Lower Bounds for Heuristic Algorithms

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der
RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Christoph Berkholz
aus Nauen

Berichter: Universitätsprofessor Dr. Martin Grohe
Universitätsprofessor Dr. Martin Otto

Tag der mündlichen Prüfung: 4. Dezember 2014

für Ruth

Zusammenfassung

Für das Constraint-Satisfaction-Problem (CSP), das Erfüllbarkeitsproblem der Aussagenlogik und das Graphisomorphieproblem sind keine effizienten Algorithmen bekannt. Um sie zu lösen werden deshalb heuristische Algorithmen mit polynomieller Laufzeit eingesetzt. In der vorliegenden Dissertation werden drei klassische Heuristiken für die genannten Entscheidungsprobleme dahingehend untersucht, ob sie mit schnelleren als den bekannten Algorithmen implementiert werden können.

Die k-Konsistenz-Heuristik für das CSP versucht durch sukzessives Ableiten neuer Bedingungen lokal konsistente Instanzen zu erzeugen und kann mit einer Laufzeit von $O(n^{2k})$ implementiert werden. Wir zeigen, dass der Anstieg der Laufzeit mit wachsendem Parameter k unvermeidbar ist. Dazu beweisen wir für eine absolute Konstante $c > 0$, dass es nicht möglich ist in Zeit $O(n^{ck})$ zu entscheiden, ob lokale Konsistenz auf einer binären CSP-Instanz erreicht werden kann. Weiterhin untersuchen wir den Ableitungsprozess, der k-Konsistenz-Algorithmen zugrunde liegt, und beweisen eine optimale untere Schranke an die Anzahl der sequentiell abhängigen Ableitungsschritte.

Eine Heuristik für das aussagenlogische Erfüllbarkeitsproblem ist das Suchen nach einer Resolutionswiderlegung, in der jede Klausel höchstens k Literale enthält. Eine solche Resolutionswiderlegung der Weite k kann von einem einfachen Suchalgorithmus in Zeit $O(n^{k+1})$ gefunden werden. Wir zeigen für eine absolute Konstante $c > 0$, dass es unmöglich ist in Zeit $O(n^{ck})$ zu entscheiden, ob eine Resolutionswiderlegung der Weite k existiert. Die beiden unteren Schranken an die Laufzeit für lokale Konsistenz und Resolution beschränkter Weite kommen ohne komplexitätstheoretische Annahmen aus und verdeutlichen eine der wenigen Anwendung des Zeithierarchiesatzes auf natürliche Entscheidungsprobleme.

Der naive Klassifizierungsalgorithmus für das Graphisomorphieproblem berechnet eine stabile Färbung der Knotenmenge eines Graphen durch iteratives Verfeinern der Farbklassen. Durch geschickte Auswahl der zu verfeinernden Farbklassen kann das Verfahren auf zusammenhängenden Graphen mit n Knoten und m Kanten mit einer Laufzeit von $O(m \log n)$ implementiert werden. Wir zeigen, dass dieses Vorgehen optimal ist und konstruieren dazu eine Familie von Graphen auf der jede Sequenz von Verfeinerungen $\Omega(m \log n)$ Berechnungsschritte benötigt.

Abstract

The constraint satisfaction problem, the Boolean satisfiability problem and the graph isomorphism problem do not have efficient algorithms. In order to solve these problems, one utilizes heuristic algorithms of polynomial running time. The present thesis studies three classical heuristics for the above-mentioned decision problems and answers the question whether they can be implemented more efficient than with the fastest known algorithms.

The k-consistency heuristic for the constraint satisfaction problem tries to establish local consistency by iteratively propagating new constraints and can be implemented time $O(n^{2k})$. We show that the degree of the polynomial that bounds the running time has to increase linear in k. To achieve this, we prove for a fixed constant $c > 0$ that there is no algorithm of running time $O(n^{ck})$ that decides whether k-consistency can be established. Furthermore, we analyze the propagation process of k-consistency algorithms and prove optimal lower bounds on the number of nested propagation steps.

One heuristic for the Boolean satisfiability problem is to find resolution refutations in which every clause contains at most k literals. Such refutations of width k can be found in time $O(n^{k+1})$ by a simple search procedure. We show for a fixed constant $c > 0$, that it cannot be decided in time $O(n^{ck})$ whether a given formula has a resolution refutation of width k. The lower bounds on the time complexity for k-consistency and bounded width resolution do not rely on complexity theoretic assumptions and demonstrate one of the rare examples where the deterministic time hierarchy theorem can be applied to natural decision problems.

