Game Characterizations and the PSPACE-Completeness of Tree Resolution Space

Alexander Hertel & Alasdair Urquhart *

June 18, 2007

Abstract

The Prover/Delayer game is a combinatorial game that can be used to prove upper and lower bounds on the size of Tree Resolution proofs, and also perfectly characterizes the space needed to compute them. As a proof system, Tree Resolution forms the underpinnings of all DPLL-based SAT solvers, so it is of interest not only to proof complexity researchers, but also to those in the area of propositional reasoning. In this paper, we prove the PSPACE-Completeness of the Prover/Delayer game as well as the problem of predicting Tree Resolution space requirements, where space is the number of clauses that must be kept in memory simultaneously during the computation of a refutation. Since in practice memory is often a limiting resource, researchers developing SAT solvers may wish to know ahead of time how much memory will be required for solving a certain formula, but the present result shows that predicting this is at least as hard as it would be to simply find a refutation.

1 Introduction & Motivation

The Tree Resolution (T-RES) proof system has been studied extensively, and is understood quite well. Its algorithmic incarnation, DPLL, forms the basis of many SAT-solvers including clause learning algorithms. Any lower bounds proved for T-RES immediately imply lower bounds for real-world DPLL algorithms. However, compared with Resolution proof size, the amount of space needed to compute a proof has not been studied nearly as extensively. The usual definition of a Resolution proof is that it is a simple sequence of clauses, but we can redefine this so that only a small number of clauses are stored in a working memory space, typically much smaller than the size of the proof itself.

The problem of determining the space required for Resolution proofs seems to have been suggested for the first time by Armin Haken in 1998, and was explored in a paper by Alekhnovich, Ben-Sasson, Razborov and Wigderson [1] and independently by Esteban and Torán [7]; Ben-Sasson proved an important tradeoff between Resolution size and space in [3]. In addition, a number of other results relating Resolution space and width are known [2, 11].

From a practical point of view, DPLL and clause learning algorithms have been highly successful at solving SAT and SAT-related problems. The main limiting factor on these algorithms is space, namely the size of the cache used for memoization. This has inspired much research into methods for pruning space in a Resolution search. Thus there are both practical and theoretical motivations for understanding Resolution space as a resource.

*The authors gratefully acknowledge NSERC and the University of Toronto Department of Computer Science for supporting this research.

In this paper we prove that for T-RES, determining clause space requirements is $\mathcal{PSPACE}$-Complete, which unfortunately implies that computing the T-RES space requirements for a formula is at least as hard as actually refuting it, and shows that it is therefore probably not feasible for researchers working with DPLL-based SAT solvers to predict a formula's memory needs. In addition, there are many formulas on which these SAT solvers fail, and it would be very useful to be able to tell ahead of time if this will be the case. If there was any hope of using T-RES space bounds to predict this, then the present result casts some serious doubt on that plan.

The key to our main theorem is the intimate connection between Resolution space and games, and we combine three previous results and ideas in order to prove that both the Prover/Delayer game and T-RES clause space problem are $\mathcal{PSPACE}$-Complete. The high-level idea behind this paper is as follows: From [7], Prover/Delayer number is known to equal T-RES clause space, and from [10], pebbling Lingas circuits is known to be $\mathcal{PSPACE}$-Complete. But the former deals with formulas, and the latter uses circuits, so we use pebbling contradiction formulas to bridge this gap. Specifically, we give tight bounds relating the black pebbling number of any Lingas circuit C to the Prover/Delayer number of the formula $Peb^2(C)$, thereby proving the $\mathcal{PSPACE}$-Completeness of the Prover/Delayer game (and T-RES clause space by the equivalence from [7]).

2 Definitions

We assume that the reader is familiar with basic proof complexity, circuit complexity, general complexity theory, and use [5] as our standard reference. By a *DAG* we mean a directed acyclic graph in which all nodes except source nodes have fan-in two, but unbounded fan-out, and there is a unique node (the *sink*, *target* or *output* node) with fan-out zero. A *monotone circuit* is defined as a DAG in which all nodes except the sources are labelled with AND or OR.

