DD3501 Current Research in Proof Complexity Nov 12, 2011

Notes on Pebble Games and Pebbling Contradictions

Lecturer: Jakob Nordström *Scribe: Jakob Nordström*

Towards the end of lecture 4, we started talking about pebble games and pebbling contradictions. The presentation was somewhat rushed, however, and despite this we did not quite cover what was intended. As a complement to what was said during the lecture, and as reference material for future lectures, here are some more properly organized notes.

Section 1 defines pebbling and state some fundamental results; Section 2 introduces pebbling contradiction CNF formulas, and in Section 3 we discuss connections between pebblings of graphs and resolution refutations of pebbling contradictions. In Section 4, we prove a strong trade-off between width and clause space. Finally, Section 5 presents some important concepts which we have not discussed so far but to which we will return later in the course.

1 Pebble Games

Pebbling is a tool for studying time-space relationships by means of a game played on directed acyclic graphs (DAGs). This game models computations where the execution is independent of the input and can be performed by straight-line programs. Each such program is encoded as a graph, and a pebble on a vertex in the graph indicates that the corresponding value is currently kept in memory. The goal is to pebble the output vertex of the graph with minimal number of pebbles (amount of memory) and steps (amount of time).

Pebble games were originally devised for studying programming languages and compiler construction, but have found a broad range of applications in computational complexity theory. An excellent survey of pebbling up to ca 1980 is [Pip80], and another in-depth treatment of some pebbling-related questions can be found in [Sav98, Chapter 10]. Some more recent developments are covered in the upcoming survey [Nor12] (which will be finished any decade now).

The *pebbling price* of a DAG G in the black pebble game captures the memory space, or number of registers, required to perform the deterministic computation described by G. We will mainly be interested in the the more general *black-white pebble game* modelling nondeterministic computation, which was introduced in [CS76]. In what follows, we refer to vertices having indegree 0 as *sources* and vertices having outdegree 0 as *sinks*. We write $pred(v)$ to denote the immediate predecessors of a vertex v, i.e., all vertices u which have an edge to v.

Definition 1.1 (Black-white pebble game). Let G be a directed acyclic graph (DAG) with a unique sink vertex z. The *black-white pebble game* on G is the following one-player game. At any time t, we have a *pebble configuration* $\mathbb{P}_t = (B_t, W_t)$ of black pebbles B_t and white pebbles W_t on the vertices of G, where $B_t, W_t \subseteq V(G)$ and $B_t \cap W_t = \emptyset$. A *(complete) black-white pebbling* of G, or a *black-white pebbling strategy* for G, is a sequence of pebble configurations $\mathcal{P} = \{\mathbb{P}_0, \mathbb{P}_1, \ldots, \mathbb{P}_\tau\}$ such that $\mathbb{P}_0 = (\emptyset, \emptyset)$, $\mathbb{P}_\tau = (\{z\}, \emptyset)$, and for all $t \in [\tau]$ it holds that $\mathbb{P}_t$ is obtained from $\mathbb{P}_{t-1}$ by one of the following rules:

1. **Black pebble placement on v:** A black pebble may be placed on v provided that v is empty and all immediate predecessors of v are covered by pebbles. More formally, letting $B_t = B_{t-1} \cup \{v\}$ and $W_t = W_{t-1}$ is allowed if $v \notin B_{t-1} \cup W_{t-1}$ and $pred(v) \subseteq B_{t-1} \cup W_{t-1}$. (In particular, a black pebble can always be placed on a source vertex s since $pred(s) = \emptyset$.)

2. **Black pebble removal from v:** A black pebble may be removed from any vertex at any time. Formally, if $v \in B_{t-1}$, then we can set $B_t = B_{t-1} \setminus \{v\}$ and $W_t = W_{t-1}$.

3. **White pebble placement on v:** A white pebble may be placed on any empty vertex at any time. Formally, if $v \notin B_{t-1} \cup W_{t-1}$, then we can set $B_t = B_{t-1}$ and $W_t = W_{t-1} \cup \{v\}$.