The color refinement heuristic for the graph isomorphism problem computes a stable coloring of the vertices of a graph by iteratively refining the color classes. By refining the color classes in a clever order, the color refinement procedure can be implemented in time $O(m \log n)$ on connected graphs with n vertices and m edges. We show that this refining strategy is optimal. To prove this, we construct graphs where every possible order of refining operations needs at least $\Omega(m \log n)$ computation steps.

Prior Publications

Most of the material in this thesis was previously published in conference proceedings and in a journal. I presented the lower bound on k-consistency at the 27th Annual IEEE Symposium on Logic in Computer Science [Ber12a]. A journal version of this result has been published in Logical Methods in Computer Science [Ber13]. The lower bound on bounded width resolution was published in the proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science [Ber12b]. Together with Oleg Verbitsky I studied the relation between the existential 2-pebble game and arc consistency algorithms [BV13]. This work provides the $k = 2$ case for the lower bound on the number of propagation steps in k-consistency algorithms. For $k > 2$, the lower bound on propagation steps for k-consistency is published at the 20th International Conference on Principles and Practice of Constraint Programming [Ber14].

The lower bound on color refinement algorithms is part of a joint work with Paul Bonsma and Martin Grohe [BBG13]. My main contribution to this work was the construction of the family of graphs for which the lower bound holds. The model of computation and several improvements on the presentation of the proof were provided by my coauthors or evolved in a fruitful discussion with them.

Acknowledgements

First of all I want to thank my supervisor Martin Grohe. The possibility to work at his chair during my undergraduate studies offered many opportunities to get in touch with research. In the four years of my doctoral studies he gave me a lot of freedom in the choice of research topics and patiently discussed my approaches (as a consequence, the open problems I was supposed to work on are still open). I thank everyone else who contributed to this thesis: my co-authors Paul Bonsma and Oleg Verbitsky, the anonymous reviewers of our papers, and Michael Elberfeld, Martin Otto and Erkal Selman for their comments on this thesis.

Special thanks go out to my wife Ruth and our children Lukas and Greta for their love and support. I am deeply indebted to them for moving with me to Aachen two years ago.

Contents

1. Introduction

Many classical decision problems in computer science are not known to be efficiently solvable. Examples include prominent NP-complete problems as Boolean satisfiability (SAT) and the constraint satisfaction problem (CSP), as well as the graph isomorphism problem with its unknown complexity.

Although we cannot solve these problems in polynomial time, there are different ways to tackle them. At first, if we know something about the structure of typical instances, we can use this information to design specific algorithms that take these structural properties into account. If no specific algorithm is available, then one considers general heuristic algorithms. Heuristic algorithms are usually fast (run in polynomial time), but do not solve the problem exactly. Often they are part of an exhaustive search and help to prune the search tree. A powerful heuristic can also be used as a stand-alone algorithm that is able to solve the problem on many (typical) instances. Unsurprisingly, heuristics that are more powerful require more running time and in practice it makes a huge difference if a heuristic can be implemented in linear, quadratic or cubic time – even if this is always polynomial. The present thesis studies the inherent complexity of heuristic approaches and answers the question, "What is the optimal time complexity of an algorithm that implements a certain heuristic?" As indicated by this question, a *heuristic* does not refer to a concrete algorithm but rather to an algorithmic strategy that can be implemented in several ways.

We consider three fundamental heuristics that are known since the 70's: The k-consistency test [Fre78] for the CSP, bounded width resolution [Gal77] for SAT and the color refinement heuristic[1] for the

[1] To our knowledge this heuristic was first mentioned in [CG70].

graph isomorphism problem. All these heuristics apply rather simple algorithmic strategies and turned out to be extremely useful. Most state-of-the-art solver for the CSP utilize the 2-consistency test, also known as arc consistency, and many graph isomorphism algorithms apply the color refinement heuristic. These heuristics have also influenced theoretical research. Bounded width resolution yields the best automatizability results for treelike resolution [BW01] and the fastest algorithm for graph isomorphism [Bab81; BL83] uses color refinement as a subroutine. A common feature of k-consistency, bounded width resolution and color refinement is that they have logical characterizations and can be described by model-theoretic pebbles games. In fact, we derive our lower bounds on the time complexity of these heuristics by applying and developing the tools from finite model theory.