2.1 Resolution Space

The definition of T-RES clause space requires a non-standard definition of T-RES proof that depends on the notion of configuration:

Definition 2.1 (Configuration-Style T-RES Proof). *A configuration $\mathbb{C}$ is a set of clauses. The sequence of configurations $\pi = \mathbb{C}_0, \mathbb{C}_1, ..., \mathbb{C}_k$ is a T-RES proof of C from the formula 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 deleting one or more of its clauses, adding the resolvent of two clauses of $\mathbb{C}_i$* ***and deleting both parent clauses****, or adding one or more of the clauses of F (initial clauses).*

This leads us to our definition of space. Intuitively, space is the amount of memory required in order to compute π:

Definition 2.2 (Clause Space). *Let F be a set of clauses and π be a configuration-style T-RES proof of clause C from F. The tree clause space of π, denoted $TCS(\pi)$ is the maximum number of clauses in any configuration of π. The tree clause space of resolving C from F, denoted $TCS(F \vdash_{\mathsf{T\text{-}RES}} C)$, is the minimum $TCS(\pi)$ over all T-RES proofs π of C from F.*

2.2 Pebbling Circuits & Games

The investigation of Resolution space is closely associated with the well-known pebbling game and pebbling number of a DAG, originally explored in [6] as a means of investigating bounds on storage requirements.

We define the *generalized black pebbling game* as a single-player game in which the goal is to 'pebble' the target node of a monotone circuit C. The game has the following rules: It starts with no pebbles on the circuit. At any point, the player may place a pebble onto any source node, or remove a pebble from any node. For any AND gate v, if both of v's immediate predecessors have pebbles on them, then the player may place a pebble on v, or slide a pebble from a predecessor to v. Similarly, for any OR gate v, if at least one of v's immediate predecessors has a pebble on it, then the player may place a pebble on v, or slide a pebble from a predecessor to v. The game ends once the target node has a pebble on it.

Most of the black pebble games found in the literature can be viewed as restricted versions of this generalization. In particular, if all gates in the circuit are AND gates, then this game is equivalent to the original pebbling game on DAGs defined by Cook and Sethi [6].

Definition 2.3 (Black Pebbling Number of a Monotone Circuit). *The 'black pebbling number' of a monotone circuit C, denoted $B\text{-}Peb(C)$, is the minimum number of pebbles needed in order to pebble C's target node using the above rules.*

Each version of the pebbling game can either be defined with or without sliding; for the purposes of this paper, we shall always allow sliding.

2.3 Pebbling Contradictions

The formulas we call 'pebbling contradictions' are based on the various forms of the pebbling game. Ben-Sasson gives a brief history of these formulas in [3]. The particular contradictory formulas we employ here are a generalization of those used by Ben-Sasson, Impagliazzo and Wigderson to give a near-optimal separation of tree-like and general resolution [4]. They can be interpreted as making the following contradictory claim: "The input nodes of a monotone circuit C are all set to true, but the output node is set to false.", where we represent "node v is set to true" by the disjunction $v_0 \vee v_1$.

Definition 2.4 (Pebbling Contradictions Based on Circuits). *Let C be a monotone circuit. The set of clauses $Peb^2(C)$ contains the following clauses:*

1. *For each source node s, the clause $\{s_0, s_1\}$;*
2. *For each AND node c with immediate predecessors a and b, 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, 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. *For the target node t, two singleton clauses $\{\neg t_0\}$ and $\{\neg t_1\}$.*

2.4 The Prover/Delayer Game

The Prover/Delayer game (P/D game), described in [4, 12], is a combinatorial game between two players, the 'Prover', and the 'Delayer,' and is played on an unsatisfiable CNF formula F. The goal of the Prover is to falsify some initial clause of F. Since the formula is unsatisfiable, this is inevitable.