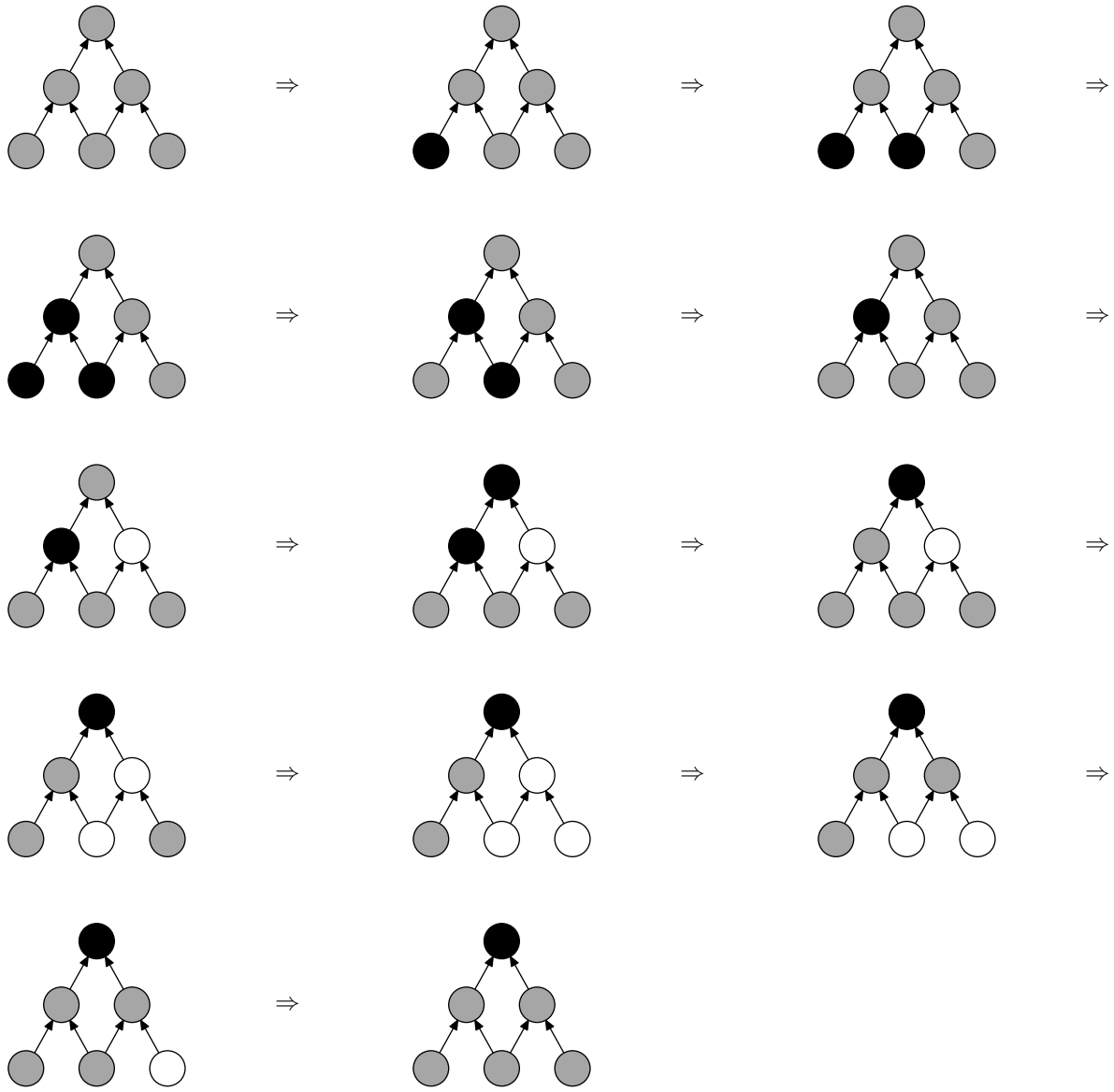


Figure 1: Complete black-white pebbling of pyramid of height 2.

4. **White pebble removal from v:** If all immediate predecessors of a white-pebbled vertex v have pebbles on them, the white pebble on v may be removed. In particular, a white pebble can always be removed from a source vertex. Formally, letting $B_t = B_{t-1}$ and $W_t = W_{t-1} \setminus \{v\}$ is allowed if $v \in W_{t-1}$ and $pred(v) \subseteq B_{t-1} \cup W_{t-1}$.

A *black pebbling* of G is a pebbling using black pebbles only, i.e., having $W_t = \emptyset$ for all t.

Figure 1 depicts the sequence of pebbling moves in an example black-white pebbling of a small graph.

We are interested in measuring the time and space needed to pebble graphs. Time in isolation is not so interesting, since any DAG with $|V(G)| = n$ vertices can be pebbled in time $O(n)$—just sort the vertices in topological order and then black-pebble the vertices in this order to get a pebbling in linear time and space. However, the minimal space needed can be much less than linear, and if one wants to optimize time and space simultaneously there can be fairly nontrivial relations between the two measures. We will discuss this in more detail later in the course.