In the next section we present two approaches to obtain lower bounds for algorithms and describe how we apply them to obtain lower bounds for k-consistency, bounded width resolution and color refinement. Afterwards we give an introduction to the logical view on algorithms and present our contributions in this area.

1.1. How to Prove Lower Bounds for Algorithms

There are two ways to obtain lower bounds on the time complexity of algorithmic approaches: proving lower bounds on the underlying decision problem and proving lower bounds on the algorithmic strategy in a restricted model of computation.

Lower bounds on the underlying decision problem

The most general way to determine the inherent complexity of an algorithm is to examine an *underlying decision problem* and to analyze its computational complexity. As a first example, consider the k-consistency test for the constraint satisfaction problem. A CSP-instance is *k-consistent* if every consistent partial assignment of at most $k-1$ variables can be extended to a larger consistent partial assignment. The k-consistency heuristic tries to establish k-consistency

on a given CSP-instance and can be implemented by an $O(N^{2k})$ time algorithm [Coo89].[2] Thus, the underlying decision problem of the k-consistency heuristic is "Given a constraint satisfaction instance. Is it possible to establish k-consistency?" It is clear that every k-consistency algorithm has to solve this decision problem. Hence, lower bounds on the time complexity of this decision problem imply lower bounds for every possible k-consistency algorithm.

Another example is bounded width resolution for the 3-SAT problem. Given a Boolean formula in conjunctive normal form where every clause contains at most three literals (3-CNF). A resolution refutation of this formula has width k, if every clause in the resolution refutation contains at most k literals. Finding resolution refutations of width k is a polynomial time heuristic that can be implemented in $O(N^{k+1})$ by a simple search procedure. The underlying decision problem of this heuristic is, "Given a 3-CNF formula. Does it have a resolution refutation of width at most k?" Again, the underlying decision problem has the property that every algorithm that implements the search for width-k resolution refutations solves this decision problem.

Proving optimal lower bounds on the time complexity of the underlying decision problem is the holy grail in this area of research as it lower bounds the time complexity of every possible algorithm. At the same time this generality is also the biggest drawback as proving optimal complexity lower bounds that do not rely on unproven assumptions is notorious hard. However, obtaining unconditional lower bounds on the time complexity is not impossible as the deterministic time hierarchy theorem shows: For every constant $c \geq 1$ there are problems in PTIME that cannot be computed in time $O(N^c)$. We reduce such problems to the underlying decision problems of k-consistency (Chapter 4) and of width-k resolution (Chapter 6). As a consequence we obtain unconditional lower bounds of $N^{\Omega(k)}$ on the time complexity of these two heuristics.

[2] We denote by N the size of the input encoding. Unless stated otherwise, complexity bounds hold for RAM machines.

Lower bounds on the algorithmic strategy

Another way to study the complexity of heuristics is to investigate the underlying algorithmic strategy. In this approach one assumes that an algorithm performs a certain kind of operations within a restricted framework. This is much in line with how proof complexity aims at proving lower bounds on the complexity of (SAT-solving) algorithms. In this area one views the algorithmic strategy as a proof search method in a formal derivation system. By proving lower bounds on the size of the derivations one obtains lower bounds on the running time of every algorithm that computes such a derivation. Examples for the fruitful connection between proof systems and algorithms include treelike resolution and DPLL-solver, polynomial calculus and the Gröbner basis algorithm, and several proof systems that model hierarchies of relaxations for linear and semidefinite optimization problems.

The first step, designing a formal derivation system, is less obvious than determining the underlying decision problem as one has to design a convincing derivation system that captures every possible algorithm one could think of. The advantage of this approach is twofold. At first, it is easier to prove lower bounds in a restricted model of computation than for decision problems. The second advantage is that these derivation systems follow the structure of the algorithmic strategy. Thus, analyzing the structure of derivations allows to get insight into the structure of the algorithmic approach and explains why a certain strategy is expensive. The idea of proving lower bounds for algorithms in a restricted model of computation also appears outside of proof complexity. A simple and prominent example is the $\Omega(n \log n)$ lower bound for sorting n distinct integers. In this model of computation, the machine has restricted access to the input: for every pair $1 \leq i, j \leq n$ the machine has random access to the information whether ith input number is less than the jth input number. As there are $n!$ input orders, it is not hard to see that the machine has to perform $\log(n!) = \Omega(n \log n)$ comparisons. Note that this lower bound is not for a single algorithm but holds for every algorithm that follows the "comparison sort"-paradigm.