The game proceeds in rounds. Each round starts with the Prover querying the value of a variable. The Delayer can give one of three answers: 'True', 'False', or 'You Choose'. If the Delayer says 'You Choose', then the Prover gets to decide the value of that variable, and the Delayer wins one point. This is the only way in which points can be scored. The game finishes when any clause has been falsified. The Delayer's aim is to win as many points as possible, while the Prover aims to minimize this quantity.

Definition 2.5. *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 Space & Games

The pebbling game is closely related to Resolution clause space. Let F be any arbitrary unsatisfiable formula, π a configuration-style RES refutation of F, and G the DAG underlying the structure of π. 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. Of course, the analogous idea holds for T-RES and tree clause space.

Esteban & Torán proved that the tree clause space of any unsatisfiable formula is essentially the same as its Prover/Delayer number:

Theorem 3.1 ([8]). *For any unsatisfiable CNF formula F, $TCS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset) = PD(F) + 1$.*

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 yields an upper bound on the size of T-RES proofs [7]. Combining this with Theorem 3.1 and a little extra work shows that an upper bound on the number of points scored by the Prover immediately gives an upper bound on T-RES size:

Corollary 3.2. *If F is a contradictory formula with $PD(F) \leq k$, then F has a* T-RES *proof of size $\leq \binom{n}{k+1}$.*

The P/D game can also be used to give a lower bound on T-RES size; in [4], Ben-Sasson, Impagliazzo and Wigderson prove that lower bounds on $PD(F)$ can be used to prove lower bounds on the size of T-RES proofs:

Theorem 3.3 ([4]). *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 $PD(F)$ not only immediately imply tightly corresponding upper and lower bounds on $TCS(F \vdash_{\mathsf{T\text{-}RES}} \emptyset)$, but they also imply upper and lower bounds for T-RES proof size, showing not only that the P/D game is intimately connected to the T-RES proof system, but also that T-RES size and space are closely related.

3.2 Complexity of Pebbling

A number of complexity results involving pebbling are known. In 1978, Andrzej Lingas proved the following:

Theorem 3.4 ([10]). *Given a monotone circuit C and an integer k, the problem of determining if C can be black-pebbled with k pebbles is $\mathcal{PSPACE}$-Complete.*

Gilbert, Lengauer and Tarjan extended this result to DAGs in 1980:

Theorem 3.5 ([9]). *Given a DAG G and an integer k, the problem of determining if G can be black-pebbled with k pebbles is $\mathcal{PSPACE}$-Complete.*

These results are particularly interesting because the pebbling game only has one player, whereas most $\mathcal{PSPACE}$-Complete games have two.

4 Main Results

4.1 An Easy Case of the Pebbling Game

Although pebbling circuits is $\mathcal{PSPACE}$-Complete in general, 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 a vertex in a graph:

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

A source node has a pebbling number of 1, while if c has predecessors a and b, then $\text{Peb}(c) \leq \max(\text{Peb}(a), \text{Peb}(b)) + 1$. This is because we can always pebble c with the following strategy: Assuming that $\text{Peb}(a) \leq \text{Peb}(b)$, first pebble b using $\text{Peb}(b)$ pebbles, then remove all the pebbles except the one on b, next pebble a using $\text{Peb}(a)$ extra pebbles, then slide the pebble on a to c.

Definition 4.2 (Increasing DAGs). $\mathcal{G}_{\mathcal{I}} = \{G \mid G$ *is a DAG 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 the pebbling game on binary DAGs is $\mathcal{PSPACE}$-Complete in general, when we restrict ourselves to inputs from $\mathcal{G}_{\mathcal{I}}$, the problem becomes much easier:

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

Proof: Since the pebbling number of each node in G is uniquely determined by those of its predecessors, simply start at the source nodes and set 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 target node's number has been determined. Clearly this can be done in polynomial time. $\square$

This gives some insight into why the pebbling game is so difficult. An inspection of the constructions from both [9] and [10] shows that the graphs resulting from the reductions contain a large number of vertices that have the same pebbling numbers as their predecessors, so the obvious algorithm above fails.