The next definition makes the concepts of time and space in pebble games more precise and introduces some notation that we will use.

Definition 1.2 (Pebbling time and space). The *time* of a pebbling $\mathcal{P} = \{\mathbb{P}_0, \ldots, \mathbb{P}_\tau\}$ is simply $\mathsf{time}(\mathcal{P}) = \tau$ and the *space* is $\mathsf{space}(\mathcal{P}) = \max_{0 \le t \le \tau}\{|B_t \cup W_t|\}$. We say that G can be pebbled in *simultaneous time* τ *and space* s if there is a complete pebbling $\mathcal{P}$ with $\mathsf{time}(\mathcal{P}) \le \tau$ and $\mathsf{space}(\mathcal{P}) \le s$.

The *black-white pebbling price* (also known as the *pebbling measure* or *pebbling number*) of G, denoted $\mathsf{BW\text{-}Peb}(G)$, is the minimum space of any complete pebbling of G. The *(black) pebbling price* of G, denoted $\mathsf{Peb}(G)$, is the minimum space of any complete black pebbling of G.

Looking at our example pebbling $\mathcal{P}$ in Figure 1 again, it is easy to check that it has $\mathsf{time}(\mathcal{P}) = 13$ and $\mathsf{space}(\mathcal{P}) = 4$. To get a feel for how to prove pebbling space bounds, and to see why it is not entirely trivial, it might be a good exercise to show that space 4 is optimal for this graph.

A classic result in pebbling, and indeed in complexity theory, is that if a graph G as bounded indegree, then it is always possible to save a logarithmic factor over the trivial linear upper bound on space. And you do not need white pebbles for this—there is such a pebbling using black pebbles only.

Theorem 1.3 ([HPV77]). *For any DAG G of size $\Theta(n)$ and constant fan-in it holds that $\mathsf{Peb}(G) = \mathrm{O}(n/\log n)$ (where the constant hidden in the big-oh notation depends on how large the fan-in is).*

An almost equally classic result is that this is optimal. In the worst case, you can never save more than a logarithmic factor.

Theorem 1.4 ([GT78]). *There are explicitly constructible DAGs G_n of size $\Theta(n)$ and fan-in 2 such that $\mathsf{BW\text{-}Peb}(G_n) = \Omega(n/\log n)$.*

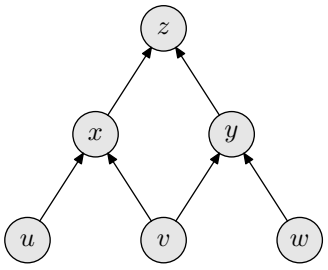
A nice feature of this result, as stated in the theorem, is that these graphs are *explicit*. That means that we do not need to use, for instance, the probabilistic method to argue that we just know that such graphs exist but do not know more about them, but that we can in fact give an algorithm that constructs such graphs in an efficient manner (i.e., polynomial time). We will not go into details of the construction here, since it is somewhat elaborate, but the interested reader can refer to [Nor12, Section 7] to see what these graphs look like and how the lower bound is proven.

2 Pebbling Contradictions

In the last decade, there has been renewed interest in pebbling in the context of proof complexity. The way pebbling results have been used in proof complexity has mainly been by studying so-called *pebbling contradictions* (also known as *pebbling tautologies*[1] or *pebbling formulas*). These are CNF formulas encoding the pebble game played on a DAG G by postulating the sources to be true and the sink to be false, and specifying that truth propagates through the graph according to the pebbling rules. The idea to use such formulas seems to have appeared for the first time in Kozen [Koz77], and they were also studied in [RM99, BEGJ00] before being defined in full generality by Ben-Sasson and Wigderson in [BW01].

Definition 2.1 (Pebbling contradiction). Suppose that G is a DAG with sources S and a unique sink z. Identify every vertex $v \in V(G)$ with a propositional logic variable v. The *pebbling contradiction* over G, denoted Peb_G, is the conjunction of the following clauses:

[1] Recall that we warned already during the first lecture for this sometimes confusing convention in proof complexity to consider contradictory CNF formulas to be tautologies (since they are used to encode negations of tautological statements).



(a) Pyramid graph Π_2 of height 2.

$$
\begin{aligned}
& u \\
\wedge\ & v \\
\wedge\ & w \\
\wedge\ & (\overline{u} \vee \overline{v} \vee x) \\
\wedge\ & (\overline{v} \vee \overline{w} \vee y) \\
\wedge\ & (\overline{x} \vee \overline{y} \vee z) \\
\wedge\ & \overline{z}
\end{aligned}
$$

(b) Pebbling contradiction Peb_{Π_2}.