In Chapter 8 we use the method of proving lower bounds on the

algorithmic strategy to obtain lower bounds for the color refinement heuristic for graph isomorphism. The goal of color refinement is to compute a specific partition of the vertex set of a graph, the *coarsest stable partition*, by iteratively refining a partition of the vertices. The running time of this process essentially depends on the order in which the refining operations are chosen. By applying the refinement steps in a clever order, color refinement can be implemented in $O(N \log N)$ [CC82]. We model the refinement process in a formal derivation system and show that there are graphs on which every possible way of refining the vertex partition requires $\Omega(N \log N)$ computation steps. Hence, this lower bound applies to every color refinement algorithm that iteratively refines a partition of the vertex set.

We also apply the technique of proving lower bounds on the algorithmic strategy to the k-consistency heuristic. *Constraint propagation* is an algorithmic technique in constraint solving to reduce the search space by propagating new constraints that follow from previous ones. In fact, the k-consistency test is the most prominent representative in this class of algorithms. As pointed out by Atserias, Kolaitis and Vardi [AKV04], constraint propagation can be seen as a proof system where the lines in the proof correspond to the propagated constraints. Thus, the size of such proofs lower bounds the running time of constraint propagation algorithms. In Chapter 2 we formalize a derivation system that models the behavior of propagation algorithms for k-consistency and observe that the *depth* of such derivations correlates with the time complexity of parallel k-consistency algorithms. Afterwards, in Chapter 3, we prove optimal lower bounds on the depth. As a consequence we obtain optimal lower bounds on the number of propagation steps for sequential and parallel k-consistency algorithms.

1.2. The Logical Approach to Algorithms

Although *finite model theory* is not dedicated to the study of algorithms, it turns out that several techniques from this area can be utilized to study the complexity and expressive power of polynomial

time heuristics. In the sequel we survey relevant results and present our contributions to this area.

Constraint Satisfaction and k-Consistency

Feder and Vardi [FV98] showed that the constraint satisfaction problem can be characterized as the problem of finding a homomorphism between two relational structure A and B. Using this view, Kolaitis and Vardi [KV00] provided a characterization of the k-consistency test in terms of the model-theoretic existential k-pebble game, an Ehrenfeucht-Fraïssé game that corresponds to the existential-positive k-variable fragment of first order logic. They showed that k-consistency can be established, if and only if Duplicator wins the existential k-pebble game on the corresponding structures A and B. Dalmau, Kolaitis and Vardi [DKV02] used this characterization to show that k-consistency is able to solve the constraint satisfaction problem if the core of A has treewidth at most $k-1$. Atserias, Bulatov and Dalmau [ABD07] complemented this result by showing that if the treewidth of the core of A is not bounded by $k-1$, then k-consistency is not guaranteed to solve the constraint satisfaction problem. The pebble game characterization has also been used by Kolaitis and Panttaja [KP03] to prove lower bounds on the complexity of the underlying decision problem. They showed that deciding whether Duplicator has a winning strategy in the existential pebble game is EXPTIME-complete. Hence, the running time of the k-consistency heuristic has to be exponential in k. As this does not exclude a running time of $O(2^k N^2)$, which is still feasible for constant k, they asked whether k-consistency can be implemented in time $N^{o(k)}$.

In Chapter 2 we shed more light on the connection between the existential k-pebble game and k-consistency tests. We show that a winning strategy for Spoiler is intimately connected to the way constraint propagation algorithms try to achieve k-consistency. As a consequence it holds that the minimal number of rounds Spoiler needs to win the existential k-pebble game on A and B, denoted by $\mathrm{depth}^k(\mathsf{A},\mathsf{B})$, corresponds to the minimal number of nested propagation steps every propagation algorithm has to perform. Note that in

general $\text{depth}^k(\mathsf{A},\mathsf{B}) \leq |V(\mathsf{A})|^{k-1}|V(\mathsf{B})|^{k-1}$.[3] Our first main theorem (proven in Chapter 3) shows that this trivial upper bound is tight, and implies a strong lower bound for propagation algorithms.