4.2 Prover Strategy for the $\mathcal{G}_{\mathcal{I}}$ DAGs

In this section we describe an efficient Prover strategy for the $\mathcal{G}_{\mathcal{I}}$ graphs:

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

Proof: In the interests of concision, we omit a detailed proof, and instead provide the following proof sketch: The Prover first labels each node in G with its pebbling number, and first queries the variables associated with the target. The Delayer must set them both to False, or else the game is over. The Prover then follows a path towards the source nodes, forcing the Delayer to set both variables c_0 and c_1 associated with each node c on the path to False.

Given any node c on the path, let us suppose that it has both variables c_0 and c_1 set to False, and that the predecessors of c are a and b. In all cases, it is possible for the Prover to query the variables associated with a and b in such a way that sets both a_0 and a_1, or b_0 and b_1 to False while giving up at most one point every time the pebbling number decreases. Eventually the path will reach a node with two source nodes as its children, in which case it is easy to see that the Delayer can win at most 2 more points. Since the Delayer wins at most 1 point every time the pebbling number decreases, and since only 2 points can be won in the base case, the Prover limits the Delayer to at most $B\text{-}Peb(G)$ points, as required. $\square$

4.3 Prover Strategy for the Lingas Circuits

Lingas' 1978 paper [10] shows that the black pebbling game on monotone circuits is $\mathcal{PSPACE}$-Complete by reducing from the 3-QBF problem with alternating quantifiers. This is done by taking a formula F and outputting 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 builds a number of 'widgets' based on F. The original example from [10] can be seen below in Figure 1. The construction includes one widget for each quantifier, literal, and clause, as well as one pyramid graph that 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 [6], and an example is given in the upper corner of the diagram.

The pebbling number output by the reduction is $k = 2U + E + M$, where U, E, and M are the respective number of universal quantifiers, existential quantifiers, and clauses in F. Intuitively, the reduction works by forcing any pebbling to simulate a brute force QBF proof system. For each universal quantifier, the pebbling must travel up both sides, requiring that the entire circuit below that point be re-pebbled, thereby ensuring that F is true.

We refer to the circuits corresponding to true QBF formulas as the 'Lingas circuits', defined formally as follows:

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

We shall make use of the $\mathcal{PSPACE}$-Completeness of the Lingas circuits to prove the $\mathcal{PSPACE}$-Completeness of the P/D game as well as the T-RES clause space problem. To this end we must first prove that for any Lingas circuit C, when playing on $Peb^2(C)$, the Prover has a strategy limiting the Delayer to at most $B\text{-}Peb(C)$ points. The strategy is quite simple: the Prover first forces the Delayer to admit that both variables associated with the target node must be False. As with the $\mathcal{G}_{\mathcal{I}}$ DAGs, the Prover then proceeds to propagate this 'Double False' setting downwards.

We show that it is possible to traverse each existential widget while giving up at most one point, each universal widget while giving up at most two, and finally that it is possible to traverse the conjunctive pyramid and derive the ultimate 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 $B\text{-}Peb(C)$ points. A contradiction can be obtained (thereby ending the game) by falsifying an initial clause of $Peb^2(C)$.

The Prover's strategy is interesting in that whenever the Delayer responds 'You Choose' to a query, the Prover shall always respond with 'True'. Note that the Delayer has the choice of three different responses after each variable is queried, but our decision tree only needs to be binary, 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.

