The canonical pairs of bounded depth Frege systems

Pavel Pudlák *

September 30, 2020

Abstract

The canonical pair of a proof system P is the pair of disjoint NP sets where one set is the set of all satisfiable CNF formulas and the other is the set of CNF formulas that have P-proofs bounded by some polynomial. We give a combinatorial characterization of the canonical pairs of depth d Frege systems. Our characterization is based on certain games, introduced in this article, that are parametrized by a number k, also called the depth. We show that the canonical pair of a depth d Frege system is polynomially equivalent to the pair (A_{d+2}, B_{d+2}) where A_{d+2} (respectively, B_{d+1}) are depth $d+1$ games in which Player I (Player II) has a positional winning strategy. Although this characterization is stated in terms of games, we will show that these combinatorial structures can be viewed as generalizations of monotone Boolean circuits. In particular, depth 1 games are essentially monotone Boolean circuits. Thus we get a generalization of the monotone feasible interpolation for Resolution, which is a property that enables one to reduce the task of proving lower bounds on the size of refutations to lower bounds on the size of monotone Boolean circuits. However, we do not have a method yet for proving lower bounds on the size of depth d games for $d > 1$.

1 Introduction

There are two basic problems associated with every propositional proof system P:

1. Given a CNF formula ϕ, decide whether ϕ is satisfiable or has a short P refutation, provided that we know that one of these statements is true.

2. Let two CNF formulas ϕ and ψ with disjoint sets of variables and a refutation π of $\phi \wedge \psi$ be given and assume that one of the formulas is satisfiable. Decide which.

These problems are formalized by defining pairs of disjoint NP sets. The pair of the first problem is called the *canonical pair* of P (introduced by Razborov in [17]), the pair of

*The author is supported by the project EPAC, funded by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X, and the institute grant RVO: 67985840. Part of this work was done when the author was supported by the ERC Advanced Grant 339691 (FEALORA).

the second one is called the *interpolation pair* (introduced in [15]). We will see that these two problems are tightly connected and thus one can focus only on one of them. Since interpolation pairs are not so well-known as canonical pairs, we use this name in the title, but in fact, we will study interpolation pairs.

We conjecture that the hardness of these problems increases with the strength of the proof systems, where we compare hardness of disjoint NP pairs by polynomial reductions. We do not have means to prove that one pair is stronger than another, because if P=NP all are interreducible. So we have only two possibilities what to do. First, we can try and find mathematical principles equivalent to the facts that these pairs are disjoint. If the principles seem of increasing strength we can view it as evidence of increasing hardness of these pairs. Second, we can consider monotone versions of these problems. Since we do have lower bounds on monotone Boolean circuits and some other monotone computational models (in contrast to the desperate state of the affairs with general Boolean circuits), there is some chance that we can prove separation at least with respect to monotone reductions.

It is natural to start with the weakest systems. Prior to this work, a combinatorial characterization of the canonical pair was known only for the Resolution system [4]. Bounded depth Frege is a well-studied hierarchy of proof systems above Resolution. In this article we will present a combinatorial characterizations of interpolation pairs for all levels of this hierarchy, which also gives characterizations of canonical pairs.

Our characterization is based on certain two player games which we will briefly explain now and give a precise definition later. Two players alternate in writing symbols on a finite tape. They start on one end, say the left one, and proceed to the other. When they reach the end they either stop, if the game has only one round, or they reverse the direction and go back. They may reverse direction $(k-1)$-times if the depth parameter is k. What is a legal move only depends on the symbol in the current square and the next one. We define *positional* strategies and show that given a positional strategy one can decide in polynomial time whether it is a winning strategy for the particular player. This enables us to define an NP pair for every depth $k \geq 1$ and characterize the interpolation pair of depth-d Frege systems by games of depth $d+1$. The canonical pair of depth d Frege systems is polynomially equivalent to the interpolation pair of depth $d+1$ Frege systems.

One can view a game of depth $k+1$ as follows. In the first round the players alternate to define a string of symbols that determine a game of depth k that is played after the first round. This suggests the intuition that it should be harder to decide who has a (positional) winning strategy in a game of depth $k+1$: we cannot use an oracle for games of depth k because the game of depth k is yet to be determined by playing the first round.

Our result can also be viewed as a contribution to the line of research that studies monotone computation models. We will show that one can interpret our games, more precisely game schemas, as monotone computation models generalizing monotone Boolean circuits. It is not clear how difficult it may be to prove lower bounds on these models, but if we could do it, we may be able to solve an important problem about bounded depth Frege systems. Impagliazzo and Krajíček [11] proved that depth d Frege systems cannot polynomially simulate depth $d+1$ Frege systems w.r.t. refuting CNFs. However their lower bound is

only mildly superpolynomial, while we believe there should be an exponential separation. The tautologies, or rather contradictions, based on games are candidates for exponential separations.

A reader familiar with results in proof complexity and Bounded Arithmetic will recognize several connections between this work and previous ones. The *Symmetric Calculus,* which we will introduce in Section 3, is inspired by the calculus invented by Skelley and Thapen [18], but there are other deep-inference calculi that are related to our Symmetric Calculus. Skelley and Thapen used their calculus for proving that *Game Induction* principles characterize $\forall\Sigma^b_1$ provable principles in fragments of Bounded Arithmetic. These principles are also very much related to our bounded depth games. It is possible that some arguments from [18] could be used for proving our result, similarly as one might use Skelley-Thapen's calculus instead of our Symmetric Calculus. A game similar to ours appeared in an article of Kołodziejczyk, Nguyen and Thapen [12] (Lemma 10). We will see in Section 8 that the *point-line game*, introduced an article of Beckmann, Pudlák, and Thapen [4], can be viewed as a version of our depth-2 game. There are certainly more connections than those mentioned above.

This article is essentially a proof of a single theorem, Theorem 4.2, plus some observations. We start by recalling the definitions of bounded depth sequent calculi (that are used instead of bounded depth Frege calculi), canonical and interpolation pairs and stating some basic facts. Then we define the *Symmetric Calculus*. This calculus, more precisely its bounded depth version, has been designed for proving our theorem, but it may be of independent interest. The basic idea is due to Skelley and Thapen [18], but our calculus differs in several particulars. We show that the bounded version of Symmetric Calculus is polynomially equivalent to the standard formalization by the sequent calculus. In Section 4 we define the games used in the characterization. In Sections 5 and 6 we construct the reductions between the canonical pairs of bounded depth calculi and pairs defined by our games. In Section 7 we prove a stronger version of our main theorem. In Section 8 we will have a closer look at games of depth 1 and 2. We conclude the article with some open problems. At the end there is a short appendix in which we mention a connection to Bounded Arithmetic, which is important, but not used in this article, and explain one technical point from the simulation of bounded depth calculi by the bounded depth Symmetric Calculi.

Acknowledgment. I am grateful to Emil Jeřábek and Jan Krajíček for their comments on the draft of this article and especially to Neil Thapen for reading the whole manuscript and pointing to incomplete, or unclear parts. I would like to thank Anupam Das for telling me about related deep-inference calculi. I am indebted to an anonymous referee for suggesting many corrections.

2 Basic notions

In this section we recall some concepts and results from proof complexity that we will use.

2.1 Bounded depth sequent calculus

Classical propositional logic can be formalized by various types of calculi, which may have different power. We compare calculi by how efficiently they can prove tautologies. A tautology that can only be proved by exponentially long proofs in one system may have polynomial size proofs in another. The standard formalization of propositional logic is based on axioms and derivation rules. These calculi are called *Frege calculi*. They are equivalent, from the point of view of efficiency, to the *sequent calculus*.

In this article we are interested in restricted versions of these calculi where the depth of formulas is bounded by a constant. To this end we will restrict our language to the De Morgan basis $\neg, \vee, \wedge$. The *depth* of a formula is defined to be the number of alternations of $\neg, \vee, \wedge$, where negations at variables are not counted. Thus variables and negated variables, called *literals*, have depth 0, conjunctions of literals have depth 1, disjunctions of literals, called*clauses*, have depth 1, CNFs and DNFs have depth 2, etc.

For formalizing reasoning with bounded depth formulas, the sequent calculus is more convenient than Frege calculi. We define *depth d sequent calculus* to be the standard sequent calculus restricted to the De Morgan basis and formulas of depth at most d. In the sequel we will only use bounded depth sequent calculi, but the reader should keep in mind that they are equivalent to bounded depth Frege calculi.

Another useful convention is to use *refutations* instead of derivations. Given a DNF tautology τ, we take the CNF contradictory formula σ obtained by the dualization of τ and prove contradiction from σ (contradiction is represented by the empty sequent). Since a CNF formula can be represented by a set of clauses and clauses can be represented by sequents consisting of literals only, we can use even the depth 0 calculus. The depth 0 calculus is essentially the *Resolution* system, because the only non-structural rule that can be used is cut with a literal as the cut formula.

We will shortly introduce yet another calculus for reasoning with bounded depth formulas, the *Symmetric Calculus*. In this calculus, Π_{k+2} proofs correspond to proofs in the depth k sequent calculus.

A remark on notation. We will use various letters, Greek and Latin, to denote propositional formulas. Different types of variables do not correspond to particular types of formulas. We will only follow one rule: small Latin letters stand for variables and numbers, never formulas.

2.2 Polynomial simulations and disjoint NP pairs of propositional proof systems

We say that a proof system P *polynomially simulates a proof system* Q if there exists a polynomial time algorithm that from a given P-proof (or refutation) π of a formula ϕ, constructs a Q-proof (or refutation) of ϕ. We say that P *and* Q *are polynomially equivalent* if they polynomially simulate each other.

Let (A, B) and (C, D) be two pairs of disjoint NP sets. We say that (A, B) is *polynomially reducible* to (C, D) if there exists a polynomial time algorithm that maps A into C and B into D.

The *canonical pair of a proof system* P (defined in [17]) is the pair of disjoint NP-sets (A, B) where

$$A := \{(\phi, 0^m) \mid \phi \text{ satisfiable } \},$$
$$B := \{(\phi, 0^m) \mid \phi \text{ has a } P \text{ refutation of size } m\}.$$

The string of zeros 0^m of length m is a padding that enables us to consider proofs of arbitrary size. In all natural proof systems we can replace this padding by padding the formulas with trivially satisfiable clauses. Then we can define A to be satisfiable formulas and B to be formulas ϕ that have P-refutations of size $|\phi|^2$.

The *interpolation pair of a proof system* P (defined in [15]) is the pair of disjoint NP-sets (A, B) where

$$A := \{(\phi, \psi, \pi) \in \Delta \mid \phi \text{ satisfiable } \},$$
$$B := \{(\phi, \psi, \pi) \in \Delta \mid \psi \text{ satisfiable } \},$$

where Δ is the set of triples (ϕ, ψ, π) such that ϕ and ψ are formulas with disjoint sets of variables and π is a P-refutation of $\phi \wedge \psi$.

In these definitions we have not specified the class of Boolean formulas. Propositional proof systems may use restricted classes of formulas and if they use different classes, then polynomial simulation does not make sense. A natural minimal requirement is that a proof system is complete with respect to refutations of unsatisfiable CNF formulas. Therefore *we restrict the above definitions to CNF formulas.* One can show, under very mild assumptions about the proof systems, that such restricted pairs are polynomially equivalent to the pairs defined above. Hence we do not lose information about the complexity of these pairs if we focus on CNFs.

Furthermore, we will say that *a proof system is natural,* if given a P-proof of $\alpha(\bar{z}, \bar{x})$, one can construct in polynomial time a P-proof of $\alpha(\bar{a}, \bar{x})$ for every assignment $\bar{a}$ to $\bar{z}$. Here we identify proofs of ϕ with refutations of $\neg\phi$.

The basic facts about canonical and interpolation pairs of natural proof systems are:

1. *the interpolation pair of P is polynomially reducible to the canonical pair of P;*

2. *if P polynomially simulates Q, then the canonical (respectively, interpolation) pair of Q is polynomially reducible to the canonical (interpolation) pair of P.*

For bounded depth sequent calculi, we have the following important fact.

Proposition 2.1 ([4], Proposition 1.4) *For $k \geq 0$, the canonical pair of depth k sequent calculus is polynomially equivalent to the interpolation pair of depth (k+1) sequent calculus.*

Therefore it suffices to characterize the interpolation pairs.

With each disjoint NP pair (A, B), there is an associated *separation problem:* given the promise that $x \in A \cup B$, how difficult is to decide whether $x \in A$ or $x \in B$? For the

interpolation pair of depth 0 sequent calculus, which is the same as Resolution, the problem is decidable in polynomial time. For higher systems we do not know if the separation is solvable in polynomial time. By improving some previous results, Bonet et al. [5] proved that assuming factoring Blum integers or computing the Diffie-Helman function is sufficiently hard, the separation problem for the interpolation pairs is not polynomially solvable for all depth d sequent calculi starting from some small d_0. Less convincing evidence of the hardness are results showing that the decision of who has a winning strategy in certain combinatorial games can be reduced to the interpolation pairs of some small depth d sequent calculi [10, 2, 4]. In particular, the decision problem for *parity games* can be reduced to the canonical pair of depth 0 sequent calculus, i.e., Resolution, and the decision problem for *simple stochastic games* can be reduced to the canonical pair of depth 1 sequent calculus. The decision problem for parity games is solvable in quasipolynomial time, but the author of this article thinks that the canonical pair of Resolution is harder. This belief is supported by the recent result of Atserias and Müller [3] that the proof search in the Resolution system is NP-hard.

2.3 Feasible interpolation

The feasible interpolation property. We say that a proof system P has the *feasible interpolation property* if there exists a polynomial time algorithm A such that given a P-proof D of a formula of the form

$$\phi(\bar{z}, \bar{x}) \vee \psi(\bar{z}, \bar{y}),$$

where the strings of variables $\bar{z}$, $\bar{x}$ and $\bar{y}$ are disjoint, and given an assignment $\bar{a} \in \{0,1\}^n$ to variables $\bar{z}$, the following holds true:

$$\begin{aligned} &\text{if } \phi(\bar{a}, \bar{x}) \text{ is satisfiable, then } A(D, \bar{a}) = 0, \\ &\text{if } \psi(\bar{a}, \bar{y}) \text{ is satisfiable, then } A(D, \bar{a}) = 1. \end{aligned} \tag{1}$$

It follows that for every D, there exists a *Boolean circuit* $C(\bar{z})$ whose size is polynomial in the size of D and such that condition (1) is satisfied with $A(D, \bar{a})$ replaced with $C(\bar{a})$. For natural proof systems, P has the feasible interpolation property iff the interpolation pair of P is separable by a polynomial time algorithm.

The monotone feasible interpolation property. Often one can show that there exists a *monotone* Boolean circuit with the above properties. For this, it is necessary to ensure that the two sets can be separated by monotone functions, which is done by assuming that all common variables $\bar{z}$ occur only negatively in ϕ, or all occur positively in ϕ (or both). If for this kind of formulas, there exist polynomial size monotone circuits with property (1), then we say that the proof system has the *monotone feasible property.*

It is well-known that the Resolution proof system has both the feasible interpolation property and the monotone feasible property. There are a few more natural proof systems that have the feasible interpolation property, some also have the monotone feasible property,

see [13], Chapter 17. Our Theorem 7.1 can be viewed as a generalization of the result for Resolution to stronger fragments of the propositional sequent calculus.

3 The Symmetric Calculus

This calculus is specially designed for the proof of the main theorem. The starting point was the calculus of Skelley and Thapen [18]. In their calculus the eigenformulas of non-structural rules are just literals. Thus instead of the general cut rule they use the resolution rule, i.e., cut with a literal as the eigenformula, and the rule for conjunction introduction only allows a conjunction to be extended by a literal. The fact that such a calculus can polynomially simulate the bounded depth sequent calculus is a remarkable discovery, because it is well-known that if one restricts cuts in the sequent calculus to depth d formulas, then the resulting system has only the power of the depth d sequent calculus (for refuting CNFs). The reason why restriction to literals does not limit the Skelley-Thapen calculus is that the calculus uses deep inferences.

In the Symmetric Calculus, unlike the Skelley-Thapen calculus, all rules can be applied as deep inferences. In order to achieve symmetry of the rules, we have replaced conjunction introduction by dual resolution. Furthermore, the proofs do not have the traditional structure where the set of initial formulas is gradually extended by derived formulas. In the Symmetric Calculus a proof has a linear structure—a sequence of formulas such that the next formula follows only from the previous formula.

Our calculus is similar to other deep-inference calculi that appeared in the literature. In Section 3.2 we will briefly mention its relation to a popular deep-inference calculus SKS [7, 6].

In the bounded version of the Symmetric Calculus there are further structural restrictions. Let us repeat that this is only because we need to make a connection with certain games.

Definition of the Symmetric Calculus. The language of the Symmetric Calculus consists of $\vee, \wedge, \bot, \top$, and literals $x_i, \neg x_i$. Negations are allowed only at literals. Given a literal p, we denote by $\neg p$ its dual.

The calculus is based on *deep inferences*, which means that one can replace a *subformula* by a formula allowed by a rule. An application of a rule $\frac{B}{C}$ is a substitution

$$\frac{A[\dots B\dots]}{A[\dots C\dots]}$$

Formula B can occur on several places in A, but in one step of the proof only one occurrence of B is replaced.

In the Symmetric Calculus every rule has one assumption and one conclusion, so the calculus is a term rewriting system. A *proof* of $\Phi \vdash \Psi$ is a sequence of formulas $\Phi = \Phi_1, \dots, \Phi_m = \Psi$ where Φ_{i+1} follows from Φ_i by an application of a deduction rule.