Figure 2: Pebbling contradiction for the pyramid graph Π_2.

- for all $s \in S$, a unit clause s (*source axioms*),
- For all non-source vertices v, the clause $\bigvee_{u \in pred(v)} \overline{u} \vee v$ (*pebbling axioms*),
- for the sink z, the unit clause $\overline{z}$ (*sink axiom*).

If G has n vertices and maximal indegree ℓ, the formula Peb_G is a minimally unsatisfiable $(1+\ell)$-CNF formula with $n+1$ clauses over n variables. We will almost exclusively be interested in dags with bounded indegree $\ell = \mathrm{O}(1)$, usually $\ell = 2$. We note that DAGs with fan-in 2 and a single sink have sometimes been referred to as *circuits* in the proof complexity literature, although we will not use that terminology in this course. For an example of a pebbling contradiction, see the CNF formula in Figure 2(b) defined in terms of the graph in Figure 2(a).

3 Reductions Between Pebbling and Resolution

We mentioned towards the end of lecture 4 that any black pebbling of a graph can be simulated by a resolution refutation of the corresponding pebbling contradiction. The formal statement is as follows.

Observation 3.1 ([BIW04]). *If G is a DAG with constant fan-in and a single sink, then from any complete black-only pebbling $\mathcal{P}$ of G one can extract a resolution refutation $\pi : Peb_G \vdash \bot$ of the pebbling contradiction over G such that $L(\pi) = \mathrm{O}(\mathsf{time}(\mathcal{P}))$ and $TotSp(\pi) = \mathrm{O}(\mathsf{space}(\mathcal{P}))$.*

Proof. We omit the proof as it is part of problem set 1, but remark that it is fairly straightforward. □

The other direction is much less obvious, but essentially says that any resolution refutation of a pebbling contradiction can be simulated by a black-white pebbling of the underlying graph.

Theorem 3.2 ([Ben09]). *If G is a DAG with constant fan-in and a single sink, then from any resolution refutation $\pi : Peb_G \vdash \bot$ of the pebbling contradiction over G one can extract a complete black-white pebbling $\mathcal{P}_\pi$ of G such that $\mathsf{time}(\mathcal{P}_\pi) = \mathrm{O}(L(\pi))$ and $\mathsf{space}(\mathcal{P}_\pi) = \mathrm{O}(TotSp(\pi))$.*

Proof sketch. This is also part of problem set 1, but we dropped some heavy hints in class, as well as in the problem statement in the problem set. So here they are again.

The general idea is to let positive literals correspond to black pebbles and negative literals to white pebbles. Using this correspondence, one gets a pebble configuration from every clause configuration. Then one needs to glue these pebble configurations together in such a way that one obtains a correct pebbling for which the bound stated above holds.

There are two helpful technical observations that one needs to prove (although in the problem set it is allowed to use them without proof). Namely, without loss of generality one can make the following assumptions:

- Any clause appearing in any configuration of the proof (other than the final empty clause) is resolved over at least once before being erased.
- When a clause is erased from a configuration after having been used in a resolution inference for the last time, it is erased immediately after this final resolution inference step (or, if both clauses used for the inference are erased, then they are erased immediately after one another in any order you prefer).

The rest is just figuring out how to put the pieces together. Note that the hard part is to show that you really get a correct pebbling strategy for the graph, and this needs to be argued carefully. □

4 A Trade-off Between Width and Clause Space in Resolution

When we showed in lecture 3 that clause space is an upper bound on width in resolution, we observed that the proof of this fact it [AD08] worked by taking a space-efficient resolution refutation π and and transforming it into a potentially completely different narrow resolution refutation π' (namely, by first using π to derive a good strategy for the Spoiler in the combinatorial game characterizing width, and then using the Spoiler strategy to obtain a refutation π' in small width). It is natural to ask whether such a transformation is really necessary. Maybe one can prove that a space-efficient refutation is also narrow simply by virtue of having small space? Or maybe a refutation in small space can at least be massaged into one that has small width also without blowing up the space?

It turns out that this question had been answered even before we knew there was this reason to ask it, namely in the conference paper [Ben02] preceding [AD03]. (Both of these papers took quite a while before being published as the journal versions [AD08] and [Ben09] in reverse chronological order.) The answer is that in general it is impossible to optimize space and width simultaneously in any meaningful way.