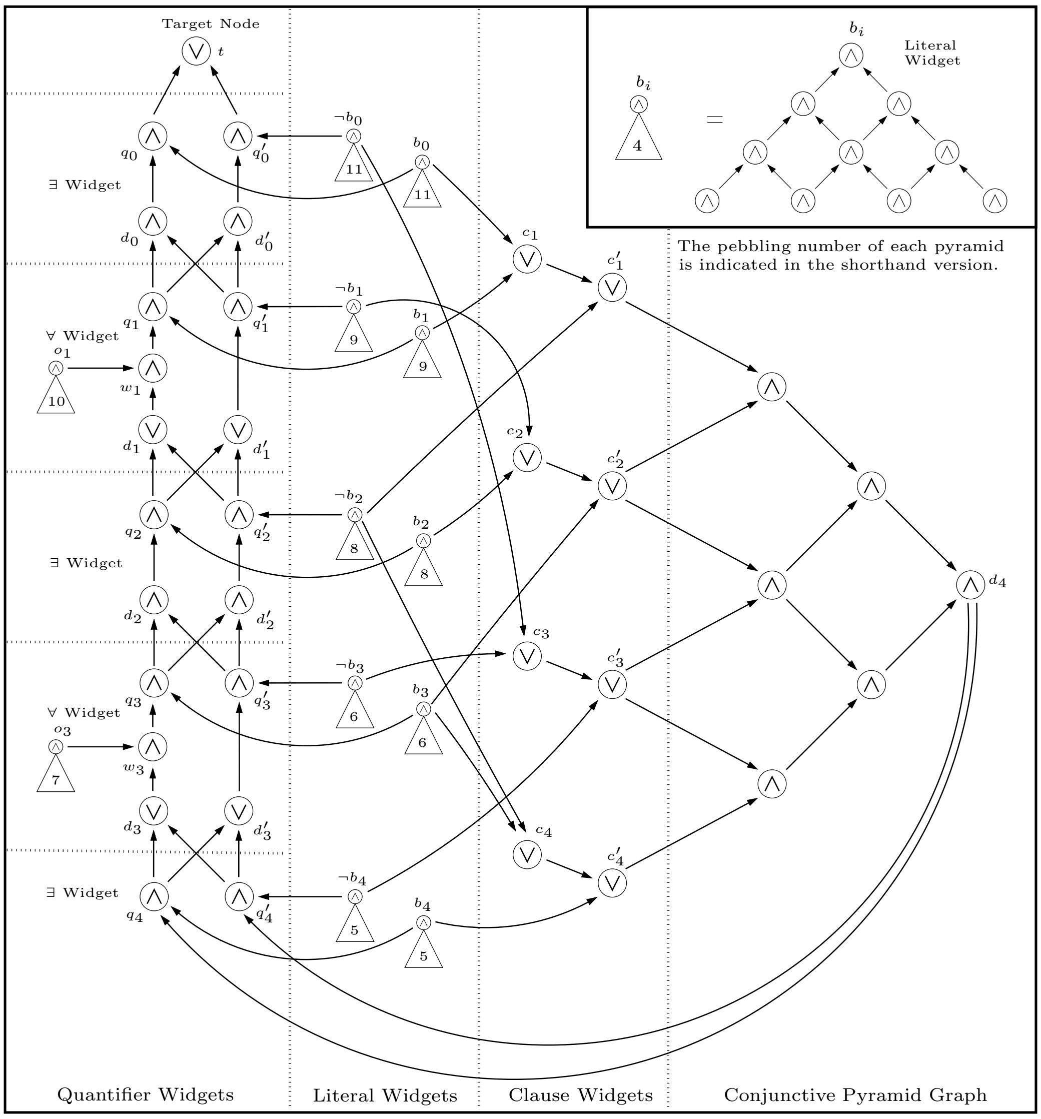

Figure 1: The monotone circuit resulting from applying Lingas's reduction to the true formula $F = \exists x_0 \forall x_1 \exists x_2 \forall x_3 \exists x_4\ (x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_0 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_2 \vee x_3 \vee x_4)$. An example of a pyramid graph is shown in the upper corner. In this case $k = 2 \times 2 + 3 + 4 = 11$, but if F were false, then the circuit would require $k + 1$ pebbles and could not be pebbled using only k.

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

Proof. The Prover's strategy proceeds as follows: first query t_0 and then t_1. The Delayer must set these variables to False, or else the game is over. The Delayer has therefore scored no points, and we enter the first quantifier widget with k points remaining. Next the Prover inductively traverses the quantifier widgets in order, propagating the 'Double False' setting downwards towards the conjunctive pyramid. The Prover has a strategy that gives up at most one point while traversing an existential widget, and at most two points when traversing a universal widget.