The rules of the calculus:

commutativity and associativity of $\vee$ *and* $\wedge$,[1]

contraction/cloning

$$\frac{A \vee A}{A} \qquad\qquad \frac{A}{A \wedge A}$$

$\bot$*-elimination /* $\top$*-introduction*

$$\frac{A \vee \bot}{A} \qquad\qquad \frac{A}{A \wedge \top}$$

weakenings

$$\frac{A}{A \vee B} \qquad\qquad \frac{A \wedge B}{A}$$

dual resolution/resolution

$$\frac{A \wedge \top \wedge B}{(A \wedge p) \vee (B \wedge \neg p)} \qquad\qquad \frac{(A \vee p) \wedge (B \vee \neg p)}{A \vee \bot \vee B}$$

In the last two rules A or B or both formulas may be not present; e.g., $\frac{\top}{p \vee \neg p}$ is considered to be an instance of dual resolution.

The reason for calling this calculus "symmetric" is that each rule has its dual. Hence, given a proof of $\Phi \vdash \Psi$ we can obtain a proof of $\neg\Psi \vdash \neg\Phi$, by inverting the order of formulas and replacing connectives, truth constants, and literals by their duals. Here and also in the sequel, $\neg\Phi$ denotes the dual of the formula Φ. More importantly, the symmetry allows us to cut the case analysis to one half.

Several comments about the rules are in order. The reason for using the rules for the truth constants in this form instead of the standard ones

$$\frac{\bot}{A} \qquad\qquad \frac{A}{\top}$$

is purely technical. The weakenings are also called "disjunction introduction" and "conjunction elimination". We prefer to view them as weakenings, because we do not have rules for disjunction elimination and conjunction introduction.[2] The truth constants in resolution and dual resolution rules can, clearly, be omitted when at least one of the formulas A or B is present, but we prefer to keep the constants also when the context is nonempty because it will simplify the simulation of proofs by our combinatorial games. However, we will omit the constants when proving simulations of bounded depth Frege systems, in order to simplify formulas used in these proofs.

[1] Emil Jeřábek has observed that associativity is redundant and if we add also the other two versions of weakenings, $\frac{B}{A \vee B}$, $\frac{A \wedge B}{B}$, then also commutativity will be redundant.

[2] Another reason is that the rule of weakening can be omitted in sequent calculi if we use two versions of every other rule: one in which the premises are "consumed" and one in which they stay. This might be possible also in this calculus, but we have not investigated this possibility.

3.1 Cuts and dual cuts

Our aim now is to prove that dual cuts and cuts can be simulated. These derived rules are generalizations of the rules of dual resolution and resolution where one can use arbitrary formulas C instead of literals:

$$\frac{A \wedge \top \wedge B}{(A \wedge C) \vee (B \wedge \neg C)} \qquad \frac{(A \vee C) \wedge (B \vee \neg C)}{A \vee \bot \vee B}$$

where $\top$ and $\bot$ can be omitted if at least one of the formulas A, B is present. Recall that $\neg C$ denotes the formula obtained from C by replacing the connectives and literals by their duals.

We will start with a simple fact.

Fact 1 *The distributivity of $\vee$ over $\wedge$ can be polynomially simulated.*

Proof.

$$\dfrac{\dfrac{A \vee (B \wedge C)}{(A \vee (B \wedge C)) \wedge (A \vee (B \wedge C))}\ \text{by cloning}}{(A \vee B) \wedge (A \vee C)}\ \text{by weakenings}$$

■

The proof above explains what we mean by *polynomial simulation:* if ψ is a formula obtained from ϕ by replacing an occurrence of $\alpha \vee (\beta \wedge \gamma)$ with $(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$, then one can construct in polynomial time a derivation of ψ from ϕ in the Symmetric Calculus.

Using the symmetry of our calculus we immediately get that

$$\frac{(A \wedge B) \vee (A \wedge C)}{A \wedge (B \vee C)}$$

also can be polynomially simulated. The distributivity of $\wedge$ over $\vee$ can be simulated too, but to prove it, we first need to simulate cuts and dual cuts.

Lemma 3.1 *Dual cuts and cuts can be polynomially simulated.*

Proof. We will describe a procedure that constructs a proof that simulates cut. Suppose we want to simulate the following deduction

$$\frac{(A \vee C) \wedge (B \vee \neg C)}{A \vee B}.$$

Using weakening, the initial formula can be transformed into $(A \vee B \vee C) \wedge (A \vee B \vee \neg C)$. Hence it suffices to simulate cuts with the same context, which we will denote by A:

$$\frac{(A \vee C) \wedge (A \vee \neg C)}{A}.$$

If C is a literal, then this is just an application of the resolution rule. Now suppose that C is $C_1 \wedge C_2$ (the case of C being $C_1 \vee C_2$ will follow by symmetry, because $\neg C$ would be $\neg C_1 \wedge \neg C_2$). First we use distributivity to obtain

$$(A \vee C_1) \wedge (A \vee C_2) \wedge (A \vee \neg C_1 \vee \neg C_2).$$

Then we apply the procedure recursively to the subformula $(A \vee C_1) \wedge (A \vee \neg C_1 \vee \neg C_2)$ (after permuting $A \vee C_1$ and $A \vee C_2$). Thus we obtain

$$\frac{(A \vee C_2) \wedge (A \vee A \vee \neg C_2)}{(A \vee C_2) \wedge (A \vee \neg C_2)}.$$

using contraction in the last step. In this way we have reduced the problem to a smaller cut-formula C_2 and we can recursively call the procedure again.

To see that this gives a polynomial simulation it suffices to observe the following facts:

1. the number of times the procedure calls itself is equal to the number of occurrences of subformulas of C;

2. each call of the procedure adds a term to the initial conjunction $(A \vee C) \wedge (B \vee \neg C)$ whose size is bounded by either the size of $A \vee C$ or $B \vee \neg C$. In the derivation above the added formula was first a clone of the formula $A \vee C$ and then it was weakened to $A \vee C_2$, while we consider $A \vee C_1$ to be only a weakened original $A \vee C$.

3. Because of the added formulas, the size of a formula in the proof may be quadratic in the size of the first formula, the assumption. However, the number of recursive calls is still bounded by the number of occurrences of subformulas of C.

The case of dual cuts follows by symmetry. See also Appendix for a remark and an example. ∎

We can now show the simulation of the distributivity of $\wedge$ over $\vee$.

Lemma 3.2 *The distributivity of $\wedge$ over $\vee$ with can be polynomially simulated.*

Proof.

$$\cfrac{\cfrac{\cfrac{\cfrac{A \wedge (B \vee C)}{A \wedge (B \vee C) \wedge A}\ \text{by cloning } A}{(A \wedge (B \vee C) \wedge \neg C) \vee (A \wedge C)}\ \text{by a dual cut with } C}{(A \wedge B) \vee (A \wedge C)}\ \text{by a cut with } C$$

∎

By symmetry, we get immediately

$$\frac{(A \vee B) \wedge (A \vee C)}{A \vee (B \wedge C)} \tag{2}$$

This is important because it enables us to simulate *conjunction introduction* in the Sequent Calculus. Since we also have dual resolution (we can derive $p \vee \neg p$ by $\top$-introduction and dual resolution), we have simulations of all rules of the Sequent Calculus as was formalized by Tait, cf. [8] paragraph 1.2.14. Thus we have shown:

Proposition 3.3 *The Symmetric Calculus polynomially simulates the Sequent Calculus, hence also Frege calculi.*

Later we will also need

$$\frac{A \to (B \vee C)}{(A \wedge D) \to ((B \wedge D) \vee C)} \tag{3}$$

If we write $\to$ in terms of $\vee, \neg$, this becomes

$$\frac{\neg A \vee B \vee C}{\neg A \vee \neg D \vee (B \wedge D) \vee C}$$

which is simply an application of dual cut.

We will show that these simulations also hold for bounded depth versions of the calculi; this will be more complicated.

3.2 A remark on the SKS calculus

The SKS deep-inference calculus was introduced by K. Brünnler and A. F. Tiu [7, 6]. Not knowing about this calculus we decided to call our calculus the Symmetric Calculus; incidentally the first S from SKS also stands for "symmetric". Although superficially the two systems look very similar, we cannot use SKS instead of the Symmetric Calculus. Consider the *switch* rule of SKS:

$$\frac{A \wedge (B \vee C)}{(A \wedge B) \vee C}.$$

When simulating this step in our game, one player would have to decide which of the two formulas $A \wedge B$ and C is true knowing that $A \wedge (B \vee C)$ is true. But each of the two players knows only his/her half of variables, hence in general cannot perform this task.[3]

Furthermore, we cannot simplify the simulation of the sequent calculus by the Symmetric Calculus using SKS. We do not see a *simple* way (and believe there isn't any) to polynomially simulate the switch rule by the Symmetric Calculus. We can simulate the switch by first simulating the distributivity, cf. Lemma 3.2, and then applying weakening, but in order to get the distributivity we need to simulate cut, which is nontrivial. Once we have cut, we can easily simulate Tait's sequent calculus and do not need SKS.

[3]This may be not quite clear at this point, so we invite the reader to return to this paragraph after reading the rest of the paper.

3.3 The Bounded Depth Symmetric Calculus

In the Bounded Depth Symmetric Calculus we follow the convention used in the unbounded case that negations are allowed only at literals. We define Π and Σ classes of propositional formulas in the usual manner: literals are $\Sigma_0 = \Pi_0$, conjunctions of literals are Π_1, disjunctions of literals are Σ_1 etc.

In the bounded depth calculus conjunctions and disjuctions are operations with an arbitrary finite number of arguments $n = 1, 2, 3, \ldots$. The rules of commutativity and associativity are replaced by a general rule of permutation.[4] The other rules of the Symmetric Calculus are applied to consecutive formulas in a possibly longer disjunction, or conjunction, and the derived formula is inserted on the position of the premise. For instance, an application of the contraction rule transforms a disjunction into a disjunction with one term less as follows:

$$\frac{B_1 \vee \cdots \vee B_k \vee A \vee A \vee C_1 \vee \cdots \vee C_l}{B_1 \vee \cdots \vee B_k \vee A \vee C_1 \vee \cdots \vee C_l}$$

Stratified formulas. Furthermore, we require that $\wedge$ and $\vee$ alternate regularly in formulas as we go from the top connective to the bottom of the formula. Balanced formulas with $\wedge$ and $\vee$ alternating regularly will be called *stratified*. More precisely, *stratified* Π_k *formulas* (Σ_k *formulas*) have the structure of rooted trees in which

1. every branch has length k (measured by the number of edges),
2. on every branch $\wedge$ and $\vee$ alternate regularly starting with $\wedge$ (respectively with $\vee$),
3. the leaves are labeled by literals.

We will use Π_k^s and Σ_k^s for the classes of stratified Π_k and Σ_k formulas. Note that formulas in classes Π_k^s and Σ_k^s have depth *exactly* k.

In order to represent formulas that are not in this form, we will use unary operations of $\wedge$ and $\vee$. We will use prefix notation for these unary conjunctions and disjunctions, e.g., $\wedge(A)$, or just $\wedge A$, while keeping infix notation for strings of formulas with at least two terms.

> *Example.* The formula $(p \vee q) \wedge r$ can be represented by the stratified formula $(p \vee q) \wedge (\vee r)$. Note that in general there may be several different representations (see zipping and unzipping below).

We will say that a formula ϕ is a *legal subformula* of ψ if it is a formula determined by a node N in the tree representing ψ in the following sense: the tree of ϕ is the entire tree below N.

> *Example.* In the formula $(p \wedge q \wedge r) \vee (s \wedge t)$, the formula $p \wedge q \wedge r$ is a legal subformula, whereas $p \wedge q$ isn't.

[4] Why don't we use multisets? The reason is that we want to have correspondence between subformulas of a formula and subformulas in its successor in a proof, which is needed for defining games from proofs.

We will study Π^s_k proofs of contradiction. In such a proof $A_1, A_2, \ldots, A_n$, *every formula* A_i *must be* Π^s_k. E.g., if we are refuting a CNF formula A, we must pad it to a Π^s_k formula and the final $\bot$ must be padded to level Π^s_k too.

The deep inferences of the Symmetric Calculus may be applied only to legal subformulas. To this end we have to modify the rules. We will only define the modification of the rules in the left column of the list of rules in the previous subsection; the right column is done symmetrically.

1. Contraction means that we can replace two consecutive terms in a disjunction when they are equal.

2. Elimination of $\bot$ means that we remove it from the disjunction. This is not allowed if it is the only term in the disjunction (unlike in Resolution, in the Symmetric Calculus empty disjunctions are not used).

3. Weakening of a disjunction means inserting an arbitrary formula of an appropriate depth into the disjunction on arbitrary place.

4. Dual resolution means that we split a conjunction into the part before $\top$ and the part after, omit $\top$, and add a literal to the first part and the dual literal to the second:

$$\frac{A_1 \wedge \cdots \wedge A_i \wedge \top \wedge A_{i+1} \wedge \cdots \wedge A_n}{(A_1 \wedge \cdots \wedge A_i \wedge p) \vee (A_{i+1} \wedge \cdots \wedge A_n \wedge \neg p)}.$$

   The added literals must be padded to the appropriate level in order to preserve stratification. The padding is not shown in the formula above and will also be omitted in the sequel in order to simplify notation.

   Example. Let $A, B, C \in \Sigma^s_k$. The following is *not* a legal application of the dual resolution in the bounded depth Symmetric Calculus

$$\frac{A \wedge \top \wedge B \wedge C}{((A \wedge p) \vee (B \wedge \neg p)) \wedge (\vee \wedge C)},$$

   even though the conclusion is stratified, because the rule is not applied to the entire conjunction.

For the sake of readability, we have omitted the padding of literals and $\top$ in the equations above.

We require that the rules be applied so that stratification is preserved. If contraction is applied to a disjunction in which there are only the two terms $A \vee A$, then the result is a unary disjunction $\vee(A)$; similarly for $\bot$-elimination. Weakening $\frac{A}{A \vee B}$ can only be applied if A is a part of disjunction, which may be just $\vee(A)$. Similarly for dual resolution, $A_1 \wedge \cdots \wedge A_i \wedge \top \wedge A_{i+1} \wedge \cdots \wedge A_n$ must be a term in a disjunction, possibly the unique term in the disjunction. Thus we have

- all rules in the left column can only be applied to disjunctions and
- all rules in the right column can only be applied to conjunctions.

Negations. In order to simulate the sequent calculus we need to define negations of stratified formulas. Given a Π^s_k formula A, if we just dualize it, as we did before, we get a Σ^s_k. These formulas cannot occur as subformulas in the same proof, because the bottom connectives are different. Therefore we define the stratified negation of A to be the dualized formula with literals padded by the bottom connective of A. So $\neg A$ is a stratified Σ^s_{k+1} formula.

Example If A is $p \wedge \neg q$, then $\neg A$ is $\wedge(\neg p) \vee \wedge(q)$.

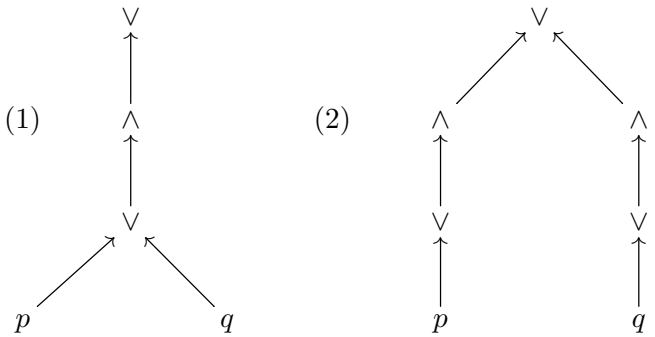
Similarly, if A is Σ^s_k, then $\neg A$ is Π^s_{k+1}. But there is an exception: if A has the form of a negated formula, say $\neg B$, then $\neg A$ is just B.

Efficient simulations. Given two formula schemas $\mathcal{A}$ and $\mathcal{B}$, we will say that $\mathcal{A} \vdash \mathcal{B}$ can be *efficiently simulated* if there exists a polynomial simulation which, for every given instance $A \vdash B$, produces a proof in bounded depth Symmetric Calculus in which the depth of the formulas does not exceed the depth of the two formulas, i.e., if $A, B \in \Pi^s_i$ (respectively $A, B \in \Sigma^s_i$), then all formulas in the proof are in Π^s_i (in Σ^s_i).

3.4 Zipping and unzipping

There is ambiguity in representing formulas by stratified formulas, one formula may have several representations. We must show that it is easy to transform one representation to any other representation, otherwise the system would not be natural. The basic transformations that enable us to do this will be called zipping and unzipping.

Example. Suppose we need to represent the formula $p \vee q$ as a Σ^s_3 formula. Then we have two possibilities: (1) $\vee \wedge (p \vee q)$, (2) $(\wedge \vee p) \vee (\wedge \vee q)$.



In general we can have arbitrary formulas instead of literals. We call the operation (1)↦(2) *unzipping* and the converse *zipping*. In the example above we have unzipped the formula from a vertex with $\vee$ to the next vertex with $\vee$ above it, but in general unzipping can be done to any vertex above it with the same connective. The same concerns zipping.

Fact 2 *Any representation of a formula can be transformed to any other representation by zipping and/or unzipping. The number of the operation is bounded by the size of the formulas.*

The proof is an easy exercise (which also involves a proper definition of representation).

Lemma 3.4 *Zipping and unzipping can be efficiently simulated, assuming that cuts and dual cuts can be.*

Proof. Due to the symmetry of the calculus it is enough to simulate zipping and unzipping of conjunctions. Unzipping conjunctions is easy. It is done by cloning and weakenings:

$$\wedge\vee(A\wedge B) \;\mapsto\; (\wedge\vee(A\wedge B))\wedge(\wedge\vee(A\wedge B)) \;\mapsto\; (\vee\wedge(A))\wedge(\vee\wedge(B)).$$

To simulate zipping we use a dual cut and a cut:

$$(\vee\wedge(A))\wedge(\vee\wedge(B)) \mapsto\; ((A\wedge B)\vee\neg B)\wedge(\vee\wedge(B)) \mapsto\; \wedge\vee(A\wedge B).$$

If B is a literal, then $\neg B$ should be $\wedge(\neg B)$; otherwise $\neg B$ is lifted to a higher level. For the sake of simplicity, we have assumed that the conjunctions $\wedge(A)$ and $\wedge(B)$ have single terms. We leave the case when we have $\bigwedge_i A_i$ or $\bigwedge_i B_i$ with more terms A_i or B_i to the reader. ∎

Note that this implies that we can efficiently simulate zipping and unzipping not only conjunctions and disjunctions with two terms, but with any number greater or equal to 2.

