepted)*, 2007.

[HHU07] A. Hertel, P. Hertel, and A. Urquhart. Formalizing Dangerous Reductions. *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT 2007)*, 2007.

[HP07] P. Hertel and T. Pitassi. Optimal Size / Space Tradeoffs For Resolution. *Submitted to FOCS 2007*, 2007.

[Hru06] P. Hrubes. A Lower Bound for Intuitionistic Logic. Unpublished Manuscript, 2006.

[KP03] Phokion G. Kolaitis and Jonathan Panttaja. On the complexity of existential pebble games. In *Proceedings of Computer Science Logic CSL '03*, pages 314–329. Springer, 2003. Lecture Notes in Computer Science 2803.

[KV95] Phokion G. Kolaitis and Moshe Y. Vardi. On the expressive power of Datalog: Tools and a case study. *Journal of Computer and System Sciences*, 51:110–134, 1995.

[Lin78] A. Lingas. A PSPACE-Complete Problem Related to a Pebble Game. In *Proceedings of the Fifth Colloquium on Automata, Languages and Programming*, pages 300 – 321, London, UK, 1978. Springer-Verlag.

[Nor06] J. Nordström. Narrow Proofs May Be Spacious: Separating Space and Width in Resolution. *Proc. of the 38th ACM Symposium on the Theory of Computing*, 2006.

[PI00] P. Pudlák and Russell Impagliazzo. Lower Bounds for DLL Algorithms for k-SAT. *Proceedings of SODA 2000*, 2000.

[Pip80] N. Pippenger. Pebbling. *Proceedings of the 5th IBM Symposium on Mathematical Foundations of Computer Science, Japan (Technical Report RC8528, IBM Watson Research Center)*, 1980.

[Sav70] W. Savitch. Relationships Between Nondeterministic and Deterministic Tape Complexities. *Journal of Computer and System Sciences*, 4:177 – 192, 1970.

[Urq06] A. Urquhart. Width Versus Size in Resolution Proofs. *Proceedings of the 3rd Annual Conference on Theory and Applications of Models of Computation*, pages 79 – 88, 2006.