Theorem 4.1 ([Ben09]). *There is a family of k-CNF formulas F_n of size $\Theta(n)$ such that $Sp(F_n \vdash \bot) = \mathrm{O}(1)$ and $W(F_n \vdash \bot) = \mathrm{O}(1)$, but for any resolution refutation $\pi_n : F_n \vdash \bot$ it holds that $Sp(\pi) \cdot W(\pi) = \Omega(n/\log n)$.*

What this theorem says is that although the formulas F_n can be refuted in essentially minimal clause space and essentially minimal width, when we optimze one of these measures the other has to blow up to almost worst possible. (Recall that the worst-case upper bound is linear in n, and the lower bound we get here is linear except for a $\log n$ factor.)

We will spend the rest of this section proving this theorem. Let us start by an easy observation.

Observation 4.2. *For any resolution refutation π it holds that $Sp(\pi) \cdot W(\pi) \geq TotSp(\pi)$.*

Proof. The refutation π never has more that $Sp(\pi)$ clauses in memory, and each clause has size at most $W(\pi)$. Thus the total number of literals in memory at any point during π, i.e., the total space, is at most $Sp(\pi) \cdot W(\pi)$. □

Now we can present the formulas we want to use to prove Theorem 4.1. Namely, we take the graphs G_n from Theorem 1.4, which are very hard with respect to pebbling space, and look at pebbling contradictions over these graphs. Using Theorem 3.2, we can conclude that for these formulas it must hold that $TotSp(Peb_{G_n} \vdash \bot) = \Omega(n/\log n)$.

All that remains now is to prove that these formulas are easy with respect to both width and clause space. But this is in fact the case for any pebbling contradiction, as stated next.

Observation 4.3. *For any DAG G of size $\Theta(n)$ and constant fan-in there is a resolution refutation $\pi : Peb_G \vdash \bot$ with $L(\pi) = \mathrm{O}(n)$ and $W(\pi) = \mathrm{O}(1)$.*

Lemma 4.4 ([Ben09]). *For any DAG G of size $\Theta(n)$ there is a refutation $\pi : Peb_G \vdash \bot$ with $L(\pi) = \mathrm{O}(n)$ and $Sp(\pi) = \mathrm{O}(1)$.*

Before proving Observation 4.3 and Lemma 4.4, let us just note that if we can do so then we are done also with Theorem 4.1. We already argued above that $TotSp(Peb_{G_n} \vdash \bot) = \Omega(n/\log n)$. This means that although we can refute the formulas Peb_{G_n} in width $W(Peb_{G_n} \vdash \bot) = \mathrm{O}(1)$ by Observation 4.3 and also in clause space $Sp(Peb_{G_n} \vdash \bot) = \mathrm{O}(1)$ by Lemma 4.4, any refutation π optimizing one of the measures must have the other measure being large.

Proof of Observation 4.3. Suppose that G is a DAG with indegree ℓ and unique sink z. Let $n = |V(G)|$. Sort the vertices of G in topological order. We will show how to derive the unit clause v for all vertices $v \in V(G)$ in this topological order in length $n(1+\ell)$ and width $1+\ell$. Once we derive z, we then resolve with $\overline{z}$ to get the empty clause. Since ℓ is assumed to be constant, this is sufficient to prove the observation.

The argument is by induction. If s is a source vertex, then the unit clause s is an axiom of Peb_G and there is nothing to prove. If u is a non-source, say with immediate predecessors $pred(u) = \{v_1, \ldots, v_{\ell'}\}$ for $\ell' \leq \ell$, then by induction we have already derived all the unit clauses v_i for $i = 1, \ldots, \ell'$. Download the axiom $\overline{v}_1 \vee \cdots \vee \overline{v}_{\ell'} \vee u$ and resolve in ℓ' steps with v_i, $i = 1, \ldots, \ell'$, to derive u. This is a total of at most $1+\ell$ steps per vertex, and the widest clauses that appear in the refutation are the axiom clauses. □

Proof of Lemma 4.4. Suppose again that G is a DAG with unique sink z and number of vertices $n = |V(G)|$, but this time sort the vertices of G in *reverse* topological order. We will consider the vertices in this order $v_1 = z, v_2, \ldots, v_{n-1}, v_n$ and derive in constant clause space (in fact, minimal clause space 3) for each v_i a clause $D_i = \bigvee_{w \in W_i} \overline{w}$ such that $W_i \subseteq \{v_{i+1}, \ldots, v_n\}$. When we reach v_n, this means we have derived the empty clause.

Start by downloading the sink axiom $\overline{z}$ as well as the pebbling axiom $\bigvee_{w \in pred(z)} \overline{w} \vee z$ for the sink, and resolve these two clauses to get $D_1 = \bigvee_{w \in pred(z)} \overline{w}$. This clause clearly satisfies the invariant specified above for $W_1 = pred(z)$.