Theorem 1. *For every integer $k \geq 2$ there exists a constant $\varepsilon > 0$ and two positive integers n_0, m_0 such that for every $n \geq n_0$ and $m \geq m_0$ there exist two binary structures A_n and B_m with $|V(\mathsf{A}_n)| = n$ and $|V(\mathsf{B}_m)| = m$ such that $\text{depth}^k(\mathsf{A}_n,\mathsf{B}_m) \geq \varepsilon n^{k-1}m^{k-1}$.*

In Chapter 4 we prove our second result, which provides a lower bound on the underlying decision problem for k-consistency and answers the open question of Kolaitis and Panttaja [KP03].

Theorem 2. *For every fixed $k \geq 15$ and any $\varepsilon > 0$, the winner of the existential k-pebble game on two given binary structures A and B cannot be determined in time $O((\|\mathsf{A}\| + \|\mathsf{B}\|)^{\frac{k-2}{12}-\varepsilon})$ on deterministic multi-tape Turing machines.*

Boolean Satisfiability and Bounded Width Resolution

Besides finite model theory, proof complexity plays an important role in the study of algorithms, especially for the Boolean satisfiability problem. The main reason for this is that many SAT-solving algorithms can be viewed as a proof search method, mostly within the resolution calculus. Pudlak [Pud00] introduced a Prover-Adversary game for resolution to prove lower bounds on the length of resolution refutations. Atserias and Dalmau [AD08] showed that if every clause in the refutation contains at most k literals, then this Prover-Adversary game equals the existential $(k+1)$-pebble game on the natural structure encoding of 3-CNF formulas. Hence, the power and the complexity of finding resolution refutations of bounded width can also be analyzed by inspecting a model-theoretic pebble game. Using this pebble game we prove the following lower bound on the complexity of the width-k heuristic.

Theorem 3. *For every integer $k \geq 15$ and $\varepsilon > 0$, to decide whether a given 3-CNF formula Γ has a resolution refutation of width k requires $\Omega(\|\Gamma\|^{\frac{k-2}{12}-\varepsilon})$ computation steps on multi-tape Turing machines.*

[3] By $V(\mathsf{A})$ we denote the domain of structure A.

It follows that there is no significant faster way to decide the existence of a width-k refutation than exhaustively searching for it. Additionally, we investigate the computational complexity of the decision problem where the parameter k is part of the input. Deciding whether there is a resolution refutation of arbitrary width is co-NP complete, because this equivalent to deciding whether the formula is unsatisfiable. Our fourth main theorem shows that the problem gets much harder if the width is required to be bounded by some given number.

Theorem 4. *Given a 3-CNF formula* Γ *and a parameter* k*. It is* EXPTIME*-complete to decide whether* Γ *has a resolution refutation of width* k*.*

We prove this theorem by showing that determining the winner in the existential pebble game on 3-CNF formulas is EXPTIME-complete, which implies that playing the existential pebble game on 3-CNF formulas is as hard as on arbitrary structures. The last result in this area is on regular resolution. A resolution refutation is *regular* if on every path in the proof-dag no variable has been used twice by the resolution rule. Regular resolution is also a sound and complete refutation system for SAT and therefore it is also co-NP complete to decide whether a given formula has a regular resolution refutation. However, bounding the width of regular resolution refutations leads to a different complexity.

Theorem 5. *Given a 3-CNF formula* Γ *and a parameter* k*. It is* PSPACE*-complete to decide whether* Γ *has a regular resolution refutation of width* k*.*

This result illuminates the differences between regular and full resolution from a complexity theoretic perspective. The proof uses a regular variant of the existential pebble game provided by Hertel and Urquhart [Her08; Urq12].

Graph Isomorphism and the Weisfeiler-Lehman Algorithm
The k-dimensional Weisfeiler-Lehman algorithm is a polynomial time heuristic for graph isomorphism and can be implemented in time $O(n^k \log n)$ on n-vertex graphs. This algorithm equals color refinement for $k = 2$ and generalizes the color refinement heuristic to higher dimensions by computing a coarsest stable partition of all $(k-1)$-tuples of vertices. Immerman and Ladner [IL90] introduced a combinatorial pebble game that characterizes equivalence in k-variable counting logic C^k and used this game to show that two graphs satisfy the same C^2 sentences if and only if they cannot be distinguished by the color refinement heuristic. Cai, Fürer and Immerman [CFI92] extended this result to $k > 2$: the k-dimensional Weisfeiler-Lehman heuristic can be characterized in terms of the equivalence problem for C^k. They used the pebble game characterization of C^k to show that there are pairs $\mathsf{G}_n, \mathsf{H}_n$ of non-isomorphic n-vertex graphs that satisfy the same C^k sentences for all $k = o(n)$. Therefore, the k-dimensional Weisfeiler-Lehman algorithm fails to distinguish them.