3.5 Cuts and dual cuts in the bounded depth system

Now we want to show that one can efficiently simulate cuts and dual cuts in the bounded depth Symmetric Calculus. The simulation is the same as in the unbounded case except for one complication: zipped formulas. Suppose we want to simulate the cut

$$\frac{(A\vee C)\wedge(\neg C\vee A)}{A}.$$

If C is a literal padded to the particular level, this is the resolution rule of the bounded depth Symmetric Calculus.

Assume that C is more complex formula. W.l.o.g. we may assume that C is a conjunction, or padded conjunction, otherwise we would take $\neg C$. First suppose C is not padded; let it be $C_1\wedge C_2\wedge\cdots\wedge C_n$ with $n\geq 2$. The idea of the proof is the same as in the unbounded case: we distribute A to get

$$(A\vee C_1)\wedge(A\vee(C_2\wedge\cdots\wedge C_n))\wedge(\neg C_1\vee\neg C_2\vee\cdots\vee\neg C_n\vee A).$$

Then we use the efficient simulation of the cut for the simpler formula C_1 and after contracting two occurences of A we get

$$(A\vee(C_2\wedge\cdots\wedge C_n))\wedge(\neg C_2\vee\cdots\vee\neg C_n\vee A).$$

Thus we have reduced our task to a simulation of cut with the simpler cut formula $C_2\wedge\cdots\wedge C_n$. Recall that the version of distributivity that we needed can be derived by cloning and

weakening, which do not increase the depth. Furthermore, we assume that the simulation of smaller cuts, namely for C_1, does not exceed the depth of the formulas involved. Hence we do not exceed the original depth of the formula in this derivation.

If C is a zipped conjunction $\wedge \vee \ldots (C_1 \wedge C_2 \wedge \cdots \wedge C_n)$, we cannot use distributivity immediately, we have to first unzip the formula. Unzipping conjunction is easy, but we also have to unzip the disjunction in $\vee \wedge \ldots (\neg C_1 \vee \neg C_2 \cdots \vee \neg C_n) \vee B$. To unzip disjunction, we need a dual cut and a cut. Fortunately, we need these operation for formulas that are simpler than C. So if we assume that we already have simulations for simpler formulas, we also have simulations of unzipping disjunctions for these formulas.

Thus the simulation is based, as in the unrestricted depth case, on recursively calling procedures for smaller formulas. The reason why it is polynomial time is the same as in the unrestricted depth case (see the proof of Lemma 3.1), although we have into account also zipping and unzipping. But the same three facts from Lemma 3.1 apply to zipping and unzipping.

So we have shown:

Lemma 3.5 *Cuts and dual cuts can be efficiently simulated in the bounded depth Symmetric Calculus.*

3.6 Efficient simulation of distributivity

In order to simulate conjunction introduction, we have to efficiently simulate the distributive law in the form of the following rule

$$\frac{(A \vee B) \wedge (A \vee C)}{A \vee (B \wedge C)}$$

Lemma 3.6 *Distributivity can be efficiently simulated.*

Proof. We will use the dual form of the proof of Lemma 3.2 as before, however we have to show that the depth of the formulas in the proof does not exceed the depth of the formulas $(A \vee B) \wedge (A \vee C)$ and $A \vee (B \wedge C)$. We will suppose that $(A \vee B) \wedge (A \vee C)$ is Π^s_{k+2} and $A \vee (B \wedge C)$ is Σ_{k+1} padded to Π^s_{k+2}. Consider the derivation:

$$\begin{array}{c} (A \vee B) \wedge (A \vee C) \\ \hline (A \vee (B \wedge C) \vee \neg C) \wedge (A \vee C) \\ \hline A \vee (B \wedge C) \vee A \\ \hline A \vee (B \wedge C) \end{array} \qquad \begin{array}{l} \\ \text{by a dual cut with } C \\ \\ \text{by a cut with } C \\ \\ \text{by contraction} \end{array}$$

Note that although B or C could be Σ_{k+1} in $(A \vee B) \wedge (A \vee C)$, they can only be Π_k in $A \vee (B \wedge C)$. Hence $\neg C$ is at most Σ_{k+1}. Thus the second line, the most complex formula of the derivation, is Π_{k+2}. ∎

3.7 Simulation of bounded depth sequent calculi

Our aim now is to prove that our formalization of a bounded depth propositional proof is equivalent to the standard ones based on sequent calculi. To this end we will again simulated the Tait calculus. In this calculus restricted to depth-k, sequents are sets of Π_k and Σ_k formulas. The interpretation of a sequents is the disjunction of the formulas in it, so they represent Σ_{k+1} formulas. A set of sequents that appear in a proof can be represented by a conjunction of the Σ_{k+1} formulas. Thus we get a formula of complexity at most Π_{k+2}. Therefore the Symmetric Calculus system corresponding to a depth k sequent proof system is the system based on Π^s_{k+2} proofs.

Proposition 3.7 *For every $k \geq 1$, Π^s_{k+2}-Symmetric Calculus is polynomially equivalent to depth-k Sequent Calculus.*

Proof.

1. We will show how to simulate the depth-k Tait Calculus. The Tait Calculus is formalized as follows (for more details, see [8]). Negations are only at variables, so $\neg A$ is the same formula as we defined in subsection 3.1. The logical axioms are the sequents of the form $\Gamma, p, \neg p$, the rules are disjunction introduction, conjunction introduction, and cut. They will be simulated by dual resolution, weakening, the derived rule for distributive law of the form (2), and cut respectively. We have shown how to simulate the distributive laws and cut. It only remains to explain the technical issue concerning the representation of formulas and sequents. The complication is that we have to use stratified formulas.

For $\Pi_k, \Sigma_{k-1}, \Pi_{k-2}, \Sigma_{k-3}, \ldots$ formulas, we will use $\Pi^s_k, \Sigma^s_{k-1}, \Pi^s_{k-2}, \Sigma^s_{k-3}, \ldots$ stratified formulas; a particular representation is not important, because any such representation can be transformed to any other. For $\Sigma_k, \Pi_{k-1}, \Sigma_{k-2}, \Pi_{k-3}, \ldots$ formulas, we will use $\Sigma^s_{k+1}, \Pi^s_k, \Sigma^s_{k-1}$, $\Pi^s_{k-2}, \ldots$ stratified formulas obtained by shifting a stratified representation to higher level by adding unary conjunctions or disjunctions to literals. Negations $\neg A$ are defined as in section 3.3.

A sequent Γ will be represented as follows. First we represent formulas of Γ as described above. Then we pad every formula that has complexity smaller than Π_k to level Π_k by adding unary $\wedge$s and $\vee$s, say, on the top. Finally, we form a Σ^s_{k+1} disjunction from these formulas. If the sequent consists of a single Π_k formula, or formula of smaller complexity, the top disjunction in the Σ^s_{k+1} stratified formula will be unary. The Σ^s_{k+1} formulas representing Σ_k formulas are disjunctions of Π^s_k formulas and as such they will become parts of the Σ^s_{k+1} disjunction representing the sequent.

The simulation of a sequent proof is as follows. We represent the initial sequents by a conjunction of the stratified Σ^s_{k+1} formulas. Then for every line in the proof we take the conjunction of all stratified formulas that represent sequents derived up to this line. If needed, we insert between two consecutive formulas a proof that simulates conjunction introduction, or cut. Finally, we use weakenings to remove all formulas except the one that we want to prove. It will be, of course, padded to the level Π^s_{k+2}.

2. Now we consider the opposite simulation. Given a Π^s_{k+2} proof $A_1, \ldots, A_n$, we transform every formula A_i of the proof into a set of sequents S_i with formulas of complexity at most

Π_k: we interpret the Σ^s_{k+1} disjunctions of A_i as sequents of formulas obtained from the Π^s_k subformulas of these disjunctions by omitting the unary $\wedge$s and $\vee$s if there are any. The disjunctions may contain multiple copies of Π^s_k formulas, but this will not be reflected in the sets S_j; they are sets, not multisets. Furthermore, we omit the truth constants $\bot$ and $\top$. We will show that the sequents of S_{j+1} can be proved from the sequents of S_j by polynomial size proofs of depth k.

First we consider the case when a rule R of the Symmetric Calculus is applied to a formula A_i of the proof, not to its subformula. Since A_i is a premise of a rule and its main connective is $\wedge$, R can only be a rule from the right column. If R is an instance of cloning there is no extra step in the simulation, because the sequent S_j representing A_i can be used repeatedly. We also do not have to simulate the rules for truth constants, because they do not appear in the sequents S_j. The $\wedge$-version of weakening (conjunction elimination) need not be simulated, because we keep all derived sequents in the Sequent Calculus. Finally, resolution is simulated by cut.

Suppose a rule R of the Symmetric Calculus is applied to a Σ^s_{k+1} subformula B of A_i. Since the main connective of B is $\vee$, R can only be a rule from the left column. Contraction is simulated by contraction in the Sequent Calculus, $\bot$ is not used in the Sequent Calculus, and weakening is simulated by weakening in the Sequent Calculus (which is not among the rules of the Tait Calculus, but can be easily simulated). The only rule that needs special treatment is dual resolution

$$\frac{\bigwedge_i C_i \wedge \top \wedge \bigwedge_j D_j}{(\bigwedge_i C_i \wedge p) \vee (\bigwedge_j D_j \wedge \neg p)}.$$

The formulas $\bigwedge_i C_i$ and $\bigwedge_j D_j$ are Π^s_k, hence they are simulated by formulas C and D of complexity at most Π_k. We need to show the following derivation:

$$\frac{\Gamma, C \wedge D}{\Gamma, C \wedge p, D \wedge \neg p}.$$

This is easy—it suffices to derive the sequent

$$\neg C, \neg D, C \wedge p, D \wedge \neg p,$$

from which we get $\neg(C \wedge D), C \wedge p, D \wedge \neg p$ by disjunction introduction and then we can apply cut to get what we need: $C \wedge p, D \wedge \neg p$. To derive the sequent, first derive sequents $\neg C, C$ and $\neg D, D$ and $p, \neg p$. Then apply conjunction introductions.

When a rule of the Symmetric Calculus is applied to a subformula of complexity Π^s_k or lower, then we proceed as follows. Let B be the Π^s_k subformula to whose subformula the rule is applied and let C be B after the rule is applied. Let $\hat{B}$ and $\hat{C}$ be the formulas that represent C and B in the Sequent Calculus. Then we first prove an auxiliary sequent $\neg\hat{B}, \hat{C}$ and then we use cut to obtain $\hat{C}$.

We leave the construction of $\neg\hat{B}, \hat{C}$ to the reader. To construct $\neg\hat{B}, \hat{C}$ in the case when the rule is applied B itself, not to a proper subformula of B, use the same argument as we used for dual resolution. If the application is deeper, use induction on the complexity of the formulas. ∎

3.8 Deep inferences using axioms

Suppose we want to derive a formula from axioms. The axioms are typically clauses of a CNF formula from which we want to derive contradiction. So we start with the conjunction of the axioms and gradually extend the conjunction by adding new derived formulas. In this process we need to do deep inferences *using axioms*. E.g., we have an axiom $\neg A \vee B$ (representing $A \to B$) and there is an occurrence of A deep inside of the currently derived formula. We want to replace A by B there. To this end we need to insert a copy of $\neg A \vee B$ next to the occurrence of A, so that on the position of this occurrence we will get $(\neg A \vee B) \wedge A$. Then we can use cut to reduce it to B. The following lemma shows that a padded formula A can be inserted into a disjunction.

Lemma 3.8 *Let* $\vee(A) \wedge (B_1 \vee B_2 \vee \cdots \vee B_n)$ *and* $\wedge((A \wedge B_1) \vee B_2 \vee \cdots \vee B_n)$ *be* Π^s_k *formulas. Then*

$$\vee(A) \wedge (B_1 \vee B_2 \vee \cdots \vee B_n) \ \vdash \wedge((A \wedge B_1) \vee B_2 \vee \cdots \vee B_n). \tag{4}$$

can be efficiently simulated.

Proof. This is just a weaker version of the distributive law, see Lemma 3.2, and it is proved in the same way. ∎

By iterating this lemma, we may insert A as deeper and deeper until all unary padding is removed from it.

4 Games

In this section we will introduce a new kind of games that we will use to characterize the interpolation pairs of bounded depth sequent calculi. First we describe the games in an intuitive way. The formal definition is in the next section.

We start with a concept that is a general form of many combinatorial games and it will be the bottom layer in our hierarchy of games. Such a game has two numerical parameters n, the length of the game and m, the number of symbols. In general, m can be exponential in n, but we prefer to imagine that it is polynomially bounded. The actual relationship will depend on applications. A game of this type is given by

1. sets of symbols $A_1, A_2, \ldots, A_n \subseteq [m]$,
2. transition functions $T_0 : \{0,1\} \to A_1$, $T_i : \{0,1\} \times A_i \to A_{i+1}$, $i = 1, \ldots, n-1$, and
3. $W \subseteq A_n$, a set of winning symbols.

The game is played by two players, Player I and Player II, who alternate in choosing one of the the actions $T_0(0)$ or $T_0(1)$ and, for $i > 1$, $T_i(0, x)$ or $T_i(1, x)$, where x is the symbol that the previous player played. They play until they produce a sequence of symbols of length n. Player I wins if the last symbol played is an element of W, otherwise Player II wins.

It is possible to determine who has a winning strategy by a computation that runs in time polynomial in n and m: for $i = n, n-1 \ldots 1$ compute inductively the set of winning symbols in step i. In fact, if we remove W from such a structure, we can view it as a monotone Boolean circuit where the possible inputs are (strings encoding) sets $W \subseteq A_n$. For a given W, the circuit outputs 1 iff in the game with W, Player I has a winning strategy.

In order to motivate our generalization we give an alternative definition of these games. Such a game will simply be given by a *nondeterministic automaton* T with the set of states A and a set of accepting states $W \subseteq A$. We will again assume that there are always only two possible action the automaton can do. The players alternate in choosing one of the possible actions at each step. This is not literally equivalent to the previous definition, because it corresponds to the case when $A_1 = \cdots = A_n$, but it is only a minor modification.

Our games have another numerical parameter k, called *the depth of the game*. The game will again be given by a nondeterministic automaton T, but now it will also use a tape with n squares. T reads the symbol on the currently visited square, moves to an adjacent square, reads the symbol there, and rewrites it. We assume that the only thing that the automaton can remember is the symbol that it has just read. T starts at the leftmost square and moves to the right until it reaches the nth square. It ends there if $k = 1$, otherwise it reverses the direction and goes to the left. At the first square it stops if $k = 2$, otherwise it reverses the direction, and continues in this manner until it passes the tape k-times. As in the previous definition, players alternate in controlling the automaton and the set of winning symbols is some subset W of symbols.

In order to simplify the formal definition of the games, we will assume that the set of symbols are the same for each step, and we will have only two transitions functions, one for the directions from the left to the right, and one for the opposite direction, and the first symbol played will be fixed.

Clearly, the tape plays no role when $k = 1$, but it is important if $k \geq 2$. Specifically, it is not possible to use the simple backtracking method to decide who has a winning strategy in polynomial time for $k \geq 2$. We think that, in fact, it is not possible to decide it using any polynomial time algorithm.

We will be interested in a special kind of strategies called *positional strategies*. A positional strategy is a set of rules that instructs a player which action to choose based solely on the current state of the automaton and the content of the square it reads. This restricts the class of strategies significantly, so it is possible that a player has a general winning strategy, but no positional winning strategy. The advantage of positional strategies is that they have concise descriptions, polynomial in n and m, where the degree depends on the depth k, and that, given such a strategy, one can check in polynomial time that it is a winning strategy.

4.1 Definition of games and positional strategies

In order to simplify the formalization, we will assume, w.l.o.g., that the game starts by Player I rewriting the *second* symbol on the tape.

Definition 1 *A* game of depth k *is given by the length of a round* n*, the number of rounds* k*, a finite alphabet* A *with a distinguished symbol* $\Lambda \in A$*, two functions, the* transition functions *or* legal moves *of the game,*

$$\overrightarrow{T} : \{0,1\} \times A \times A \to A,$$

$$\overleftarrow{T} : \{0,1\} \times A \times A \to A,$$

and a set of winning symbols $W \subseteq A$*.*

We will denote by T the pair $(\overrightarrow{T}, \overleftarrow{T})$.

The play—playing the game. The play starts with a string $\bar{a}$ of Λs of length n, viewed as a tape with n squares in which Λ is printed. Player I starts by choosing $h \in \{0,1\}$ and replacing the second symbol by $\overrightarrow{T}(h, \Lambda, \Lambda)$. Then players alternate and replace the $i+1$-st symbol a_{i+1} of $\bar{a}$ (which is just Λ in the first round) with $\overrightarrow{T}(h, a_i, a_{i+1})$, where h is chosen by Player I if i is odd and Player II if i is even. They go on until the end of the string. If $k \geq 2$ they go back using $\overleftarrow{T}$ and so on. When they reverse direction, the last symbol, respectively the first symbol, is not rewritten again. This means that if they arrive at the end of the tape and the symbols on the tape are $a_1 \dots a_{n-1} a_n$, then the player whose turn it is rewrites a_{n-1} to $\overleftarrow{T}(h, a_{n-1}, a_n)$ for some $h \in \{0,1\}$ and they continue in the direction to the left. The same happens at the beginning of the tape.[5]

After k rounds the play ends and Player I wins if the last played symbol is in W, otherwise Player II wins.