When we get to the vertex v_i, the clause $D_{i-1} = \bigvee_{w \in W_{i-1}} \overline{w}$ must contain the literal $\overline{v}_i$ since v_i is the predecessor of some $v_{i'}$ for $i' < i$, and the literal $\overline{v}_i$ was added to the clause when the pebbling axiom for $v_{i'}$ was downloaded. Now download the pebbling axiom $\bigvee_{w \in pred(z)} \overline{w} \vee v_i$ for v_i and resolve with D_{i-1} to get a new clause $D_i = \bigvee_{w \in W_i} \overline{w}$ for $W_i = (W_{i-1} \cup pred(v_i)) \setminus \{v_i\}$, and then erase everything except D_i from memory. Since the resolution step removed the literal $\overline{v_i}$ and all new literals added must be for vertices before v_i in the topological ordering, the invariant is maintained. Since the derivation works by always keeping one clause in memory and downloading axioms and resolving with this clause, the space of the derivation is 3. We make exactly one resolution inference (plus two erasures) per vertex, so the length of the derivation is linear in the number of vertices (independent of the fan-in of the graph). The lemma now follows by the induction principle. □

5 Generalized Pebbling Contradictions

In some cases when we will want to use pebbling to establish proof complexity results, the formulas in Definition 2.1 are not quite sufficient for our purposes since they are a bit too easy to refute. We therefore want to make them (moderately) harder, and it turns out that a good way of achieving this is to substitute some suitable Boolean function $f(x_1, \ldots, x_d)$ for each variable x and expand to get a new CNF formula.

It will be useful to formalize this concept of substitution for any CNF formula F and any Boolean function f. To this end, let f_d denote any (non-constant) Boolean function

$f_d : \{0,1\}^d \mapsto \{0,1\}$ of arity d. We use the shorthand $\vec{x} = (x_1, \ldots, x_d)$, so that $f_d(\vec{x})$ is just an equivalent way of writing $f_d(x_1, \ldots, x_d)$. Every function $f_d(x_1, \ldots, x_d)$ is equivalent to a CNF formula over $x_1, \ldots, x_d$ with at most 2^d clauses. Fix some canonical set of clauses representing f_d and let $Cl[\neg f_d(\vec{x})]$ denote the clauses in some chosen canonical representation of the negation of f_d. This canonical representation can be given by a formal definition (in terms of min- and maxterms), but we do not want to get too formal here and instead try to convey the intuition by providing a few examples. For instance, we have

$$Cl[\vee_2(\vec{x})] = \{x_1 \vee x_2\} \quad \text{and} \quad Cl[\neg\vee_2(\vec{x})] = \{\overline{x}_1, \overline{x}_2\} \tag{5.1}$$

for logical or of two variables and

$$Cl[\oplus_2(\vec{x})] = \{x_1 \vee x_2, \overline{x}_1 \vee \overline{x}_2\} \quad \text{and} \quad Cl[\neg\oplus_2(\vec{x})] = \{x_1 \vee \overline{x}_2, \overline{x}_1 \vee x_2\} \tag{5.2}$$

for exclusive or of two variables. If we let thr_d^k denote the threshold function saying that k out of d variables are true, then for thr_4^2 we have

$$Cl[thr_4^2(\vec{x})] = \left\{ \begin{array}{l} x_1 \vee x_2 \vee x_3, \\ x_1 \vee x_2 \vee x_4, \\ x_1 \vee x_3 \vee x_4, \\ x_2 \vee x_3 \vee x_4 \end{array} \right\} \quad \text{and} \quad Cl[\neg thr_4^2(\vec{x})] = \left\{ \begin{array}{l} \overline{x}_1 \vee \overline{x}_2, \\ \overline{x}_1 \vee \overline{x}_3, \\ \overline{x}_1 \vee \overline{x}_4, \\ \overline{x}_2 \vee \overline{x}_3, \\ \overline{x}_2 \vee \overline{x}_4, \\ \overline{x}_3 \vee \overline{x}_4 \end{array} \right\} . \tag{5.3}$$

The following observation is rather immediate, but nevertheless it might be helpful to state it explicitly.