The combinatorial characterization of the Weisfeiler-Lehman algorithm was subsequently used to prove lower bounds on the computational complexity of this algorithm. Grohe [Gro99] showed that, for every constant $k \geq 2$, deciding whether two graphs are C^k-equivalent is complete for PTIME. Hence, assuming NC $\neq$ PTIME, the Weisfeiler-Lehman algorithm cannot be parallelized. Afterwards, Fürer [Für01] used the Cai-Fürer-Immerman construction to prove a linear lower bound on the number of iterated refinement steps in the Weisfeiler-Lehman algorithm. The result can be viewed as a non-parallelizability result in a restricted model of computation. The logical characterization also leads to upper bounds on the power of the Weisfeiler-Lehman heuristic. The most remarkable result of this kind was proven by Grohe [Gro12], who showed that the Weisfeiler-Lehman algorithm is able to solve graph isomorphism on every graph class that excludes some minor. Examples of such graph classes include graphs of bounded treewidth and planar graphs.

In Chapter 8 we prove lower bounds for the color refinement heuristic (the 2-dimensional Weisfeiler-Lehman algorithm) in a restricted model of computation. As mentioned in the previous section, we es-

tablish the notion of *refinement algorithms* that iteratively refine a partition of the vertices.

Theorem 6. *Every refinement algorithm requires at least* $\Omega(m \log n)$ *computation steps on graphs with* n *vertices and* m *edges.*

This is tight as it matches the $O(m \log n)$ upper bound. Although we do not use the characterization as equivalence problem for the 2-variable counting logic C^2, our graph construction makes essential use of the gadgets of Cai, Fürer and Immerman [CFI92] and provide another application of this tool. What remains open in this area of research is the complexity of the k-dimensional Weisfeiler-Lehman algorithm for $k > 2$. Can the algorithm be implemented faster than $O(n^k \log n)$? Again, there are two ways to tackle this problem: by considering the underlying decision problem and by analyzing the algorithmic structure. To lower bound the complexity of the underlying decision problem one has to investigate the equivalence problem for C^k.

Open Question 1.1. What is the time complexity of deciding C^k-equivalence?

Currently, it is not excluded that this problem can be solved in linear time for every fixed k, and it is even not known whether some established complexity theoretic assumption implies a super-linear lower bound. If k is part of the input, this problem can be solved in exponential time and is a candidate for being EXPTIME-complete. Another approach to lower bound the complexity of Weisfeiler-Lehman is to analyze the number of refinement iterations. The trivial upper bound is $O(n^{k-1})$ and the best lower bound is $\Omega(n)$ [Für01]. As the number of refinement iterations needed to distinguish two graphs equals the minimal number of rounds Spoiler needs to win the C^k-game, we finish the introduction with our second open question that addresses this parameter.

Open Question 1.2. What is the minimal number of rounds Spoiler needs to win the C^k-game on two given graphs?

Part I.

Constraint Satisfaction

2. Constraint Propagation

Constraint satisfaction is a framework to model and solve general search problems. A constraint satisfaction instance consists of a set X of variables over a domain D and a set of constraints C that restrict possible assignments of the variables. The *constraint satisfaction problem* (CSP) is to find an assignment of the variables with values from D that satisfies all constraints. Many search problems can naturally be expressed in this framework. As an example consider the task of finding a solution to a system of linear equations over the integers: the variables are precisely the variables in the linear equations, the domain is the set of integers and there is one constraint for every equation stating that this equation has to be satisfied.

Two further examples come from graph theory. The first one is to find a k-clique in a graph. There are k variables and the domain is the vertex set of the graph. The constraints formalize that two distinct variables only get adjacent vertices as assignment. Hence every satisfying assignment corresponds to a k-clique. Our last example is the 3-colorability problem. We want to color the vertices of a graph with three colors such that no two adjacent vertices get the same color. In the corresponding constraint satisfaction instance there is one variable for each vertex and the domain is a set of three colors. Additionally, there is one constraint for every pair of adjacent vertices stating that the vertices must get different colors. The main difference between these three examples is the size of the domain. For the system of equations, the domain is infinite. In the k-clique example, the domain is finite but depends on the input size. For 3-colorability the domain is fixed and contains three elements. Here we are only interested in CSPs with finite (but not necessarily fixed) domains.