Definition 2 *A* positional strategy for Player I *is a pair* $(\overrightarrow{\sigma}, \overleftarrow{\sigma})$ *such that*

$$\overrightarrow{\sigma} : [k] \times [n]_{odd} \times A \times A \to A,$$

$$\overleftarrow{\sigma} : [k] \times [n]_{odd} \times A \times A \to A,$$

where $[n]_{odd}$ *is the set of odd numbers* $\leq n$*. Furthermore,* $\overrightarrow{\sigma}$ *and* $\overleftarrow{\sigma}$ *must be compatible with* T*, which means that for every* r, i, b, c*, there exists an* h *such that* $\overrightarrow{\sigma}(r, i, b, c) = \overrightarrow{T}(h, b, c)$ *and similarly for* $\overleftarrow{\sigma}$ *and* $\overleftarrow{T}$*.*

A positional strategy for Player II *is a pair* $\sigma = (\overrightarrow{\sigma}, \overleftarrow{\sigma})$ *defined in a similar way with* $[n]_{odd}$ *replaced by* $[n]_{even}$*.*

For better readability, we will write r and i as subscripts, e.g., $\overrightarrow{\sigma}_{r,i}(b, c)$. The arrows above σ are, clearly, determined by the index r, but we prefer to keep the arrows to stress the direction the strategy is used.

Instead of viewing the game as rewriting symbols on a tape, it is better to imagine that the players choose symbols in a $k \times n$ matrix in a zig-zag way, and the admissible choices are given by the previously played symbol and the symbol above the square that is to be filled. We will call such a partially filled matrix a *history* of the play. More precisely, a *history up*

[5] This rule of the game is not essential, but it makes formalization simpler.

to step (r, i) is the record of a game played up to this step with the rest of the $k \times n$ matrix filled with Λs. We note a couple of useful properties of the history matrix M.

$$M_{1,1} = \Lambda, \tag{5}$$

$$M_{2r,n} = M_{2r-1,n} \text{ for all } 1 \leq r \leq k/2, \tag{6}$$

$$M_{2r+1,1} = M_{2r,1} \text{ for all } 1 \leq r < k/2. \tag{7}$$

The latter two express that the players do not rewrite the last/first symbol when reversing the direction of playing.

We will call a column vector of at most k symbols of elements of A a *position* and view it as the first r, $r \leq k$, entries of a column of a $k \times n$ matrix. Given a history M we say that $\bar{a}$ is the (r, i)-position if $\bar{a}$ is the column vector $(M_{1,i}, \ldots, M_{r,i})^{\intercal}$.[6]

Example. Consider a 3-round game in which a play reached a position $(3, i)$. Let this be the history up to this position:

Λ	$M_{1,2}$	$\ldots$	$\rightarrow$	$\ldots$	$M_{1,i}$	$M_{1,i+1}$	$\ldots$	$M_{1,n}$
$M_{2,1}$	$\ldots$	$\ldots$	$\leftarrow$	$\ldots$	$M_{2,i}$	$M_{2,i+1}$	$\ldots$	$M_{2,n}(= M_{1,n})$
$M_{3,1}(= M_{2,1})$	$\ldots$	$\ldots$	$\rightarrow$	$\ldots$	$M_{3,i}$			

Then the next symbol played, i.e., $M_{3,i+1}$, must be either $\overrightarrow{T}(0, M_{3,i}, M_{2,i+1})$ or $\overrightarrow{T}(1, M_{3,i}, M_{2,i+1})$. If moreover it is the turn of Player I and he uses a strategy σ^I, then $M_{3,i+1} = \overrightarrow{\sigma}^I_{3,i}(M_{3,i}, M_{2,i+1})$ and the same for Player II and strategy σ^{II}.

In general, $\overrightarrow{T}$ and $\overrightarrow{\sigma}_{r,i}$ are always applied to $M_{r,i}, M_{r-1,i+1}$ (for r odd), and $\overleftarrow{T}$ and $\overleftarrow{\sigma}_{r,i+1}$ are always applied to $M_{r-1,i}, M_{r,i+1}$ (for r even).

Let a game of depth k be given. Let $r, s \leq k$, $i \leq n$, $\bar{a} \in A^r$, $\bar{b} \in A^s$. We say that $\bar{a}$ and $\bar{b}$ are *T-compatible on steps i and $i+1$*, if they are compatible with the transition functions applied at particular places. Given a strategy σ, we say that $\bar{a}$ and $\bar{b}$ are *σ-compatible on steps i and $i+1$*, if they are T-compatible and moreover they are compatible with σ applied at particular places. A formal definition of T and σ compatibilities would be a long list of cases and formulas, while the concept is intuitively clear. Therefore we state formally only one case.

If σ is a strategy for Player I and $i < n$, $r \leq s$, i and r even, then $\bar{a}$ and $\bar{b}$ are *σ-compatible on steps i and $i+1$* if there exits $h_1, h_3, \ldots, h_{r-1} \in \{0, 1\}$ such that

$$\begin{array}{rcl} b_1 & = & \overrightarrow{T}(h_1, a_1, \Lambda), \\ a_2 & = & \overleftarrow{\sigma}_{2,i+1}(a_1, b_2), \\ b_3 & = & \overrightarrow{T}(h_3, a_3, b_2), \\ a_4 & = & \overleftarrow{\sigma}_{4,i+1}(a_3, b_4), \\ & \ldots & \\ b_{r-1} & = & \overrightarrow{T}(h_{r-1}, a_{r-1}, b_{r-2}), \\ a_r & = & \overleftarrow{\sigma}_{r,i+1}(a_{r-1}, b_r). \end{array} \tag{8}$$

[6] $^{\intercal}$ denotes transposition of vectors (here a row vector to a column vector).

We will abbreviate *"T/σ-compatible on positions i and $i+1$"* by *"$T/\sigma, i$-compatible"* and omit i if it is determined by the context.

For $r \leq k$, $i \leq n$, $\bar{a} \in A^r$ and σ a strategy, we say that a position $\bar{a}$ is (r, i, σ)-*reachable*, if there is a history of a play played according to σ in which $\bar{a}$ is the (r, i)-position. We will denote by $R^\sigma_{r,i}$ the set of (r, i, σ)-reachable positions.

Lemma 4.1 *For k constant, given a positional strategy for Player I (respectively Player II), represented as a string of $kn|A|^2$ symbols from A, it is possible to decide in polynomial time if it is a winning strategy for Player I (respectively Player II).*

Proof. We will show that sets $R^\sigma_{r,i}$ satisfy the following inductive conditions.

For $r = 1$,

$$a \in R^\sigma_{1,i+1} \;\equiv\; \exists b \in R^\sigma_{1,i}(b, a \text{ are } \sigma\text{-compatible}). \tag{9}$$

If, e.g., i is odd and σ is a strategy for Player I, this means that $\overrightarrow{\sigma}_{1,i}(a, \Lambda) = b$.

For odd $r \geq 3$,

$$\bar{a} \in R^\sigma_{r,i+1} \;\equiv\; (a_1, \ldots, a_{r-1}) \in R^\sigma_{r-1,i+1} \wedge \exists \bar{b} \in R^\sigma_{r,i}(\bar{b}, \bar{a} \text{ are } \sigma\text{-compatible}). \tag{10}$$

For even $r \geq 2$,

$$\bar{a} \in R^\sigma_{r,i} \;\equiv\; (a_1, \ldots, a_{r-1}) \in R^\sigma_{r-1,i} \wedge \exists \bar{b} \in R^\sigma_{r,i+1}(\bar{a}, \bar{b} \text{ are } \sigma\text{-compatible}). \tag{11}$$

We will only prove (10), the proof of (11) is similar, and (9) is trivial.

Suppose $\bar{a} \in R^\sigma_{r,i+1}$. Let M be the history of a play in which $\bar{a}$ is the $(r, i+1)$-position. Let $\bar{b}$ be the (r, i) position in M. Then, clearly, the right-hand side is satisfied.

We will use a "hybrid argument" to prove the opposite implication. Suppose that $(a_1, \ldots, a_{r-1}) \in R^\sigma_{r-1,i+1}$, $\bar{b} \in R^\sigma_{r,i}$ and $\bar{b}, \bar{a}$ are σ-compatible. We will consider histories of plays played according to σ. Let M be a history of a play up to the point (r, i) in which $\bar{b}$ is the (r, i)-position and N be a history of a play up to the point $(r-1, n)$ in which $(a_1, \ldots, a_{r-1})^\intercal$ is the $(r-1, i+1)$-position. Let M' be the matrix consisting of the first i columns of M and N' be the matrix consisting of the last $n-i$ columns of N. It is not difficult to see then that $M'N'$ is a history of a play up to (r, i) in which $\bar{b}$ is the (r, i)-position and $(a_1, \ldots, a_{r-1})^\intercal$ is the $(r-1, i+1)$-position. Now we can continue the play one more step to obtain $\bar{a}$ as the $(r, i+1)$-position. This finishes the proof of (10).

Conditions (9,10,11) give us a recursive procedure to compute the sets $R^\sigma_{r,i}$. The procedure has kn steps and at each step we only need to consider at most $|A|^{2k}$ positions. Since k is constant, this gives us a polynomial time algorithm. Finally, we only need to check that the last set contains only winning positions of the player in question. ∎

4.2 Modifications of the game

There are various modifications of the definition of the game that are equivalent in the sense that they can efficiently simulate each other.[7]

[7] An efficient simulation of games of type t_1 by games of type t_2 is a polynomial time algorithm such that given a game G_1 of type t_1, it constructs a game G_2 of type t_2 such that G_2 has the same depth as G_1 and Player I (resp. Player II) has a positional winning strategy in G_2 iff he/she has in G_1.

First we note that we can assume w.l.o.g. that the transition functions also depend on the position on the tape. To simulate such a game by one whose transition function does not depend on the position, we let the players encode the position in the printed symbols. Thus if the game proceeds to the right and a player knows that he is on the ith position of the tape, he will encode the number $i+1$ in the symbol he will play.

We also do not have to insist that players alternate regularly. We may even allow steps that are done without players deciding anything. Furthermore, whose turn it is to move may also depend on the symbol to which they arrive.

Another modification, one of those that we are going to use, is that the players do not have to go to the ends of the tapes and can reverse the direction at other places on the tape. If we want to satisfy the original definition, we may introduce an auxiliary symbol and let the players play this symbol until the end of the row and then back until they get to the place where they were supposed to pass to the next row.

We leave the formal statements and simulations to the readers, because they are easy, but may be complicated to write down formally.

4.3 The disjoint NP pairs of the games and the main theorem

We can now define the disjoint NP pairs of the games.

Definition 3 *For $k \geq 1$,*

$$A_k := \{G \mid G \text{ is a game of depth } k \text{ in which Player I has a positional winning strategy}\},$$

$$B_k := \{G \mid G \text{ is a game of depth } k \text{ in which Player II has a positional winning strategy}\}.$$

The disjointness is obvious, the membership in NP is a consequence of Lemma 4.1.

For $k = 1$, every strategy is positional and one can decide in polynomial time who has a winning strategy by backtracking winning positions. Also note that if we fix the transition function $\overrightarrow{T}$ and view sets W as inputs, then such a game schema is essentially a monotone Boolean circuit that for a given W decides who has a winning strategy (cf. Section 8).

For $k \geq 2$, one can easily construct games in which neither player has a positional wining strategy. For games of depth 2, it is open if one can decide in polynomial time who has a positional winning strategy given the promise that one of the players has such a strategy.

In the following two sections we will prove our main theorem.

Theorem 4.2 (Main Theorem) *For $k \geq 1$, the pair (A_k, B_k) is polynomially equivalent to the interpolation pair of the depth $k-1$ sequent calculus.*

We have stated the theorem for the depth $k-1$ sequent calculus, but in the proof we will use the Π^s_{k+1}-Symmetric Calculus. By Proposition 2.1, this theorem also implies that for $k \geq 2$, (A_k, B_k) is polynomially equivalent to the canonical pair of the depth $k-2$ Sequent Calculus.

To prove the theorem we need to show two reductions:

1. from (A_k, B_k) to the interpolation pair of the Π^s_{k+1} Symmetric Calculus; this is Lemma 5.1, and

2. from the interpolation pair of Π^s_{k+1} Symmetric Calculus to (A_k, B_k); this is Lemma 6.1.

5 Proofs from games

In this section we will construct, for every $k \geq 1$, a reduction from the pair (A_k, B_k) to the interpolation pair of depth Π^s_{k+1}-Symmetric Calculus.

Lemma 5.1 *Given a game G of depth k, one can construct in polynomial time formulas $\Phi(\bar{x})$ and $\Psi(\bar{y})$ with disjoint sets of variables $\bar{x}$, $\bar{y}$, and a Π^s_{k+1} refutation D of $\Phi(\bar{x}) \wedge \Psi(\bar{y})$ such that $\Phi(\bar{x})$ is satisfiable when Player I has a positional winning strategy in G and $\Psi(\bar{y})$ is satisfiable when Player II has a positional winning strategy in G.*

Thus (A_k, B_k) is reducible to the interpolation pair of Π^s_{k+1}-Symmetric Calculus and, by Proposition 3.7, also to the interpolation pair of the depth $k-1$ sequent calculus.

We will prove this lemma by formalizing the statements that $\bar{x}$ is a positional strategy for Player I and $\bar{y}$ is a positional strategy for Player II, and constructing a depth Π^s_{k+1} proof that it is impossible that both strategies are winning.

5.1 The formula

Let a game G of length n and depth k be given. In order to formalize strategies σ^I and σ^{II} we will use not only variables for the strategies but also sets of reachable positions. Thus we will have variables both for elements of the strategies and elements of $R^I_{r,i}$ and $R^{II}_{r,i}$, $r = 1, \ldots, k$, $i = 1, \ldots, n$, where $R^I_{r,i}$ and $R^{II}_{r,i}$ are abbreviations for $R^{\sigma^I}_{r,i}$ and $R^{\sigma^{II}}_{r,i}$.

We will use the convention that a propositional variable representing the truth of a relation $P(a_1, \ldots, a_t)$ is denoted by $[P(a_1, \ldots, a_t)]$ to represent propositions about strategies: $[\sigma^I_{r,i}(a, b) = c]$, $[\sigma^{II}_{r,i}(a, b) = c]$. For $\bar{a} \in R^I_{r,i}$ and $\bar{b} \in R^{II}_{r,i}$, we will denote the variables simply by $R^I_{r,i}(\bar{a})$ and $R^{II}_{r,i}(\bar{b})$ in order not to overload notation with unnecessary symbols. One should keep in mind that in this notation the relation $R^I_{r,i}$ is indeterminate while elements $a_1, \ldots, a_r$ are fixed, so the propositional variables are indexed by $r, i, a_1, \ldots, a_t$; and this also concerns R^{II}, σ^I, and σ^{II}.

We will use $\times$ to refer either to Player I, or Player II and $*$ to refer to either direction $\rightarrow$, or $\leftarrow$. For σ, we can omit the index referring to a player, because the player is determined by the index i (i odd is for I and i even for II), and we can also omit arrows, because they are determined by the indices r of rows.

We will use implication $A \rightarrow B$ to represent $\neg A \vee B$.

Variables of the formula

1. $[\sigma_{r,i}(a,b)=c]$ for $r=1,\dots,k$, $i=1,\dots,n$, $a,b,c\in A$ (the variables for the strategies of Players I and II),

2. $R^I_{r,i}(\bar a)$, $R^{II}_{r,i}(\bar a)$ for $\bar a\in A^r$, $r=1,\dots,k$, $i=1,\dots,n$ (the variables for the sets of reachable positions).

Clauses of the formula

1. Clauses saying that "σ is a positional strategy"

$$\bigvee_c [\sigma_{r,i}(a,b)=c]$$

for $r=1,\dots,k$, $i=1,\dots,n$, and where the disjunction is over c such that a,b,c are T compatible.[8]

2. Clauses expressing (5), (6), and (7):

$$R^I_{1,1}(\Lambda),\ R^{II}_{1,1}(\Lambda) \tag{12}$$

$$R^\times_{2r-1,n}(a_1,\dots,a_{2r-1}) \equiv R^\times_{2r,n}(a_1,\dots,a_{2r-1},a_{2r-1}) \text{ for all } 1\le r\le k/2, \tag{13}$$

$$R^\times_{2r,1}(a_1,\dots,a_{2r}) \equiv R^\times_{2r+1,1}(a_1,\dots,a_{2r},a_{2r}) \text{ for all } 1\le r< k/2. \tag{14}$$

3. Clauses expressing the inductive conditions (10),(11) for $R^\times_{r,i}$.

First we need to express that $\bar a$ and $\bar b$ are $\sigma^\times$-compatible. Consider Player II, i odd, r even, and $\bar a,\bar b\in A^r$. Then "$\bar a,\bar b$ are σ^I-compatible" is defined by the conditions (8). Hence if there are no $h_1,h_3,\dots,h_{r-1}\in\{0,1\}$ such that $b_1=\overrightarrow{T}(h_1,a_1,\Lambda)$, $b_3=\overrightarrow{T}(h_3,a_3,b_2)$, $\dots$, $b_{r-1}=\overrightarrow{T}(h_{r-1},a_{r-1},b_{r-2})$, then a,b are not σ^I-compatible. In this case compatibility is expressed by $\bot$. Otherwise they are σ^I-compatible iff $a_2=\overleftarrow{\sigma}^I_{2,i+1}(b_2,a_1)$, $a_4=\overleftarrow{\sigma}^I_{4,i+1}(b_4,a_3)$, $\dots$, $a_r=\overleftarrow{\sigma}^I_{r,i+1}(b_r,a_{r-1})$. In this case compatibility can be expressed by a conjunction of propositional variables

$$[a_2=\overleftarrow{\sigma}^I_{2,i+1}(b_2,a_1)]\wedge[a_4=\overleftarrow{\sigma}^I_{4,i+1}(b_4,a_3)]\wedge\dots\wedge[a_r=\overleftarrow{\sigma}^I_{r,i+1}(b_r,a_{r-1})].$$

For Player I and other r and i, it is similar.