Observation 5.1. *Suppose for any non-constant Boolean function f_d that $C \in Cl[f_d(\vec{x})]$ and that ρ is any partial truth value assignment such that $\rho(C) = 0$. Then for all $D \in Cl[\neg f_d(\vec{x})]$ it holds that $\rho(D) = 1$.*

Proof. If $\rho(C) = 0$ this means that $\rho(f_d) = 0$. Then clearly $\rho(\neg f_d) = 1$, so, in particular, ρ must fix all clauses $D \in Cl[\neg f_d(\vec{x})]$ to true. □

We want to define formally what it means to substitute f_d for the variables $Vars(F)$ in a CNF formula F. For notational convenience, we assume that F only has variables x, y, z, et cetera, without subscripts, so that $x_1, \ldots, x_d$, $y_1, \ldots, y_d$, $z_1, \ldots, z_d, \ldots$ are new variables not occurring in F.

Definition 5.2 (Substitution formula). For a positive literal x and a non-constant Boolean function f_d, we define the *f_d-substitution* of x to be $x[f_d] = Cl[f_d(\vec{x})]$, i.e., the canonical representation of $f_d(x_1, \ldots, x_d)$ as a CNF formula. For a negative literal $\neg y$, the f_d-substitution is $\neg y[f_d] = Cl[\neg f_d(\vec{y})]$. The f_d-substitution of a clause $C = a_1 \vee \cdots \vee a_k$ is the CNF formula

$$C[f_d] = \bigwedge_{C_1 \in a_1[f_d]} \cdots \bigwedge_{C_k \in a_k[f_d]} (C_1 \vee \ldots \vee C_k) \tag{5.4}$$

and the f_d-substitution of a CNF formula F is $F[f_d] = \bigwedge_{C \in F} C[f_d]$.

For example, for the clause $C = x \vee \overline{y}$ and the exclusive or function $f_2 = x_1 \oplus x_2$ we have

$$\begin{aligned} C[f_2] = \quad & (x_1 \vee x_2 \vee y_1 \vee \overline{y}_2) \wedge (x_1 \vee x_2 \vee \overline{y}_1 \vee y_2) \\ & \wedge (\overline{x}_1 \vee \overline{x}_2 \vee y_1 \vee \overline{y}_2) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee y_2) \ . \end{aligned} \tag{5.5}$$

$$
\begin{aligned}
&(u_1 \vee u_2) \\
\wedge\ &(v_1 \vee v_2) \\
\wedge\ &(w_1 \vee w_2) \\
\wedge\ &(\overline{u}_1 \vee \overline{v}_1 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_1 \vee \overline{v}_2 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_2 \vee \overline{v}_1 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_2 \vee \overline{v}_2 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{v}_1 \vee \overline{w}_1 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_1 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_2 \vee \overline{w}_1 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_2 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{x}_1 \vee \overline{y}_1 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_1 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_2 \vee \overline{y}_1 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_2 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ &\overline{z}_1 \\
\wedge\ &\overline{z}_2
\end{aligned}
$$

(a) Substitution pebbling contradiction $Peb_{\Pi_2}[\vee_2]$ with respect to binary logical or.

$$
\begin{aligned}
&(u_1 \vee u_2) \\
\wedge\ &(\overline{u}_1 \vee \overline{u}_2) \\
\wedge\ &(v_1 \vee v_2) \\
\wedge\ &(\overline{v}_1 \vee \overline{v}_2) \\
\wedge\ &(w_1 \vee w_2) \\
\wedge\ &(\overline{w}_1 \vee \overline{w}_2) \\
\wedge\ &(u_1 \vee \overline{u}_2 \vee v_1 \vee \overline{v}_2 \vee x_1 \vee x_2) \\
\wedge\ &(u_1 \vee \overline{u}_2 \vee v_1 \vee \overline{v}_2 \vee \overline{x}_1 \vee \overline{x}_2) \\
\wedge\ &(u_1 \vee \overline{u}_2 \vee \overline{v}_1 \vee v_2 \vee x_1 \vee x_2) \\
\wedge\ &(u_1 \vee \overline{u}_2 \vee \overline{v}_1 \vee v_2 \vee \overline{x}_1 \vee \overline{x}_2) \\
\wedge\ &(\overline{u}_1 \vee u_2 \vee v_1 \vee \overline{v}_2 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_1 \vee u_2 \vee v_1 \vee \overline{v}_2 \vee \overline{x}_1 \vee \overline{x}_2) \\
\wedge\ &(\overline{u}_1 \vee u_2 \vee \overline{v}_1 \vee v_2 \vee x_1 \vee x_2) \\
\wedge\ &(\overline{u}_1 \vee u_2 \vee \overline{v}_1 \vee v_2 \vee \overline{x}_1 \vee \overline{x}_2) \\
\wedge\ &(v_1 \vee \overline{v}_2 \vee w_1 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ &(v_1 \vee \overline{v}_2 \vee w_1 \vee \overline{w}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ &(v_1 \vee \overline{v}_2 \vee \overline{w}_1 \vee w_2 \vee y_1 \vee y_2) \\
\wedge\ &(v_1 \vee \overline{v}_2 \vee \overline{w}_1 \vee w_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ &(\overline{v}_1 \vee v_2 \vee w_1 \vee \overline{w}_2 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_1 \vee v_2 \vee w_1 \vee \overline{w}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ &(\overline{v}_1 \vee v_2 \vee \overline{w}_1 \vee w_2 \vee y_1 \vee y_2) \\
\wedge\ &(\overline{v}_1 \vee v_2 \vee \overline{w}_1 \vee w_2 \vee \overline{y}_1 \vee \overline{y}_2) \\
\wedge\ &(x_1 \vee \overline{x}_2 \vee y_1 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ &(x_1 \vee \overline{x}_2 \vee y_1 \vee \overline{y}_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ &(x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee y_2 \vee z_1 \vee z_2) \\
\wedge\ &(x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee y_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ &(\overline{x}_1 \vee x_2 \vee y_1 \vee \overline{y}_2 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_1 \vee x_2 \vee y_1 \vee \overline{y}_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ &(\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee y_2 \vee z_1 \vee z_2) \\
\wedge\ &(\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee y_2 \vee \overline{z}_1 \vee \overline{z}_2) \\
\wedge\ &(z_1 \vee \overline{z}_2) \\
\wedge\ &(\overline{z}_1 \vee z_2)
\end{aligned}
$$