A *constraint satisfaction instance* I is a triple $I = (X, D, C)$ where X is a set of *variables*, the *domain* D is a set of values and C is

a set of constraints. A *constraint* is a tuple $((x_1, \ldots, x_r), R)$ where $x_1, \ldots, x_r \in X$ are variables and $R \subseteq D^r$ is an r-ary relation that lists all allowed combinations of assignments for the variables $x_1, \ldots, x_r$. Thus we say that an assignment $\alpha\colon X \to D$ *satisfies* the constraint if $(\alpha(x_1), \ldots, \alpha(x_r)) \in R$. A satisfying assignment of a constraint satisfaction instance $I = (X, D, C)$ is an assignment $\alpha\colon X \to D$ that satisfies every constraint $c \in C$.

CSP	
Input:	A constraint satisfaction instance I.
Question:	Does there exists a satisfying assignment for I?

Since we are interested in the complexity of algorithms for the CSP we have to reason about the size of a CSP-instance (in the RAM model). Relations are encoded by listing their tuples, hence the size of a relation R of arity r is $\|R\| := r \cdot |R|$. A straightforward encoding for the CSP is to store all the sets and relations the CSP-instance contains. The size of this encoding is $O(|X| + |D| + |C| + \sum_{(\bar{x},R)\in C} \|R\|)$. However, this might be very wasteful if one relation R occurs in many constraints. A better idea is to store every relation R only once and equip every constraint $(\overline{x}, R)$ with a pointer to R. Using this encoding we define the size of a CSP-instance I to be

$$\|I\| := |X| + |D| + \sum_{((x_1,\ldots,x_r),R)\in C} r + \sum_{R\in\Gamma(I)} \|R\|,$$

where $\Gamma(I)$ is called the *constraint language of* I and denotes the set of relations occurring in the constraints of I. This representation is quadratically more succinct than the straightforward one as the following simple example shows. Take n variables and the integers $1, \ldots, n$ as domain. There are $\binom{n}{2}$ constraints stating that two different variables get different values, hence the constraint language is $\{R\}$ with $R := \{(i, j) \in [n]^2 \mid i \neq j\}$.[1] The straightforward encoding has size $\Theta(n^4)$ whereas the succinct representation as defined above

[1] For every positive integer n, $[n] := \{1, \ldots, n\}$.

only requires $\Theta(n^2)$. Note that the succinct definition of size is also closer to practical implementations because it seems implausible to attach, for example, the definition of inequality to every inequality constraint or the set of truth assignments of a disjunctive clause to every clause in a SAT-instance.

Homomorphisms and the CSP

As pointed out by Feder and Vardi [FV98], another way to formalize the constraint satisfaction problem is via the structure homomorphism problem HOM:

HOM	
Input:	Two finite relational σ-structures A and B.
Question:	Is there a homomorphism from A to B?

A homomorphism h from a σ-structure A to a σ-structure B is a mapping from the domain $V(\mathsf{A})$ of A to the domain $V(\mathsf{B})$ of B such that for every relation symbol $\dot{R} \in \sigma$ and all tuples $(x_1, \ldots, x_r) \in \dot{R}^{\mathsf{A}}$ it holds that $(h(x_1), \ldots, h(x_r)) \in \dot{R}^{\mathsf{B}}$. The size of a structure A, denoted by $\|\mathsf{A}\|$, is the size of its domain $V(\mathsf{A})$ plus the size of every relation $\dot{R}^{\mathsf{A}}$. Hence, the input size of the homomorphism problem is $\|\mathsf{A}\| + \|\mathsf{B}\|$. There is a tight connection between HOM and CSP that preserves the solution space as well as the input size:

(1) For every CSP-instance $I = (X, D, C)$ there are two relational σ-structures, A with domain X and B with domain D, of size $\|\mathsf{A}\| + \|\mathsf{B}\| = \|I\|$ such that every mapping $\alpha\colon X \to D$ is a satisfying assignment for I if, and only if, it is a homomorphism from A to B.