This enables us to express formulas (10) and (11) by small propositional formulas, but not small CNFs. Fortunately, we only need implications from the right to the left. In the case of r odd, i.e. (10), it is

$$(a_1,\dots,a_{r-1})\in R^\times_{r-1,i+1}\wedge\exists\bar b\in R^\times_{r,i}(\bar b,\bar a \text{ are } \sigma,i\text{-compatible})\to\bar a\in R^\times_{r,i+1},$$

[8]The formula, in fact, expresses that σs are *total relations* defined properly, i.e., we do not formalize that they are *functions*. Recall that "a,b,c are T compatible" means $T^*(h,a,b)=c$ for some $h\in\{0,1\}$.

which is equivalent to

$$\forall \bar{b}((a_1,\ldots,a_{r-1}) \in R^{\times}_{r-1,i+1} \wedge \bar{b} \in R^{\times}_{r,i} \wedge (\bar{b},\bar{a} \text{ are } \sigma,i\text{-compatible}) \to \bar{a} \in R^{\times}_{r,i+1}).$$

This can be represented by a set of clauses, one for every $b \in A^r$. We leave the case of $r = 1$ and $r \geq 3$ odd to the reader.

> *Example.* Let r and i be odd, let $\bar{a},\bar{b} \in A^3$. Suppose that $b_2 = \overleftarrow{T}(h,a_2,b_1)$ for some $h \in \{0,1\}$. We consider the situation where $(b_1,b_2,b_3)^\intercal$ is the ith column of a history matrix and $(a_1,a_2,a_3)^\intercal$ is the $i+1$st column. Then we have the following clause
>
> $$R^I_{3,i}(b_1b_2b_3) \wedge R^I_{2,i+1}(a_1a_2) \wedge [a_1 = \overrightarrow{\sigma}^I_{1,i}(b_1,\Lambda)] \wedge [a_3 = \overrightarrow{\sigma}^I_{3,i}(b_3,a_2)] \to R^I_{3,i+1}(a_1a_2a_3).$$

4. Clauses saying that the strategies of both players are winning. Strategy σ^I is winning for Player I if all final positions reachable using σ^I are winning. If k is odd, this means

$$\forall a_k\ ((a_1 \ldots a_k) \in R^I_{k,n} \to a_k \in W).$$

This is expressed in the propositional calculus by

$$\bigwedge_{a_k \notin W} \neg R^I_{k,n}(a_1,\ldots,a_k) \tag{15}$$

Similarly, for Player II, still assuming k odd, we get

$$\bigwedge_{a_k \in W} \neg R^{II}_{k,n}(a_1,\ldots,a_k) \tag{16}$$

In the case of k even, the index n at R is replaced by 1.

This defines CNF formulas $\Phi_k(\bar{x})$ and $\Psi_k(\bar{y})$, where $\bar{x}$ are propositional variables $[\sigma^I_{r,i}(a,b) = c]$, $R^I_{r,i}(\bar{a})$, and $\bar{y}$ are propositional variables $[\sigma^{II}_{r,i}(a,b) = c]$, $R^{II}_{r,i}(\bar{a})$. The formula $\Phi_k(\bar{x}) \wedge \Psi_k(\bar{y})$ expresses a contradictory fact that both players have positional winning strategies.

5.2 The refutation of the formula

We will now construct a Π^s_{k+1} derivation of contradiction from the formula $\Phi_k(\bar{x}) \wedge \Psi_k(\bar{y})$ defined above.

Lemma 5.2 *One can construct in polynomial time a Π^s_{k+1}-refutation of the formula $\Phi_k(\bar{x}) \wedge \Psi_k(\bar{y})$.*

This lemma implies Lemma 5.1 because $\Phi_k(\bar{x})$ (respectively $\Psi_k(\bar{y})$) is a formalization the fact that Player I (Player II) has a winning strategy.

Before going into details, we will explain the essence of the proof. Formula $\Phi_k(\bar{x}) \wedge \Psi_k(\bar{y})$ says that it is impossible that both players have positional winning strategies. We use

positional strategies because we need formulas of certain complexity, but, clearly, there cannot be *any pair* of winning strategies for opposing players. The standard argument is that if we run the two strategies, then at the end only one player wins, so the two strategies cannot be winning. Let us try to formalize it and see why this argument cannot be used for depth d games for $d > 1$.

Let $S_{r,i}(x_1, \ldots, x_r)$ denote that position $(x_1, \ldots, x_r)^\intercal$ can be reached by playing strategies σ^I and σ^{II}. To get a contradiction, we need to show that there exists a position $(x_1, \ldots, x_d)^\intercal$ such that $S_{d,n}(x_1, \ldots, x_d)$ holds true if d is odd, and $S_{d,1}(x_1, \ldots, x_d)$ if d is even. Clearly, we have $S_{1,1}(\Lambda)$, thus $\exists x\ S_{1,1}(x)$. Applying σ^I we get $\exists x\ S_{1,2}(x)$, then using σ^{II} we get $\exists x\ S_{1,3}(x)$ and so on until we obtain $\exists x\ S_{1,n}(x)$. If the depth of the game $d = 1$, we are done, because a position in the last step cannot be winning for both players.

If $d > 1$ we would like to continue. By definition, if $S_{1,n}(x)$, then $S_{2,n}(x, x)$, thus we have $\exists x, y\ S_{2,n}(x, y)$. Now we want to prove $\exists x', y'\ S_{2,n-1}(x', y')$. Let x and y be such that $S_{2,n}(x, y)$. We know that $\exists x'\ S_{1,n-1}(x')$, but this is not enough; we need an x' such that $\sigma^\times(x') = x$ (where $\times$ is I or II depending on whether n is even or odd). So in order to be able to go back to a position in the first column, we need that

$$\exists x_1 \ldots \exists x_n (S_{1,1}(x_1) \wedge x_2 = \sigma^I(x_1) \wedge S_{1,2}(x_2) \wedge x_3 = \sigma^{II}(x_2) \wedge \ldots).$$

But if expressed as a propositional formula, it has exponential size, because the range of quantification is of size $|A|^n$.

What we can do instead is this. Observe that we have

$$\forall x \exists y (S_{1,n}(x) \to S_{2,n}(x, y)), \tag{17}$$

because for a given x, we can take $y = x$. Suppose w.l.o.g. that n is even. We can go back, to the left, with this formula. Suppose $S_{1,n-1}(x')$. Then $S_{1,n}(\sigma^\times(x'))$. From (17), we get a y such that $S_{2,n}(\sigma^\times(x'), y)$ and then we can conclude $S_{2,n-1}(x', \sigma^{\times'}(y))$ using (10) or (11). Thus we have shown $\forall x' \exists y' (S_{1,n-1}(x') \to S_{2,n-1}(x', y'))$. Repeating this argument we eventually get

$$\forall x'' \exists y'' (S_{1,1}(x'') \to S_{2,1}(x'', y'')). \tag{18}$$

Now recall that we have $\exists x''\ S_{1,1}(x)$, so we get $\exists x'' \exists y'' S_{2,1}(x'', y'')$. If $d = 2$, we get a contradiction, because $(x'', y'')^\intercal$ is the final position and as such it cannot be reached by both strategies.

If $d > 2$, this does not work, but we can use a formula with more quantifiers, specifically, formulas with d alternating quantifiers.

Let's have a look at the complexity of the formulas used in the proofs sketched above. For $d = 1$, we are aiming at Π^s_2 proofs, but formulas $\exists x\ S_{1,i}(x)$ translate to Σ^s_2, because $S_{1,i}$ is $R^I_{1,i} \wedge R^{II}_{1,i}$. We cannot use that proof as it stands, but we can turn it around and argue contrapositively. We start with $\bigwedge_x \neg S_{1,n}(x)$ and proceed to the left. Thus we obtain $\bigwedge_x \neg S_{1,1}(x)$, from which we get contradiction using $S_{1,1}(\Lambda)$.

The proof for odd $d \geq 3$ is similar except that we have to use a formula with more alternations of $\wedge$s and $\vee$s; see formula (35) below.

For $d = 2$, the proof above can be formalized as Π^s_3 proof. In general, for $d \geq 2$ even, we use formulas (29). However, the complexity of these formulas does not guarantee that the proof has the same depth; it is necessary to check it, which is a little tedious, but not difficult.

The rest of this subsection is devoted to the proof of Lemma 5.2.

Proof. Let a game of depth k be given. First we observe that from (15) and (16) we get, by weakening, all clauses

$$\neg R^I_{k,n}(\bar{a}) \vee \neg R^{II}_{k,n}(\bar{a}) \tag{19}$$

for all $\bar{a}$ if k is odd, and

$$\neg R^I_{k,1}(\bar{a}) \vee \neg R^{II}_{k,1}(\bar{a}) \tag{20}$$

if k is even. (We did not use these clauses to define our formula because they mix both types of variables.) We now consider cases according to the depth of the game.

Case $k = 1$. We need a Π^s_2 refutation, which is essentially a Resolution refutation. We will omit the index 1 of $R_{1,i}$ and the arrow $\rightarrow$ above σ, because the direction does not change in this case; we will also abbreviate $\sigma(a, \Lambda)$ by $\sigma(a)$, because the second argument is always the same.

First we show by induction for $i = n, n-1, \ldots, 1$ that clauses

$$\neg R^I_i(a) \vee \neg R^{II}_i(a) \tag{21}$$

are derivable for all $a \in A$.

We already have it for $i = n$ by (19).

Suppose we have (21) for $i + 1$ and we want to get it for i. Suppose moreover that i is odd. Then a, b is σ^I, i-compatible if $\sigma^I_i(a) = b$. Hence clauses of the formula that represent inductive conditions are

$$\neg R^I_i(b) \vee \neg[\sigma^I_i(b) = a] \vee R^I_{i+1}(a), \tag{22}$$

$$\neg R^{II}_i(b) \vee R^{II}_{i+1}(a) \tag{23}$$

for all T-compatible pairs a, b. From $\neg R^I_{i+1}(a) \vee \neg R^{II}_{i+1}(a)$, (22), and (23), we get by resolution

$$\neg R^I_i(b) \vee \neg R^{II}_i(b) \vee \neg[\sigma^I_i(b) = a] \tag{24}$$

We also have

$$\bigvee_a [\sigma^I_i(b) = a] \tag{25}$$

for all $b \in A$ where the disjunction is over all a such that b, a is T-compatible (see clauses of the formula). From (24), and (25) we get $\neg R^I_i(b) \vee \neg R^{II}_i(b)$ by several applications of resolution. Since for every $b \in A$ there is an $a \in A$ such that b, a are T-compatible, we get this formula for all $b \in A$. For i even, the proof is analogous.

For $i = 1$, (21) gives, in particular,

$$\neg R^I_1(\Lambda) \vee \neg R^{II}_1(\Lambda).$$

Since our formula contains clauses $R^I_1(\Lambda)$ and $R^{II}_1(\Lambda)$, we get a contradiction.

Case $k \geq 2$ even. We will abbreviate by

$$S_{r,i}(x_1 \dots x_r) \ := \ R^I_{r,i}(x_1 \dots x_r) \wedge R^{II}_{r,i}(x_1 \dots x_r).$$

This formula expresses that $(x_1 \dots x_r)^\intercal$ is reachable using both strategies. It is a Π^s_1 formula (a conjunction of literals). Further, we introduce an abbreviation for compatibility:

$$C_{r,i}(x_1 \dots x_r, y_1 \dots y_r) \ \equiv_{def} \ x_1 \dots x_r \text{ and } y_1 \dots y_r \text{ are both } \sigma^I \text{ and } \sigma^{II} \text{ } i\text{-compatible.}$$

Also this formula is Π^s_1. We will also need $C^I_{r,i}$ and $C^{II}_{r,i}$ representing σ^I, respectively σ^{II} compatibility. So $C_{r,i} \equiv C^I_{r,i} \wedge C^{II}_{r,i}$.

Lemma 5.3 *Let $1 \leq i < n$, $1 \leq r < k$, $\bar{u}, \bar{v} \in A^r$. Then the following formulas have polynomial size Π^s_3 proofs from the formula $\Phi_k(\bar{x}) \wedge \Psi_k(\bar{y})$ for all T-compatible pairs (a, b) and $(\bar{u}a, \bar{v}b)$.*

$$S_{1,i}(a) \to \bigvee_b (C_{1,i}(a, b) \wedge S_{1,i+1}(b)). \tag{26}$$

$$(C_{r,i}(\bar{u}, \bar{v}) \wedge S_{r+1,i}(\bar{u}a) \wedge S_{r,i+1}(\bar{v})) \to \bigvee_b (C_{r+1,i}(\bar{u}a, \bar{v}b) \wedge S_{r+1,i+1}(\bar{v}b)) \tag{27}$$

for r even.

$$(C_{r,i}(\bar{u}, \bar{v}) \wedge S_{r+1,i+1}(\bar{v}b) \wedge S_{r,i}(\bar{u})) \to \bigvee_a (C_{r+1,i}(\bar{u}a, \bar{v}b) \wedge S_{r+1,i}(\bar{u}a))) \tag{28}$$

for r odd.

Proof. We will only prove (27); the other two can be proved in the same way. Let r be even and assume w.l.o.g. that i is odd. Then our formula contains clauses (inductive conditions on R)

$$C^I_{r,i}(\bar{u}, \bar{v}) \wedge R^I_{r+1,i}(\bar{u}a) \wedge R^I_{r,i+1}(\bar{v}) \wedge [b = \overrightarrow{\sigma}^I_{r,i}(a, v_r)] \to R^I_{r+1,i+1}(\bar{v}b),$$
$$C^{II}_{r,i}(\bar{u}, \bar{v}) \wedge R^{II}_{r,i}(\bar{u}a) \wedge R^{II}_{r,i+1}(\bar{v}) \to R^{II}_{r,i}(\bar{u}a).$$

Further, by definition

$$C^I_{r,i}(\bar{u}, \bar{v}) \wedge [b = \overrightarrow{\sigma}^I_{r,i}(a, u_r)] \to C^I_{r+1,i}(\bar{u}a, \bar{v}b).$$

We also have

$$C^{II}_{r,i}(\bar{u}, \bar{v}) \wedge [b = \overrightarrow{\sigma}^I_{r,i}(a, u_r)] \to C^{II}_{r+1,i}(\bar{u}a, \bar{v}b),$$

because we assume T-compatibility of $\bar{u}a$ with $\bar{v}b$. Using conjunction introduction, see (2), we get for every T-compatible a and b,

$$C_{r,i}(\bar{u}, \bar{v}) \wedge S_{r+1,i}(\bar{u}a) \wedge S_{r,i+1}(\bar{v}) \wedge [b = \overrightarrow{\sigma}^I_{r,i}(a, u_r)] \to C_{r+1,i}(\bar{u}a, \bar{v}b) \wedge S_{r+1,i+1}(\bar{v}b).$$

By resolving with $\bigvee_b[b = \overrightarrow{\sigma}^I_{r,i}(a, u_r)]$ we get for every a,

$$C_{r,i}(\bar{u}, \bar{v}) \wedge S_{r+1,i}(\bar{u}a) \wedge S_{r,i+1}(\bar{v}) \to \bigvee_b (C_{r+1,i}(\bar{u}a, \bar{v}b) \wedge S_{r+1,i+1}(\bar{v}b)).$$

■

For $i = 1, \ldots, n$, we will denote by Δ_i the following formula

$$\bigwedge_{x_1}(S_{1,i}(x_1) \to \bigvee_{x_2}(S_{2,i}(x_1x_2) \wedge \bigwedge_{x_3}(S_{3,i}(x_1x_2x_3) \to \cdots \bigvee_{x_k} S_{k,i}(x_1 \ldots x_k)))). \tag{29}$$

Note that Δ_i is Π^s_{k+1}. Our plan is:

1. Prove Δ_n.
2. Construct proofs of $\Delta_{i+1} \vdash \Delta_i$ for $i = n-1, \ldots, 1$. Thus we get Δ_1.
3. Then it would suffice to prove $\neg\Delta_1$, but the complexity of this formula is Σ^s_{k+1} which is too much. Instead, one can derive

   $$\bigvee_{x_1}(S_{1,i}(x_1) \wedge \bigwedge_{x_2}(S_{2,i}(x_1x_2) \to \bigvee_{x_3}(S_{3,i}(x_1x_2x_3) \wedge \cdots \bigvee_{x_{k-1}} S_{k-1,i}(x_1 \ldots x_{k-1}))))$$

   and use it with Δ_1 to derive

   $$\bigvee_{x_1,x_2,\ldots,x_k} S_{k,1}(x_1, x_2, \ldots, x_k). \tag{30}$$

   But it is easier to derive (30) from Δ_1 only using clauses of the formula $\Phi(\bar{x}) \wedge \Psi(\bar{y})$.
4. Finally we derive contradiction from (30) using cuts with formulas (20), which are $\neg S_{k,1}(\bar{a})$ in the new notation.

To finish the proof of Lemma 5.2 it remains to construct proofs of formulas in items 1, 2 and 3.

1. We will prove Δ_n. From clauses (13) (where we only need the implications from the left to the right) we get, using conjunction introduction,

$$\bigwedge_{x_1}(S_{1,n}(x_1) \to (S_{2,n}(x_1x_1) \wedge \bigwedge_{x_3}(S_{3,n}(x_1x_1x_3) \to (S_{4,n}(x_1x_1x_3x_3) \wedge \cdots \bigvee_{x_k} S_{k,n}(x_1x_1 \ldots x_{k-1}x_{k-1}))))).$$

Then Δ_n follows by weakening (but we can use this formula as well to derive Δ_{n-1}).