(b) Substitution pebbling contradiction $Peb_{\Pi_2}[\oplus_2]$ with respect to binary exclusive or.

Figure 3: Examples of substitution pebbling formulas for the pyramid graph Π_2.

Note that $F[f_d]$ is a CNF formula over $d \cdot |Vars(F)|$ variables containing strictly less than $|F| \cdot 2^{d \cdot W(F)}$ clauses. (Recall that we defined a CNF formula as a set of clauses, which means that $|F|$ is the number of clauses in F.) It is easy to verify that $F[f_d]$ is unsatisfiable if and only if F is unsatisfiable.

Two examples of substituted version of the pebbling formula in Figure 2(b) are the substitution with logical or in Figure 3(a) and with exclusive or in Figure 3(b). As we shall see later in the course, these formulas have played an important role in the line of research trying to understand proof space in resolution.

Although we will not prove it here, it can be shown (and we will need it later) that Observation 3.1 in fact extends to generalized pebbling contradictions $Peb_G[f_d]$ (although the hidden constant factors will then depend on the specific substitution function f_d chosen).

References

[AD03] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. In *Proceedings of the 18th IEEE Annual Conference on Computational Complexity (CCC '03)*, pages 239–247, July 2003.

[AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version appeared in *CCC '03*.

[BEGJ00] Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal on Computing*, 30(5):1462–1484, 2000. Preliminary version appeared in *FOCS '98*.

[Ben02] Eli Ben-Sasson. Size space tradeoffs for resolution. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 457–464, May 2002.

[Ben09] Eli Ben-Sasson. Size space tradeoffs for resolution. *SIAM Journal on Computing*, 38(6):2511–2525, May 2009. Preliminary version appeared in *STOC '02*.

[BIW04] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near optimal separation of treelike and general resolution. *Combinatorica*, 24(4):585–603, September 2004.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version appeared in *STOC '99*.

[CS76] Stephen A. Cook and Ravi Sethi. Storage requirements for deterministic polynomial time recognizable languages. *Journal of Computer and System Sciences*, 13(1):25–37, 1976.

[GT78] John R. Gilbert and Robert Endre Tarjan. Variations of a pebble game on graphs. Technical Report STAN-CS-78-661, Stanford University, 1978. Available at `http://infolab.stanford.edu/TR/CS-TR-78-661.html`.

[HPV77] John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, April 1977.

[Koz77] Dexter Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science (FOCS '77)*, pages 254–266, 1977.

[Nor12] Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. Manuscript in preparation. To appear in *Foundations and Trends in Theoretical Computer Science*. Current draft version available at http://www.csc.kth.se/~jakobn/research/, 2012.

[Pip80] Nicholas Pippenger. Pebbling. Technical Report RC8258, IBM Watson Research Center, 1980. Appeared in Proceedings of the 5th IBM Symposium on Mathematical Foundations of Computer Science, Japan.

[RM99] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, March 1999. Preliminary version appeared in *FOCS '97*.

[Sav98] John E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley, 1998. Available at http://www.modelsofcomputation.org.