The existential widget and the Prover's strategy for traversing it are shown in Figure 2. We assume that we have j points remaining before the Delayer reaches k points, and that both variables associated with either the node d_{i-1} or d'_{i-1} have been set to False. Decision tree edges are labelled with the different possibilities at each step during the P/D game; 'YC,T' represents the case when the Delayer says 'You Choose', and the Prover replies 'True', and 'F' represents the case when the Delayer says 'False'. Leaves are labelled in the P/D game outcomes which they lead to; those 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 that has pebbling number j. Since each 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. This is 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.

The universal widget and the Prover's strategy for traversing it are shown in Figure 2. We have j points remaining before the Delayer reaches k points, and both variables associated with either the node d_i or d'_i have been set to False. Once again, some paths lead to the Delayer scoring 3 points, but this is all right even if this is the final quantifier widget, since we have at least one point's leeway. Also as before, the pyramid graphs corresponding to both the literal widgets and 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. Note the two leaves 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, 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

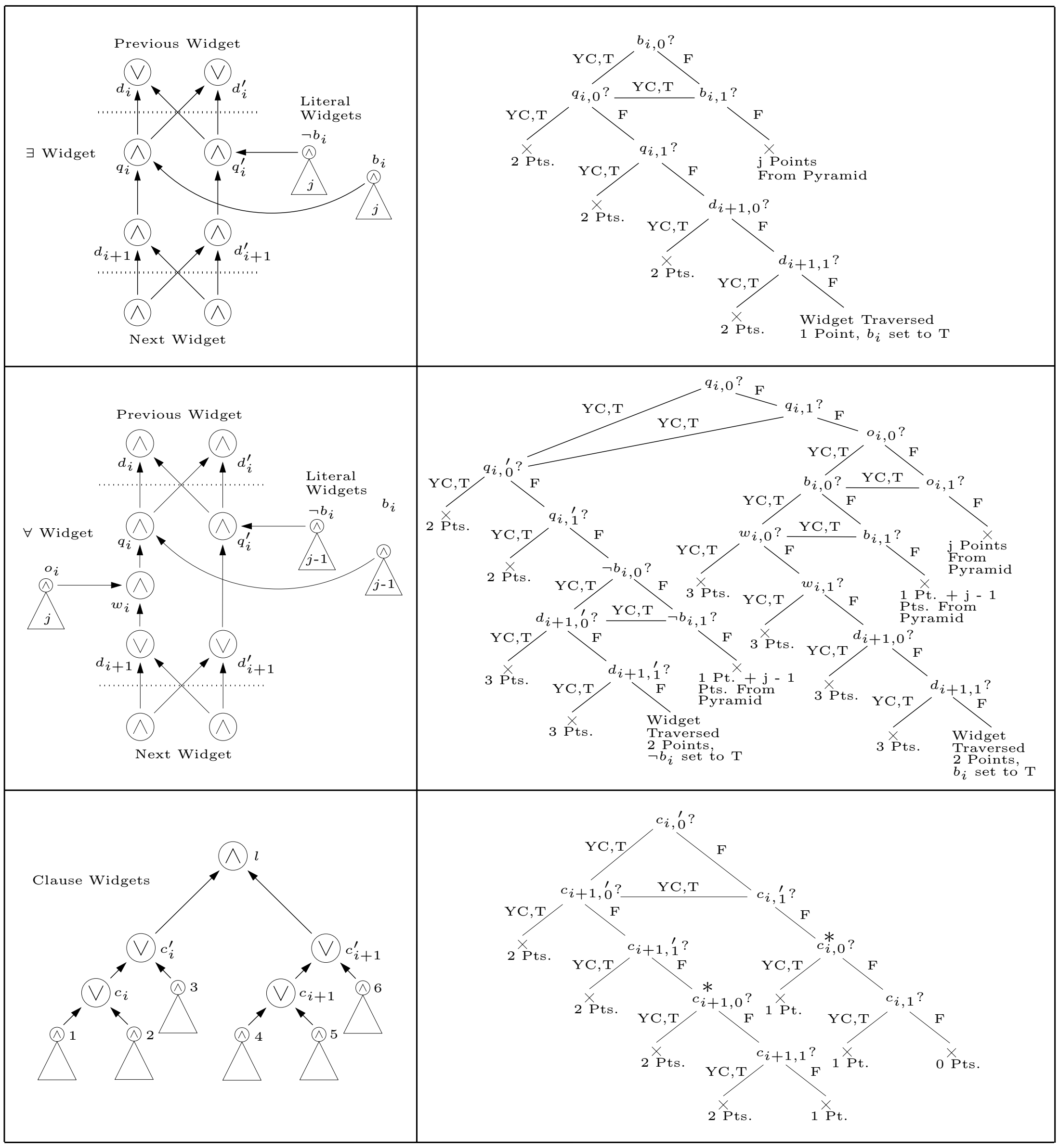

Figure 2: The different widgets from Lingas's construction together with the decision trees showing the prover's strategies for traversing them. Duplicate subtrees are indicated by having multiple parents.

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, and shall show how to force a contradiction given a conjunction pyramid leaf l that has had both of its associated variables set to False. This strategy is shown in Figure 2.

A key observation to make is that it is possible for the Prover to traverse the quantifier widgets such that the literal widgets that 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 that has been set to True. This ensures that the Prover's strategy given by the decision tree in Figure 2 will always work. 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 and universal widgets can be traversed while only giving up 1 and 2 points respectively, the conjunctive pyramid can be traversed while only giving up $M - 2$ points, and 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, as required. □

4.4 Delayer Strategy for Monotone Circuits

We now show that for any binary monotone circuit C, when playing the P/D game on $Peb^2(C)$, the Delayer has a strategy that will always win at least $B\text{-}Peb(C)$ points. In [4] the authors give a Delayer strategy that 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 that is guaranteed to win at least $B\text{-}Peb(G)$ points, defined with sliding.