2. We will now prove $\Delta_{i+1} \vdash \Delta_i$. This is more complicated and we need to use some abbreviations:

$$A_r := S_{r,i}(x_1 \ldots x_r),\ B_r := S_{r,i+1}(y_1 \ldots y_r),\ C_r = C_{r,i}(x_1 \ldots x_r, y_1 \ldots y_r).$$

First we rewrite formulas (26), (27), and (28) using the abbreviations.

$$A_1 \to \bigvee_{y_1} (B_1 \wedge C_1) \tag{31}$$

$$A_{r+1} \wedge B_r \wedge C_r \to \bigvee_{y_{r+1}} (B_{r+1} \wedge C_{r+1}) \tag{32}$$

$$A_r \wedge B_{r+1} \wedge C_r \to \bigvee_{x_{r+1}} (A_{r+1} \wedge C_{r+1}) \tag{33}$$

Recall that r is even in (32) and odd in (33). These formulas have Π^s_3 proofs, so we can use them, because we are constructing a Π^s_{k+1} proof where $k \geq 2$.

The following formulas are the first steps of the derivation of Δ_i with Δ_{i+1} being the first formula (a).

(a)	$\bigwedge_{y_1}(B_1 \to$	$\bigvee_{y_2}(B_2 \wedge$	$\bigwedge_{y_3}(B_3 \to$	$\bigvee_{y_4}(B_4 \wedge \ \ldots$
(b)	$\bigwedge_{x_1}\bigwedge_{y_1}((B_1 \wedge C_1) \to$	$\bigvee_{y_2}(B_2 \wedge C_1 \wedge$	$\bigwedge_{y_3}(B_3 \to$	$\bigvee_{y_4}(B_4 \wedge \ \ldots$
(c)	$\bigwedge_{x_1}(A_1 \to$	$\bigvee_{y_1}\bigvee_{y_2}(B_2 \wedge C_1 \wedge$	$\bigwedge_{y_3}(B_3 \to$	$\bigvee_{y_4}(B_4 \wedge \ \ldots$
(d)	$\bigwedge_{x_1}(A_1 \to$	$\bigvee_{y_1}\bigvee_{y_2}(A_1 \wedge B_2 \wedge C_1 \wedge$	$\bigwedge_{y_3}(B_3 \to$	$\bigvee_{y_4}(B_4 \wedge \ \ldots$
(e)	$\bigwedge_{x_1}(A_1 \to$	$\bigvee_{y_1}\bigvee_{y_2}\bigvee_{x_2}(A_2 \wedge B_2 \wedge C_2 \wedge$	$\bigwedge_{y_3}(B_3 \to$	$\bigvee_{y_4}(B_4 \wedge \ \ldots$
(f)	$\bigwedge_{x_1}(A_1 \to$	$\bigvee_{y_1}\bigvee_{y_2}\bigvee_{x_2}(A_2 \wedge B_2 \wedge C_2 \wedge$	$\bigwedge_{y_3}((B_3 \wedge C_3) \to$	$\bigvee_{y_4}(B_4 \wedge C_3 \ \ldots$
(g)	$\bigwedge_{x_1}(A_1 \to$	$\bigvee_{y_1}\bigvee_{y_2}\bigvee_{x_2}(A_2 \wedge B_2 \wedge C_2 \wedge$	$\bigwedge_{y_3}\bigwedge_{x_3}((A_3 \wedge B_2 \wedge C_2) \to$	$\bigvee_{y_4}(B_4 \wedge C_3 \ \ldots$
(h)	$\bigwedge_{x_1}(A_1 \to$	$\bigvee_{y_1}\bigvee_{y_2}\bigvee_{x_2}(A_2 \wedge$	$\bigwedge_{x_3}(A_3 \to$	$\bigvee_{y_3}\bigvee_{y_4}(B_4 \wedge C_3 \ \ldots$
	$\ldots$	$\ldots$	$\ldots$	$\ldots$

We will describe how these formulas follow from previous ones.

Proof of $(a) \vdash (b)$. Add C_1 using (3).

Proof of $(b) \vdash (c)$. Using cuts with (31).

Proof of $(c) \vdash (d)$. Add A_1 using (3) and weakening.

Proof of $(d) \vdash (e)$. Since want to keep B_2, we first clone it and then apply cuts with (33).

Proof of $(e) \vdash (f)$. Add C_3 in the same way as in (b).

Proof of $(f) \vdash (g)$. Cut $B_3 \wedge C_3$ with (32) for $r = 2$.

Proof of $(g) \vdash (h)$. Cut $B_2 \wedge C_2$.

At the end we get

$$\begin{array}{llll} \dots & \dots & \dots & \bigwedge_{x_{k-1}}(A_{k-1} \to \bigvee_{y_{k-1}} \bigvee_{y_k}(B_k \wedge C_{k-1}))\dots) \\ \dots & \dots & \dots & \bigwedge_{x_{k-1}}(A_{k-1} \to \bigvee_{y_{k-1}} \bigvee_{y_k}(A_{k-1} \wedge B_k \wedge C_{k-1}))\dots) \\ \dots & \dots & \dots & \bigwedge_{x_{k-1}}(A_{k-1} \to \bigvee_{y_{k-1}} \bigvee_{y_k}(A_k \wedge C_k))\dots) \end{array}$$

We do not need C_k anymore, so we remove it by weakening. What we get is Δ_k with additional disjunctions $\bigvee_{y_1}, \dots, \bigvee_{y_k}$. Since the formulas do not depend on $y_1, \dots, y_k$ anymore, the elements of these disjunctions are identical, hence we can get rid of the disjunctions by contractions.

3. We now prove (30) from Δ_1 and the clauses

$$S_{1,1}(\Lambda), S_{2,1}(\Lambda x_2) \to S_{3,1}(\Lambda x_2 x_2), S_{4,1}(\Lambda x_2 x_2 x_4) \to S_{5,1}(\Lambda x_2 x_2 x_4 x_4), \dots \ . \tag{34}$$

Here are the first steps of the proof.

$$\begin{array}{llll} \bigwedge_{x_1}(S_{1,1}(x_1) \to & \bigvee_{x_2}(S_{2,1}(x_1 x_2) \wedge & \bigwedge_{x_3}(S_{3,1}(x_1 x_2 x_3) \to & \bigvee_{x_4} S_{4,1}(x_1 x_2 x_3 x_4))) \wedge \dots \\ & \bigvee_{x_2}(S_{2,1}(\Lambda x_2) \wedge & \bigwedge_{x_3}(S_{3,1}(\Lambda x_2 x_3) \to & \bigvee_{x_4} S_{4,1}(\Lambda x_2 x_3 x_4))) \wedge \dots \\ & \bigvee_{x_2}(S_{3,1}(\Lambda x_2 x_2) \wedge & \bigwedge_{x_3}(S_{3,1}(\Lambda x_2 x_3) \to & \bigvee_{x_4} S_{4,1}(\Lambda x_2 x_3 x_4))) \wedge \dots \\ & & & \bigvee_{x_2} \bigvee_{x_4} S_{4,1}(\Lambda x_2 x_2 x_4))) \wedge \dots \end{array}$$

Thus we get, in fact, a stronger formula

$$\bigvee_{x_2} \bigvee_{x_4} \cdots \bigvee_{x_{k-2}} \bigvee_{x_k} S_{k,1}(\Lambda x_2 x_2 x_4 x_4 \dots x_{k-2} x_{k-2} x_k).$$

from which we get contradiction using cuts with formulas (20).

Case $k \geq 3$ odd. For $i = 1, \dots, n$, we will denote by ∇_i the following formula

$$\bigwedge_{x_1}(S_{1,i}(x_1) \to \bigvee_{x_2}(S_{2,i}(x_1 x_2) \wedge \bigwedge_{x_3}(S_{3,i}(x_1 x_2 x_3) \to \cdots \bigvee_{x_{k-1}} (S_{k-1,i}(x_1 \dots x_{k-1}) \wedge \bigwedge_{x_k} \neg S_{k,i}(x_1 \dots x_k))))). \tag{35}$$

Note that ∇_i is Π^s_{k+1}. The proof is similar to the proof for the case of k even with a few modifications.

First, to derive ∇_n we also need to use clauses (19), which we now denote by $\neg S_{k,n}(x_1 \dots x_k)$.

We will now describe how to derive ∇_i from ∇_{i+1}. Using our abbreviations and writing $\neg B_k$ as $B_k \to \bot$, formula ∇_{i+1} becomes

$$\bigwedge_{y_1}(B_1 \to \bigvee_{y_2}(B_2 \wedge \bigwedge_{y_3}(B_3 \to \cdots \bigvee_{y_{k-1}} (B_{k-1} \wedge \bigwedge_{y_k}(B_k \to \bot)\dots)))).$$

This is like Δ_{i+1} for depth $k+1$ (which is even) the only difference being that B_{k+1} is $\bot$. The same holds true for ∇_i, so we can proceed in the same way as in the case of $k+1$. Having $\bot$ instead of B_{k+1} makes our task even easier.

Finally, we derive contradiction by resolving ∇_1 with clauses (34). Here is how it goes for $k = 3$.

$$\begin{aligned}
\textstyle\bigwedge_{x_1}(S_{1,1}(x_1) \rightarrow \ & \textstyle\bigvee_{x_2}(S_{2,1}(x_1x_2) \quad \wedge \quad \bigwedge_{x_3} \neg S_{3,1}(x_1x_2x_3))) \\
& \textstyle\bigvee_{x_2}(S_{2,1}(\Lambda x_2) \quad \wedge \quad \bigwedge_{x_3} \neg S_{3,1}(\Lambda x_2x_3)) \\
& \textstyle\bigvee_{x_2}(S_{3,1}(\Lambda x_2x_2) \quad \wedge \quad \bigwedge_{x_3} \neg S_{3,1}(\Lambda x_2x_3)) \\
& \textstyle\bigvee_{x_2}(S_{3,1}(\Lambda x_2x_2) \quad \wedge \quad \neg S_{3,1}(\Lambda x_2x_2)) \\
& \bot
\end{aligned}$$

We leave the generalization for all odd $k \geq 3$ to the reader.

This finishes the proof of Lemma 5.2. ∎

6 Games from proofs

In this section we prove the opposite reduction, i.e., we will reduce the interpolation pair of the Π^s_k-Symmetric Calculus to the pair (A_k, B_k) of the depth $k-1$ games.

Lemma 6.1 *For every $k \geq 2$, given a refutation D of a CNF formula $\Phi(\bar{x}) \wedge \Psi(\bar{y})$ in the Π^s_k-Symmetric Calculus, where the sets of variables $\bar{x}$ and $\bar{y}$ are disjoint, one can construct in polynomial time a game G of depth $k-1$ such that if $\Phi(\bar{x})$ is satisfiable then Player I has a positional winning strategy, and if $\Psi(\bar{y})$ is satisfiable then Player II has a positional winning strategy. Moreover, the positional winning strategies can be constructed in polynomial time from the satisfying assignments.*

Let a refutation D of a CNF formula $\Phi(\bar{x}) \wedge \Psi(\bar{y})$ in the Π^s_k-Symmetric Calculus be given. We will assume that the refuted CNF is represented by a Π^s_k formula as follows. If k is even, then the bottom connectives of Π^s_k are disjunctions. So in this case, we will simply be pad the CNF on the top, which is schematically represented by

$$\wedge \vee \cdots \wedge (\cdots \vee p \vee q \vee \dots).$$

If k is odd, the bottom connectives are conjunctions. So we will first pad literals to conjunctions and then we pad it on the top, which is schematically represented by

$$\wedge \vee \cdots \wedge (\cdots \vee (\wedge(p)) \vee (\wedge(q)) \vee \dots).$$

Let a Π^s_k refutation $D := (\Phi(\bar{x}) \wedge \Psi(\bar{y}) = \Gamma_1, \dots, \Gamma_m = \bot)$ be given. We will first define a game with k rounds and m steps in each round and then show that the last round can be omitted so that we obtain a game of depth $k-1$.

The game starts at the last column that is associated with the last formula of the proof, which is $\bot$ padded to the level Π^s_k. The game starts with $\bot$ padded to level Σ^s_{k-1} and then players proceed by Player's selecting maximal Σ^s_{k-1} subformulas of $\Gamma_{m-1}, \Gamma_{m-2}, \ldots$, so they select disjunctions from conjunctions. At some point they go to next row (if there is any) and change direction. When going to the right they select Π^s_{k-2} conjunctions from the selected Σ^s_{k-1} disjunctions. Then at some point they go to the next row (if there is any) and change direction again, and so on until the bottom row.

Players cannot select arbitrary subformulas, but only those that are in a certain sense "logically connected". Before we define the rules for selecting subformulas, we state the rules for changing directions.

Changing the direction. Players go to the next row and change the direction

1. when they hit padded $\bot$ going to the right,
2. when they hit padded $\top$ going to the left,
3. when they get to the padded refuted CNF while going left.

Note that when they hit $\bot$ going to the right, next time when going to the right they cannot get beyond it, because they hit it again, or hit another one before that. The same holds true for going to the left and hitting $\top$. In particular, if they hit $\top$ then they will not get to the initial CNF anymore.

There is a number of properties of the game we are defining that are not in accord with the formal definition given in Section 4, one of which is the possibility of going to the next row before the play reaches an end of the row. This was not allowed by the definition of the games in Subsection 4.1, but we have mentioned that it is possible to simulate such more general games. We also mentioned the possibility that the players do not alternate regularly. Another small and inessential discrepancy is that we defined games so that in the first and the last columns the symbols are not rewritten when starting in the opposite direction, which is not literally true in the case of the last column where one padding of $\bot$ is removed and the same concerns the first column where the refuted CNF is. The least important fact is that we start from the last column instead of the first one.

Legal moves. As stated above the formula played must be a stratified subformula of either the formula in the proof (in the first round) or the formula played in the previous round of particular depth. Viewing formulas as trees, it must be a node connected to the previously played node of the tree. If the played subformula is not involved in an application of a deep inference rule, then the next subformula played is uniquely determined—it is the same formula on the corresponding position. In such a case the play proceeds without any action of the players. We will now define legal moves when a rule is applied to the subformula, or the subformula is a result of such an application.

Suppose the play proceeds from the left to the right. Then, for some i, the formulas played are Π_i^s subformulas of Σ_{i+1}^s formulas. Note that only the rules in the left column (see Section 3.3) change the structure of disjunctions, so we only need to consider them.

In the following formulas we assume that $\bot, p, \neg p$ are padded to the appropriate level.

1. Permutation of a disjunction or conjunction. Players do not make any decisions; the play proceeds to the corresponding term of the disjunction or conjunction.

2. Contraction, $\dfrac{\ldots B \vee A \vee A \vee C \vee \ldots}{\ldots B \vee A \vee C \vee \ldots}$.
   When any of the two occurrences of A was played in the previous move, the next is the occurrence of A in the conclusion of the rule. So also in this case players do not act.

3. $\bot$ elimination, $\dfrac{\ldots B \vee \bot \vee C \vee \ldots}{\ldots B \vee C \vee \ldots}$.
   If $\bot$ was played, the play cannot continue in the direction to the right. So the direction reverses and starts with $\bot$ with one padding removed.

4. Weakening, $\dfrac{\ldots B \vee C \vee \ldots}{\ldots B \vee A \vee C \vee \ldots}$.
   All formulas from the premise are present in the conclusion, so the same formula is played as in the previous move.

5. Dual resolution, $\dfrac{\cdots \vee C \vee (A \wedge \top \wedge B) \vee D \vee \ldots}{\cdots \vee C \vee (A \wedge p) \vee (B \wedge \neg p) \vee D \vee \ldots}$.
   If $A \wedge \top \wedge B$ was the previous played subformula, then the legal moves are either $A \wedge p$ or $B \wedge \neg p$. Which of the two is played is decided by the player who owns the literal. When going back, if p is chosen from $A \wedge p$, then $\top$ must be played in the next move (and the direction must be reversed, or the game ends).

The following figure illustrates the last case when in the next round Player I chose p from $A \wedge p$. For the sake of readability, we have omitted padding. (For the game to continue after it reached $\top$, this occurrence of $\top$, as well as p, must be padded.)

$\leftarrow$	$\ldots$	$\cdots \vee C \vee (A \wedge \top \wedge B) \vee D \vee \ldots$	$\cdots \vee C \vee (A \wedge p) \vee (B \wedge \neg p) \vee D \vee \ldots$	$\ldots$
$\rightarrow$	$\ldots$	$A \wedge \top \wedge B$	$A \wedge p$	$\ldots$
$\leftarrow$	$\ldots$	$\top$	p	$\ldots$
		$\mapsto \quad \top$	p	$\ldots$

Now suppose the play goes from the right to the left. By symmetry, this is the same, except that now the game may reach the initial formula. Since in this direction the subformulas played are Σ_i^s for some i, the play arrives either at some unary $\vee$ by which the initial formula is padded, or to a clause. If it is the padded formula, one padding is removed. If a

clause is reached, then the player who owns it chooses a literal and they reverse the direction of play, i.e., if the clause is from variables $\bar{x}$ Player I chooses a literal, if it is from variables $\bar{y}$, Player II chooses a literal.

Termination of the game; winning positions. The game ends when players hit $\bot$, $\top$, or the first column when passing the bottom row. On the bottom row they play a literal. The player whose literal hits $\bot$, $\top$, or the first column looses the game.

Example.

...	$\leftarrow$	...	...	...	...	$\vee \wedge \vee(\bot)$
...	$\rightarrow$	...	$A \wedge (r \vee p) \wedge (\neg p \vee s) \wedge B$	$A \wedge (r \vee \bot \vee s) \wedge B$	...	$\wedge \vee (\bot)$
...	$\leftarrow$	...	$r \vee p$	$r \vee \bot \vee s$	...	$\vee(\bot)$
...	$\rightarrow$	...	p	$\bot$	game ends	

In the last step of this play we interpret $\bot$ as logically connected with p, therefore $\bot$ follows after p. If this was not in the bottom row (in which case p and $\bot$ would be padded), then the direction would be reversed and the play would go on.

The abridged game. It is clear that the bottom row is superfluous: once they get to this row, the same literal is played until the end of the game. Hence, we can omit this row and define the terminating positions to be the positions where they are supposed to go to the bottom row, and declare the position to be losing for the player who owns the literal to be played. In this way, from a Π^s_k proof, we obtain a game of depth $k-1$.

When k is even, literals of the initial formula are padded to conjunctions. Thus a literal from a disjunction may be chosen when going to the $k-1$st round, but the game is not decided yet. The literal that eventually hits $\bot$ may be different.

6.1 A winning strategy from a satisfying assignment

To prove Lemma 6.1, it suffices now to prove the following.

Lemma 6.2 *Given a satisfying assignment for $\Phi(\bar{x})$, one can construct in polynomial time a positional winning strategy for Player I. The same holds for a satisfying assignment to $\Psi(\bar{y})$ and Player II.*

Proof. For this proof, it will be convenient to consider the full game, not the abridged version.

Let $\bar{a}$ be a satisfying assignment for $\Phi(\bar{x})$. Recall that the only decisions that Player I has to make occur when the rules of resolution and dual resolution are applied and when the players arrive at a clause $C(\bar{x})$ of the refuted CNF. The strategy is defined as follows:

1. The play proceeds to the left and arrives at a resolution step with a variable $x \in \bar{x}$. Then Player I chooses the disjunction in which the literal is *falsified* by $\bar{a}$.

2. The play proceeds to the right and arrives at a dual resolution step with a variable $x \in X$. Then Player I chooses the conjunction in which the literal is *satisfied* by $\bar{a}$.

3. The play proceeds to the left and arrives at a clause $C(\bar{x})$ of the CNF. Then Player I chooses a literal from $C(\bar{x})$ that is *satisfied* by $\bar{a}$ (the literal may be padded to a conjunction if the bottom layer of connectives are conjunctions).

This is, clearly, a positional strategy. We will show that it is a winning strategy. We will consider two cases.

1. *The lowest level of connectives are disjunctions.* Then the players pass the bottom row, the level of literals, in the direction to the right. So the play stops when they hit $\bot$. We should show that the literal with which they hit $\bot$ cannot be from $\bar{x} \cup \neg\bar{x}$. Suppose, by way of contradiction, that this literal is $x \in \bar{x} \cup \neg\bar{x}$.

If they started the bottom row at its beginning, which is the refuted CNF, then it means that Player I has chosen a literal from some clause $C(\bar{x})$. Since he uses the strategy described above, the literal is satisfied by $\bar{a}$. On the other hand, in the resolution step that produced this occurrence of $\bot$, he chose the direction with the falsified literal. Since what he plays on a lower row is a subformula of a formula on a higher row, this is the same literal. (More precisely, on the higher row it is this literal padded to the appropriate level.) So this is not possible.

> *Example.* Suppose in the previous example Player I used this strategy and $p \in \bar{x}$. Then he chose $r \vee p$ because p was falsified by $\bar{a}$. When he started the bottom row, he should have picked a literal that is satisfied by $\bar{a}$, which is not p. So this situation cannot occur.

If they started at some occurrence of $\top$, then x must be one of the literals to which $\top$ split. Again, Player I chose the literal that is satisfied by $\bar{a}$, but x should be falsified by $\bar{a}$ because of the resolution step that produced x.

2. *The lowest level of connectives are conjunctions.* Then the players pass the bottom in the direction to the left and the play stops when they hit either $\top$, or the beginning of the row. Suppose that the literal with which they hit $\top$ or the beginning is $x \in \bar{x} \cup \neg\bar{x}$. Since they always start the bottom row from some occurrence of $\bot$ and go left, x must be falsified by $\bar{a}$. There are two cases:

(i) If they hit $\top$, then we get a contradiction as above.

(ii) Suppose they hit the beginning. This means that in the previous round they started from the beginning of the row. Then one player had to choose a literal (padded to a conjunction) from a clause of the CNF. Since $x \in \bar{x} \cup \neg\bar{x}$, it was chosen by Player I. But then, according to the rules of the strategy, it must be satisfied by $\bar{a}$. This is a contradiction again.

■

Example. Consider the unabridged version of the game. Let k be odd. So the bottom connective is conjunction and the literals of the initial formula are padded to conjunctions. Suppose Player I plays the strategy based on a satisfying assignment for $\Phi(\bar{x})$. Suppose they start the last but one round by Player I choosing a literal x from a clause of $\Phi(\bar{x})$; he chooses a satisfied literal. Now they proceed to the right. The single element conjunction $\wedge(x)$ may be enlarged as they go on. The rule that enables it is dual resolution. Literals from both X and Y can be added, but Player I only picks those that are satisfied. Eventually all but one, say p, are removed by weakening and they hit $\bot$. The literal p cannot be from $\bar{x} \cup \neg\bar{x}$, because if it were, then on some previous row Player I would decide where to go from $\bot$ and he would choose the unsatisfied literal from the two options. So p belongs to Player II. Then Player II loses, because p is repeated on the bottom level all the way to the end of the game. The game will end at the position where p was introduced by dual resolution. There the game will hit $\top$.

7 A stronger result

Game schemas. We will call a *game schema* a system of rules S that defines legal moves and a set of end positions. What a schema does not specify is which positions are winning for which player. We will furthermore require the end positions to be labeled by 0s, 1s, and variables $z_1, \ldots, z_n$. Given a schema $S(\bar{z})$ and an assignment $\bar{a} : \{z_1, \ldots, z_n\} \to \{0, 1\}$, we obtain a game $S(\bar{a})$ where winning positions of Player I are the positions labeled by 1 and those labeled by 0 are winning for Player II. For this article, the precise definition of game schemas is not important because we will only need particular instances, which will be defined explicitly.

A game schema can be used to define a total monotone Boolean function $f : \{0, 1\}^n \to \{0, 1\}$, if we consider general strategies, and a partial monotone function, if we consider only positional strategies. The value of the function is 1 (respectively 0), if Player I (Player II) has a winning strategy. We will mostly be interested in positional strategies.

In the case of games that we have introduced it is very easy to define the corresponding concept of a game schema—it suffices to omit the set of winning symbols W from Definition 1 and suitably label the elements of the set A. We will call resulting objects *depth k game schemas.*

Our aim is to prove the following strengthening of polynomial simulation of interpolation pairs, which generalizes monotone feasible interpolation for Resolution.

Theorem 7.1 *Let $\Phi(\bar{x}, \bar{z})$ and $\Psi(\bar{y}, \bar{z})$ be two CNF formulas whose only common variables are $\bar{z}$. Suppose variables $\bar{z}$ occur in Φ only positively and in Ψ only negatively. Let a Π^s_k refutation D of $\Phi(\bar{x}, \bar{z}) \wedge \Psi(\bar{y}, \bar{z})$ be given, $k \geq 2$. Then it is possible to construct in polynomial time a depth $k - 1$ game schema $S(\bar{z})$ such that for every assignment $\bar{a} : \bar{z} \to \{0, 1\}$, if $\Phi(\bar{x}, \bar{a})$ is satisfiable, then Player I has a positional wining strategy in $S(\bar{a})$ and if $\Psi(\bar{y}, \bar{a})$ is satisfiable, then Player II has a positional wining strategy in $S(\bar{a})$.*

Note that in particular the size of the game schema S is polynomial in the size of the proof D.

Proof. The proof is a simple adaptation of the proof of Lemma 6.2. Let a Π^s_k refutation D be given. We introduce new variables $z'_1, \ldots, z'_n$ and substitute them for $\neg z_1, \ldots, \neg z_n$ in $\Psi(\bar{y}, \bar{z})$. Let $\Psi'(\bar{y}, \bar{z}')$ be the formula after the substitution. We take the CNF formula $\Phi(\bar{x}, \bar{z}) \wedge \Psi'(\bar{y}, \bar{z}') \wedge \Delta(\bar{z}, \bar{z}')$, where $\Delta(\bar{z}, \bar{z}')$ is the conjunction of all clauses $\neg z_i \vee \neg z'_i$. One can, clearly, construct a Π^s_k refutation D' of $\Phi(\bar{x}, \bar{z}) \wedge \Psi'(\bar{y}, \bar{z}') \wedge \Delta(\bar{z}, \bar{z}')$ that is only slightly larger than D.

We define a game schema from D' in the same way as we did in Lemma 6.2 with one modification that concerns clauses $\neg z_i \vee \neg z'_i$. When the play arrives at such a clause, we let Player I choose a literal from it. If he chooses $\neg z_i$ then the play continues in the usual manner. If he chooses $\neg z'_i$, then Player II will have an opportunity to challenge Player I's move. If she challenges, then the game ends and the end position gets label z_i. If she does not challenge, the game continues as before.

Let an assignment $\bar{a} : \bar{z} \to \{0, 1\}$ be given and suppose $\Phi(\bar{x}, \bar{a})$ is satisfiable. Let $\bar{b} : \bar{x} \to \{0, 1\}$ be the satisfying assignment. We will use $\bar{a}$ also for variables $\bar{z}'$ as if they were $\neg\bar{z}$. Player I will use $\bar{b}, \bar{a}$ for his strategy in the same way as it was in Lemma 6.2. He controls the choice from clauses $\neg z_i \vee \neg z'_i$ and will always pick the satisfied literal. Player II can challenge only if he picks $\neg z'_i$. But then $\neg z'_i$ is satisfied so z'_i not satisfied and z_i is satisfied, i.e., $a_i = 1$. Thus this action of Player II would result in Player I immediately winning.

Suppose now that $\Psi(\bar{y}, \bar{a})$ is satisfiable. This means that $\Psi'(\bar{y}, \neg\bar{a})$ is satisfiable. Now Player II does not control the action when they arrive at $\neg z_i \vee \neg z'_i$. But what she only needs for her strategy is that $\neg z'_i$ is not chosen if $\neg z'_i$ is not satisfied. If Player I chooses $\neg z'_i$ in spite of $\neg z'_i$ being not satisfied, then z'_i is satisfied and z_i is not satisfied, i.e., $a_i = 0$. Hence Player II challenges and wins immediately. ∎

Let $\mathcal{S}$ be a family of game schemas with a concept of a positional winning strategy. Then for $S(\bar{z}) \in \mathcal{S}$ we will denote by $f_S(\bar{z})$ the partial Boolean function defined by S. Let $\mathcal{S}$ and $\mathcal{T}$ be two families of game schemas. Then we will say that $\mathcal{S}$ *strongly polynomially* (respectively *quasipolynomially*) reduces to $\mathcal{T}$, if for every game schema $S(\bar{z}) \in \mathcal{S}$, there exists at most polynomially (quasipolynomially) larger game schema $T(\bar{z}) \in \mathcal{T}$ such that $f_S \subseteq f_T$ (i.e., whenever $f_S(\bar{z})$ is defined, so is $f_T(\bar{z})$ and $f_S(\bar{z}) = f_T(\bar{z})$). The word "strongly" refers to the fact that in the reductions the sets of variables are exactly the same.

Corollary 7.2 *Let $k \geq 2$. Suppose Π^s_k-Symmetric Calculus can quasipolynomially simulate Π^s_{k+1}-Symmetric Calculus on CNFs. Then game schemas of depth k strongly quasipolynomially reduce to game schemas of depth $k - 1$.*

We have stated the corollary for quasipolynomial simulation, because we know that the depth $k - 2$ sequent calculus does not polynomially simulate the depth $k - 1$ sequent calculus (cf. [11] and [13], Theorem 14.5.1), hence also Π_k-Symmetric Calculus does not polynomially simulate Π_{k+1}-Symmetric Calculus.

8 Two special cases

We will consider two special cases: depth 1 and depth 2 games. We will show that depth 1 game schemas are essentially monotone Boolean circuits and depth 2 games are equivalent to point-line games introduced in [4].

8.1 Depth 1 games

Let C be a monotone Boolean circuit. C is given by a directed oriented graph H with a root r. The root is the output of the circuit. Vertices are labeled by $\vee$ and $\wedge$, except for leaves which are labeled by variables $x_1, \ldots, x_n$. Let $\bar{a}$ be an assignment to the variables. Then we can view the pair $C, \bar{a}$ as a game with two players $\bigvee$ and $\bigwedge$. They start at the root and follow the arrows with the direction chosen by the player by whose label the vertex is labeled. Player $\bigvee$ wins iff they reach a leaf whose variable is substituted by 1. It is not difficult to see that Player $\bigvee$ has a winning strategy iff $C(\bar{a}) = 1$.

Thus in our terminology a monotone Boolean circuit is a depth 1 game schema. It is a universal model in the following sense. If the number of configurations in a finite game schema S is N, then it can be represented by a monotone Boolean circuit with N vertices. In particular, monotone Boolean circuit can represent our depth 1 game schemas with the number of vertices polynomial in the size of the game schema.

The converse simulation, the simulation of depth 1 games by monotone Boolean circuits, is also easy. Given a depth 1 game, the nodes of the circuit will be positions in the game and two nodes are connected if there is a legal move from one position to the other. We leave the details to the reader.

We summurize these observation in the following proposition.

Proposition 8.1

1. *In every depth 1 game one of the players has a positional winning strategy.*
2. *One can decide in polynomial time who has a winning strategy in a depth 1 game.*
3. *Monotone Boolean circuits and depth 1 game schemas strongly reduce to each other.*

8.2 Depth 2 games

The canonical NP pair of Resolution, which is polynomially equivalent to the interpolation pair of the depth 1 sequent calculus, has been characterized by a game called the *point-line game* [4]. It follows that the pairs of the point-line game and depth 2 game are polynomially equivalent. Here we will show direct simulations which also preserve monotonicity.

A point-line game is given by a directed acyclic graph H with a root R and some additional structure. We will view the nodes of the graph as having some inner structure—like circles in which points are drawn. Each node is assigned either to player Black or player White. The root is empty, the other nodes contain some points and each leaf contains exactly one point. If there is an arrow from a node P to a node Q, then there is a partial matching $M_{P,Q}$ between the points of P and Q. A play starts at the root and proceeds along

the arrows to a leaf. At each node the player who owns it decides where to proceed. When a node is visited, it is filled with black and white pebbles put on the points for the node. The configurations of pebbles are determined by the way in which the node was reached. The rule for pebbles is:

- if the play goes $P \to Q$, then pebbles that are in the domain of $M_{P,Q}$ are moved from P along the lines to Q; the remaining points of Q are filled with pebbles of the player other than the one who did this move.

When the play reaches a leaf, then the color of the pebble that ends up there decides who wins.

There are certainly many modifications that result in essentially the same concept. For us, the most important one is to allow more points in the root. This version is then a game schema where an instance is given by putting some pebbles on the points of the root. Then we can use this schema to compute partial monotone functions.

Once we allow points in the root, we can also w.l.o.g. assume that the range of each matching $M_{P,Q}$ covers all points in Q. Then the rule about pebbles becomes simpler—just move pebble along the matchings.

Since both the depth 2 games and point-line games characterize the interpolation pair of the depth 1 sequent calculus, which is equivalent to Π_2-Symmetric Calculus, the NP-pairs of the two kinds of games are polynomially reducible to each other. Below we will show direct simulations that, moreover, show strong polynomial reduction between the corresponding game schemas.

Simulation of point-line games by depth 2 games. In order to see the connection with depth 2 games, consider a point-line game, with the modifications mentioned above, presented in a different way. The play starts with some configuration of pebbles on the root. Then the players traverse the graph, but instead of putting pebbles on the nodes they only mark the path they have taken. When they reach a leaf, they start on the point in that leaf and go back along the lines that connect points of the taken path. Thus they get back to a point in the root. The color of the pebble that is there decides who wins.

It is clear that the game is the same. Moving the pebbles in the original way of playing the game is only a means to save the trip back to the root.

In this formulation it is clear that the point-line game *is* a depth 2 game in which players can pick moves from more than two alternatives, which is an inessential generalization. In fact, it is a special case of depth 2 game—in the last round players do not decide anything. This may suggest that the point-line game is a weaker concept, but this is not the case. We will show below that one can also simulate depth 2 games by point-line games.

Simulation of depth 2 games by point-line games. The idea of the simulation is to use the directed graph of the positions in the first round as nodes of the point-line game and positions in the second round as their points. A position in the second round in the

ith column is a pair (a, b) where a is a symbol played in the first round and b is a symbol played in the second round. Thus a will be a node and (a, b) a point in it. Black pebbles will represent winning positions of Player I and white pebbles the winning positions of Player II. This means that if $\Lambda, a_2, \ldots, a_i$ are the first i moves in the depth 2 game, then a black pebble on (a_i, b) means that Player I has a strategy to win the game if started from position (a_i, b) and $\Lambda, a_2, \ldots, a_i$ are fixed on the first row. Imagine that the game has been played until this point and it remains i steps to finish the game. Having such winning positions for (a_i, b) and $\Lambda, a_2, \ldots, a_i$ and given a_{i+1} that is a legal move after a_i, we can easily determine winning positions (a_{i+1}, c) for $\Lambda, a_2, \ldots, a_i, a_{i+1}$. The process of defining the winning position (a_{i+1}, c) can be represented as moving black pebbles from node a_i to node a_{i+1}, except that it is slightly more complicated than just moving pebbles along lines. Eventually we arrive at a_n. Then, by definition, there is only one position with a_n as the first coordinate, namely (a_n, a_n). If it is a winning position for $\Lambda, a_2, \ldots, a_n$, which is represented by a black pebble on it, then Player I has a strategy to win the game from this position with $\Lambda, a_2, \ldots, a_n$ in the first row.