In [4], 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 $\text{Peb}(C, S, T, P)$, 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'.

The Delayer's strategy proceeds as follows: at the start of the game, P is empty, 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 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.

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

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

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

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

In the second stage of the Delayer's strategy, 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 (i.e. $P' := P \cup \{v\}$). If v has a pebble on it, then set the queried variable the value True.

Case 2b: If $v \in T'$, then set the queried variable to False, 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 propagation clause associated with a vertex v and its immediate predecessors u_1, u_2 if 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 False, from which it follows that $v \in T$, by Case 2a. If u_i is one of the immediate predecessors of v, one of the two variables associated with u_i must be set to True. By Case 2b, 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_i to v, showing that $\text{Peb}(C, S, T, P) = 0$. 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 Peb$(C, S, T, P) \leq$ $\max\{$Peb$(C, S, T \cup \{v\}, P)$, 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 $\text{Peb}(G, S, T \cup \{v\}, P)$ pebbles. If the result is a pebble in T, then we are through, otherwise the final configuration has a pebble on v. Simply remove the other pebbles, then pebble T from $S \cup \{v\}$; this final step uses a total of $\text{Peb}(C, S \cup \{v\}, T, P \cup \{v\}) + 1$ 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 at least 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 $\text{Peb}(C, S, T, P) = \text{Peb}(C, S', T', P')$, then there is nothing to prove, so we can assume that $\text{Peb}(C, S, T, P) > \text{Peb}(C, S', T', P')$. By Lemma 4.8, $\text{Peb}(C, S, T, P) = \text{Peb}(C, S', T', P') + 1$. 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 v_i. If v were in T', or if v had a pebble on it at the start of the round, then $\text{Peb}(C, S, T, P) = \text{Peb}(C, S', T', P')$. 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 also holds for the supertree. □

Corollary 4.10. *For any monotone circuit C, when playing on the formula Peb$^2(C)$, the Delayer has a strategy that wins at least B-Peb(C) points.*

4.5 $\mathcal{PSPACE}$-Completeness of TCSP & P/D Game

This section contains our main result, namely the $\mathcal{PSPACE}$-Completeness of the T-RES clause space problem as well as the P/D game. The T-RES clause space problem ($TCSP$) asks: 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 ($PDGAME$) asks: Given a formula F and integer k, is $PD(F)$ at most k?