We will now describe the simulation in more detail. Let a depth 2 game be given. We will first construct a point line game with slightly more complicated rules for moving pebbles. We will allow conjunctions and disjunctions with pebbles, which means that for two nodes connected by an arrow $P \to Q$ we may have two points $p_1, p_2 \in P$ connected by lines to a point $q \in Q$ and q labeled by $\vee$ or $\wedge$. The rule when the label is $\vee$ is that q gets a black pebble iff there is at least one black pebble on p_1 and p_2. If the label is $\wedge$ then q gets a white pebble iff there is at least one white pebble on p_1 and p_2. (So we interpret black pebbles as 1s and white as 0s.)

The nodes of the point-line game will be pairs (i, a), where i is a position on the tape and $a \in A$ is a symbol. They will be connected by an arrow when a transition $(i, a) \to (i + 1, b)$ is possible. Each node will be labeled by a player, where we rename Player I to Black and Player II to White. The points of a node (i, a) will be all triples (i, a, c), $c \in A$. The lines between points are defined as follows. If (a, c_1), (a, c_2) and (b, d) are positions in the depth-2 game and i is a place on the tape such that Player I is to decide to move from (b, d) to either (a, c_1) or (a, c_2), then $(i + 1, b, d)$ is labeled $\vee$. If for this triple, it is Player II who is to move, then it is labeled by $\wedge$. The initial node is $(1, \Lambda)$ and points are the end positions $(1, \Lambda, c)$, $c \in A$.[9] The initial position of pebbles consists of black pebbles placed on the winning positions of Player I, white pebbles are on winning positions of Player II.

It is clear that this point-line game simulates the depth 2 game in the sense of general winning strategies. What we must show is that a *positional* strategy in a depth 2 game can be translated into a *positional* strategy in the point-line game. But this is also easy. A positional strategy in a depth 2 game determines, in particular, what a player should do in a position in the first round of the game. These positions correspond to nodes in the point-line game and one can use the same actions.

It remains to show that we can simulate disjunctions and conjunctions by the standard rule of the point-line game. First we observe that we can assume, w.l.o.g., that for any

[9] We assume that the plays always end at the left-most position.

pair of nodes P and Q connected by an arrow $P \to Q$ there is at most one conjunction or disjunction and all other lines are as in the standard game, i.e., they only copy pebbles. This can be achieved by inserting k new nodes if there are $k+1$ conjunctions or disjunctions.

Suppose we have $(i,a) \to (i+1,b)$ and there are two points $(i,a,c_1),(i,a,c_2)$ in node (i,a) with lines going to one point $(i+1,b,c)$ in node $(i+1,b)$. We now suppose that there can only be one such point $(i+1,b,c)$. Suppose $(i+1,b,c)$ is labeled $\vee$. Then we insert three new nodes D, D_1, and D_2 and replace the arrow $(i,a) \to (i+1,b)$ with $(i,a) \to D$, $D \to D_1 \to (i+1,b)$ and $D \to D_2 \to (i+1,b)$. D is labeled Black; the nodes D_i are unlabeled, because there is only one arrow going out of each of them. D, D_1, and D_2 have the same points as (i,a), except that D_1 misses c_2 and D_2 misses c_1. The lines between (i,a) and D and between D and D_1 and D_2, and between D_1 and D_2 and $(i+1,b)$ connect the corresponding points except that the lines between the missing points are missing. As a result, Black is able to get a black pebble on $(i+1,b,d)$ iff there is at least one black pebble on (i,a,c_1), or (i,a,c_2). To simulate $\wedge$ we only need to label D by White.

Again, we have to show that the reduction reduces *positional* winning strategies to *positional* winning strategies. Let a strategy for Player I be given. The translation to the point-line game is straightforward except for the case of the new nodes introduced because of a point labeled $\vee$. Consider the situation in the previous paragraph. Then we need to define the positional strategy for Black when he is playing at node D. For this, we will use his positional strategy when he is playing at position $(i+1,b,c)$ in the depth 2 game: if this strategy is to go to (i,a,c_j), then in the point-line game Black's strategy will be to go from D to D_j.

Thus we have shown:

Proposition 8.2 *Depth 2 game schemas and point-line game schemas strongly polynomially reduce to each other.*

Corollary 8.3 *Point-line schemas interpolate* Π^s_3 *proofs (equivalently, depth 1 sequent calculus proofs) in the sense of Theorem7.1.*

General winning strategies. We will now present another possible way of viewing point-line game schemas (hence also depth 2 game schemas), but now we will not restrict ourselves to positional winning strategies. Since in every finite game one player has a winning strategy, the point-line game schema defines a *total* monotone Boolean function if we consider all winning strategies.

To motivate what follows, let us first recall how one can view monotone Boolean circuits (hence also depth 1 game schemas). Instead of the standard way where a monotone Boolean circuit is presented as a device computing with bits, one can view it as a way of defining monotone Boolean functions. At a leaf labeled by x_i of the underlying graph we compute the function x_i, at a node labeled by $\vee$ we compute the disjunction of the functions defined on the predecessors of the node and similarly on a node labeled $\wedge$. Then the circuit defines

the function computed at the root.[10]

Now suppose we are given a point-line game schema. We define functions computed on the nodes of the underlying directed acyclic graph in a similar way, but we will have different variables for every node. We introduce a variable for every point in a node and the function computed at the node will be a function of these variables. If L is a leaf with the unique point l, then f_L is the function of one variable l that is the value of this variable. Let $P \to Q$, $P \to S$ be arrows in the graph, let P belong to Black, and let f_Q and f_S be functions computed at nodes Q and S. Let $p_1, \ldots, p_k \in P$, $q_1, \ldots, q_m \in Q$, and $s_1, \ldots, s_n \in S$ be the points of the three nodes, which we will view as variables of the functions f_P, f_Q and f_S respectively. Furthermore, we will view the lines between $P \to Q$ and $P \to S$ as substitutions σ_{PQ} and σ_{PS}, where $\sigma_{PQ}(q_i) = p_j$ if there is a line from p_j to q_i, and $\sigma_{PQ}(q_i) = 0$ if there is no line from P to q_i, and similarly for σ_{PS}. Then we define

$$f_P(\bar{p}) := f_Q(\sigma_{PQ}(\bar{q})) \vee f_R(\sigma_{PS}(\bar{s})).$$

If P belongs to White, then the definition is dual ($\vee$ replaced by $\wedge$ and 1 replaced by 0). It is not difficult to see that the function at the root R computes who has a winning strategy. We state it as a proposition for further reference.

Proposition 8.4 *If we interpret black pebbles as 1s and white pebbles as 0s, then the function computed at the root is 1 if Black has a winning strategy and 0 if White has a winning strategy.*

Exponential lower bounds on the size of monotone Boolean circuits of explicitly defined monotone Boolean functions have been proved by the approximation method invented by Razborov [16]. In the contemporary presentation this method uses k-DNFs and k-CNFs, for a suitable k to approximate functions computed at the nodes of the circuit. One shows that (1) at each node only a very small error is introduced and (2) the given function cannot be approximated with a small error. The essence of the method is that the error set of approximating the function computed by the circuit is the union of errors introduced at the nodes, so if the circuit is small this set also has to be small. While the above generalization of monotone Boolean circuits is very similar to the standard monotone Boolean circuits, the approximation method fails in this case. The reason is that the error set introduced at a node is not connected with the set of variables of f_R. Various substitutions produce various versions of the error set and thus the total size of the copies can eventually be exponentially larger, even if the circuit has polynomial size.

Note, however, that this computational model may be much stronger than what we need, it may be even PSPACE complete. But we only need computations that tell us who has a *positional* winning strategy; such computation models may be more amenable to lower bounds.

[10] General Boolean circuits can, certainly, be treated in the same way, but in this article we focus on monotone functions.

8.3 Separation of depth 1 and depth 2 game schemas

The *clique-coloring tautology* $CC_{m,n,k}$, for $k < m < n$, states that there is no graph on n vertices that has a clique of size m and can be colored by k colors. As an unsatisfiable CNF formula it is formalized by using three sets M, N, K, $|M| = m$, $|N| = n$, $|K| = k$, mappings $f : M \to N$ and $g : N \to K$, and a graph G on N and saying the F is one-to-one, F maps M to a clique in G and g is a coloring of G.

The *clique-coloring function* $cc_{m,n,k}$, for $k < m < n$, is the *partial* monotone Boolean function defined on graphs on n vertices that is 1 if the graph has a clique of size m, is 0 if the graph is k-colorable, and undefined otherwise.

The clique-coloring tautology follows from the *pigeon-hole principle*, because if we compose f with g we get a one-to-one mapping from M to K. If $m = 2k$, then such a *weak pigeon-hole principle* is provable in depth 1 sequent calculus by proofs of size $n^{(\log n)^{O(1)}}$, which can be used to show that also the clique-coloring tautology $CC_{m,n,k}$ has proofs of asymptotically the same size; see [13], Section 18.7. This implies, by our Theorem 7.1, that the clique-coloring function $cc_{m,n,k}$ can be represented by depth 2 game schemas of size $n^{(\log n)^{O(1)}}$.

On the other hand, by classical lower bounds on monotone Boolean circuits [16, 1], any monotone Boolean circuit that computes the clique-coloring function $cc_{m,n,k}$ has exponential size 2^{n^ϵ}, $\epsilon > 0$, in particular for $m = 2k$ and $n = m^3$.

Corollary 8.5 *There exists a sequence of partial monotone Boolean functions that can be represented by polynomial size depth 2 schemas (equivalently, point-line schemas) but depth 1 schemas (equivalently, monotone circuits) require exponential size.*

Proof. Pad $cc_{m,n,k}$ where $m = 2k$ and $n = m^3$ with $n^{(\log n)^{O(1)}}$ dummy bits. ∎

9 Open problems

The next challenge is to characterize the canonical pair of unbounded Frege systems. This pair is polynomially equivalent to the interpolation pair.

Problem 1 *Characterize the canonical and interpolation pairs of Frege proof systems.*

It seems that our approach should work also in this case. A position in a game obtained from a Symmetric Calculus proof can be determined by a subformula of a formula in the proof. Hence the number of positions in the game is polynomially bounded. The definition of the game should accordingly be modified to allow only polynomial number of positions.

A characterization of canonical and interpolation pairs would also be interesting for other weak systems. In particular, the system with disjunctions of *parities* of literals, usually referred to as $Res(Lin)$, is currently intensively studied, but lower bounds have been obtained only for tree-like proofs.

Problem 2 *Characterize the canonical and interpolation pairs of Res(Lin).*

What we find the most desirable is to extend lower bound methods to stronger computational models. In this article we have presented game schemas as computation models for monotone Boolean functions. The weakest one for which we do not have lower bounds are depth 2 game schemas.

Problem 3 *Prove a superpolynomial lower bound on depth 2 game schemas representing (using positional strategies) an explicit monotone partial Boolean function.*

Such a lower bound would show that the method of monotone feasible interpolation could be pushed beyond the proof systems for which it is currently known to be applicable.

References

[1] N. Alon and R. Boppana, *The monotone circuit complexity of Boolean functions,* Combinatorica, 7(1) (1987), 1–22.

[2] A. Atserias and E. Maneva, *Mean-payoff games and propositional proofs*, Inform. and Comput., 209 (2011), 664–691.

[3] A. Atserias and M. Müller, *Automating Resolution is NP-hard.* Proc. 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS), to appear, 2019.

[4] A. Beckmann, P. Pudlák, N. Thapen, *Parity games and propositional proofs,* ACM Transaction on Computational Logic, Vol 15:2, article 17, 2014.

[5] M. L. Bonet, C. Domingo, R. Gavaldà, A. Maciel, and T. Pitassi, *Non-automatizability of bounded-depth Frege proofs*, Comput. Complexity, 13 (2004), 47–68.

[6] K. Brünnler, *Deep Inference and Symmetry in Classical Proofs.* Logos Verlag, Berlin, 2004.

[7] K. Brünnler and A. F. Tiu, *A local system for classical logic.* In R. Nieuwenhuis and A. Voronkov, eds., LPAR 2001, Lecture Notes in Artificial Inteligence, vol. 2250, (2001), 347–361.

[8] S. R. Buss, *An Introduction to Proof Theory,* in Handbook Proof Theory, S. R. Buss ed., Elsevier, Amsterdam, 1998, pp. 1–78.

[9] S. A. Cook, *Feasibly constructive proofs and the propositional calculus,* in Proc. 7th Annual ACM Symp. on Theory of Computing (STOC) (1975), 83–97.

[10] L. Huang and T. Pitassi, *Automatizability and simple stochastic games.* In Automata, languages and programming, Part I, vol. 7655 of Lecture Notes in Comput. Sci., Springer, Heidelberg, 2011, 605–607.

[11] R. Impagliazzo and J. Krajíček, *A note on the conservativity relations among bounded arithmetic theories,* Mathematical Logic Quarterly, 48(3) (2002), 375–377.

[12] L. Kołodziejczyk, P. Nguyen and N. Thapen, *The provably total NP search problems of weak second order bounded arithmetic,* Annals of Pure and Applied Logic, 162:6, (2011), 419–446.

[13] J. Krajíček, *Proof Complexity,* Cambridge University Press, 2019.

[14] J. Paris and A. Wilkie, *Counting problems in bounded arithmetic*, in Methods in mathematical logic (Caracas, 1983), vol. 1130 of Lecture Notes in Math., Springer, Berlin, 1985, 317–340.

[15] P. Pudlák, *On reducibility and symmetry of disjoint NP pairs*, Theoret. Comput. Sci., 295 (2003), 323–339.

[16] A. A. Razborov, *Lower bounds for the monotone complexity of some Boolean functions,* Doklady Akademii Nauk SSSR, 281(4) (1985), 798-801.

[17] A. A. Razborov, *On provably disjoint NP-pairs*, Tech. Rep. RS-94-36, Basic Research in Computer Science Center, Aarhus, Denmark, November 1994.

[18] A. Skelley and N. Thapen, *The provably total search problems of bounded arithmetic*, Proc. Lond. Math. Soc. (3), 103 (2011), 106–138.

Appendix

A1. First order theories and propositional proof systems

We briefly mention this subject, although we do not use the connection between first order theories and propositional proofs in this article. This section is also an apology why we are not using first order theories.

In fact, we are primarily interested in weak first order theories and study propositional proof complexity because it is a useful tool to prove independence from these theories. The connection first appeared in the seminal article of Stephen Cook [9]. A different form was studied by Paris and Wilkie in [14]. The latter one is more relevant to this work because it connects provability in bounded arithmetic and the size of proofs in bounded depth Frege systems. They extended bounded arithmetic by a new uninterpreted predicate P and added induction for bounded formulas in the extended language. They showed that if a Π^0_1 sentence is provable in such a theory, then a sequence of tautologies constructed from the sentence has proofs of polynomial sizes in a Frege system restricted to formulas of some constant depth. Using this relation one can show, e.g., that the pigeonhole principle stated with P is not provable in the extended bounded arithmetic.

One can get a closer relation between the theories and bounded depth Frege systems if one considers particular fragments. E.g., Buss's theory T^i_2 extended to $T^i_2[P]$ leads to

quasipolynomial Frege proofs in which formulas have depth i with the additional restriction that the *bottom fan-in is polylogarithmic*. In order to get a tight connection, Beckmann et al. [4] introduced special first order theory that capture precisely provability in the depth d sequent calculus for each $d \geq 0$. Thus one can prove polynomial upper bounds on the sizes of proofs of sequences of tautologies by arguing in a first order theory, which is often more convenient.

We could have used this connection to prove one part of our result, viz., Lemma 5.2, but we opted not to. In order to use a first order theory, we would have to describe the translation of first order formulas to propositional formulas and eventually the proof would not be much different. The difference would be essentially only in using quantifiers instead of big conjunctions and disjunctions. Furthermore, the theories of [4] are not so well established as $T_2^i[P]$ and we might need to sort out many of details.

Another reason for not using first order theories is to have this article selfcontained.

A2. A remark on simulating cuts

The simulation of cuts in Lemma 3.1 is a recursive procedure. It is important that it is run in a "depth-first" way. This means that after we split $A \vee (C_1 \wedge C_2)$ into $(A \vee C_1) \wedge (A \vee C_2)$, we first simulate cut with C_1 completely and only then we simulate cut with C_2. Here is an example.

Consider $(A \vee C) \wedge (B \vee \neg C)$ where $C = C_1 \wedge C_2$, $C_1 = p \vee q$ and $C_2 = r \vee s$. Then the proof will be:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(A \vee C) \wedge (B \vee \neg C)}{(A \vee C_1) \wedge (A \vee C_2) \wedge (B \vee \neg C_1 \vee \neg C_2)}\ \text{by distributivity}}{(A \vee p \vee q) \wedge (A \vee C_2) \wedge (B \vee \neg p \vee \neg C_2) \wedge (B \vee \neg q \vee \neg C_2)}\ \text{by distributivity}}{(A \vee C_2) \wedge (A \vee B \vee \neg C_2)}\ \text{by resolution with } p \text{ and } q}{(A \vee r \vee s) \wedge (A \vee B \vee \neg r) \wedge (A \vee B \vee \neg s)}\ \text{by distributivity}}{A \vee B}\ \text{by resolution with } r \text{ and } s$$

If we distributed C_2 immediately after distributing C_1, we would get

$$\cdots \wedge (B \vee \neg p \vee \neg r) \wedge (B \vee \neg p \vee \neg s) \wedge (B \vee \neg q \vee \neg r) \wedge (B \vee \neg q \vee \neg s)$$

This is like rewriting the DNF $(\neg p \wedge \neg q) \vee (\neg r \wedge \neg s)$ into the CNF $(\neg p \vee \neg r) \wedge (\neg p \vee \neg s) \wedge (\neg q \vee \neg r) \wedge (\neg q \vee \neg s)$. In general, this operation leads to an exponential blowup.