The results from the previous sections allow us to prove the $\mathcal{PSPACE}$-Completeness of $TCSP$ and $PDGAME$ via a straightforward reduction from the $\mathcal{PSPACE}$-Complete problem of pebbling Lingas circuits. However, before proving this result we must first give a $\mathcal{PSPACE}$ algorithm for them these problems which uses following Lemma:

Lemma 4.11. *Every unsatisfiable CNF formula F has a* T-RES *refutation with clause space at most $n + 1$, where n is the number of distinct variables in F.*

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 that 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, $TCS(F \vdash_{\text{T-RES}} \emptyset) \leq n+1$, as required. $\square$

Lemma 4.12. *$TCSP \in \mathcal{PSPACE}$ and $PDGAME \in \mathcal{PSPACE}$*

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 4.11 we simply accept. Otherwise F is unsatisfiable and $k \leq n$, so if F has a (configuration style) T-RES refutation π with $TCS(F \vdash_{\text{T-RES}} \emptyset) \leq k$, then each configuration contains at most k clauses. Since $k \leq n$ and 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: Using 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. At any time, there are only two configurations in memory, but since each configuration uses at most quadratic space, our computation is in $\mathcal{NPSPACE}$. Finally, we appeal to Savitch's theorem to show that the problem of determining whether or not F has T-RES refutation π with $TCS(F \vdash_{\text{T-RES}} \emptyset) \leq k$ is in $\mathcal{PSPACE}$, thereby completing our $\mathcal{PSPACE}$ algorithm. By Theorem 3.1, this immediately implies that $PDGAME \in \mathcal{PSPACE}$ as well. $\square$

We are now ready to prove our main result:

Theorem 4.13. *$PDGAME$ and $TCSP$ are both $\mathcal{PSPACE}$-Complete.*

Proof: By Lemma 4.12, we know that $PDGAME \in \mathcal{PSPACE}$. Next we show that $PDGAME$ is $\mathcal{PSPACE}$-Hard by reducing from black pebbling the Lingas circuits $\mathcal{C}_{\mathcal{L}}$, which proceeds by simply taking the input (C, k), and outputting $(Peb^2(C), k)$. Clearly this is a polytime reduction, and it is easy to see that it is correct by showing that $(C, k) \in \mathcal{C}_{\mathcal{L}}$ if and only if $(Peb^2(C), k) \in PDGAME$: If $(C, k) \in \mathcal{C}_{\mathcal{L}}$, then $B\text{-}Peb(C) \leq k$, so by Theorem 4.6, $PD(Peb^2(C)) \leq k$, and $(Peb^2(C), k) \in PDGAME$. On the other hand, if $(C, k) \notin \mathcal{C}_{\mathcal{L}}$, then $B\text{-}Peb(C) > k$, so by Corollary 4.10, $PD(Peb^2(C)) > k$, and $(Peb^2(C), k) \notin PDGAME$.

Therefore $PDGAME$ is $\mathcal{PSPACE}$-Complete, and by Theorem 3.1 it is easy to design a $\mathcal{PSPACE}$-Hardness reduction from $PDGAME$ to $TCSP$ as well. $\square$

References

[1] M. Alekhnovich, E. Ben-Sasson, A.A. Razborov, and A. Wigderson. Space Complexity in Propositional Calculus. *SIAM J. of Comp.*, V.31,No.4:1184 – 1211, 2001.

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

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

[4] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near Optimal Separation of Tree-like and General Resolution. *Combinatorica*, Vol. 24, No. 4:585 – 604, 2004.

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

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

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

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

[9] J. R. Gilbert, T. Lengauer, and R. E. Tarjan. The Pebbling Problem is Complete in Polynomial Space. *SIAM Journal of Computing*, Vol. 9, No. 3:513 – 524, 1980.

[10] 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.

[11] 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.

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