(2) For every pair of relational σ-structures A and B, there is a CSP-instance $I = (V(\mathsf{A}), V(\mathsf{B}), C)$ of size $\|I\| = \|\mathsf{A}\| + \|\mathsf{B}\|$ such that every mapping $\alpha\colon V(\mathsf{A}) \to V(\mathsf{B})$ is a homomorphism from A to B if, and only if, it is a satisfying assignment for I.

To derive the two σ-structures from a given CSP-instance (1) we set $\sigma = \{\dot{R} \mid R \in \Gamma(I)\}$. The σ-structure A has X as domain and interprets the relation symbols as $\dot{R}^{\mathsf{A}} := \{\overline{x} \mid (\overline{x}, R) \in C\}$. The σ-structure B is defined over the domain D and contains the relations $\dot{R}^{\mathsf{B}} := R$ for all $R \in \Gamma(I)$. It is easy to verify that every homomorphism from A to B corresponds to a solution of I and that the size of the structures matches the size of I. For the other direction (2) we add one constraint $(\overline{x}, \dot{R}^{\mathsf{B}})$ for every relation symbol $\dot{R} \in \sigma$ and every tuple $\overline{x} \in \dot{R}^{\mathsf{A}}$.

Both viewpoints, CSP and HOM, have their advantages and we will use them interchangeably. Especially, we will often call the elements of A *variables* and the elements of B *values*. In the following we mainly consider *binary* CSP-instances where the arity of every constraint is at most two. In terms of the homomorphism problem this corresponds to vertex- and edge-colored digraphs.

2.1. Local Consistency Heuristics

The constraint satisfaction problem can be solved in exponential time by exhaustive search over all possible assignments. *Constraint propagation* is a technique that is used to speed up the exhaustive search by restricting the search space in advance. This is done by propagating new constraints that follow from previous ones, as the following simple example illustrates. Assume that the domain is the set of integers from 1 to 100 and the constraints include (among others)

$$x + y = 100 \text{ and}$$
$$x \text{ is odd.}$$

First we note that an exhaustive search only has to test assignments that map x to an odd number. Furthermore, we can propagate the constraint "y is odd" that is implied by these two constraints. Using this additional constraint, the search space can be pruned even further because the exhaustive search only has to test odd values for y. Constraint propagation techniques can also be used to solve the

CSP or to detect its unsatisfiability. For example, if the CSP-instance above also contains

$$y = 2z,$$

then we can propagate "y is even". Since this contradicts the constraint "y is odd", we know – without doing any search – that the CSP-instance must be unsatisfiable. The idea of propagating new constraints is rather general and, of course, the constraint $(\overline{x}, \{\overline{a} \mid$ the mapping $x_i \mapsto a_i$ is a satisfying assignment to the CSP$\})$ is always derivable from the given constraints and prevents any search. However, solving the CSP only by propagation is considered to be less efficient than exhaustive search. There is a trade-off between the time complexity of the propagation phase and the achieved speed-up for the search phase. In practice, the solution to this trade-off is to use fast (polynomial-time) propagation heuristics followed by an exhaustive search.

Local consistency techniques are the most prominent constraint propagation techniques. The overall goal is to propagate new constraints to achieve some kind of consistency on small parts of the constraint satisfaction instance. On the other hand, if local inconsistencies are detected, it follows that the CSP-instance is also globally inconsistent and hence unsatisfiable. In the last part of this section we introduce well-known generic local consistency techniques using the notion of the homomorphism problem HOM. Algorithms for these techniques are discussed in Section 2.2.

The Arc Consistency Heuristic

One approach to shrink the search space of a HOM-instance is to reduce the set of feasible values a variable can get. Let a *domain* $D_x \subseteq V(\mathsf{B})$ of a variable $x \in V(\mathsf{A})$ be a set of values such that for every homomorphism $h\colon \mathsf{A} \to \mathsf{B}$ it holds that $h(x) \in D_x$. Arc consistency is a constraint propagation heuristic introduced in [Mac77] to find small domains for binary[2] structures. Initially, every variable $x \in V(\mathsf{A})$ can

[2]There are also variants for CSPs of higher arity, sometimes called *hyperarc consistency* or *generalized arc consistency*, which we will not consider here.

get every value $d \in V(\mathsf{B})$, hence $D_x = V(\mathsf{B})$. The first step of the arc consistency heuristic is to ensure *node consistency*, that is, D_x is the set of vertices in B that are colored with the same color as x. The second step is to iteratively shrink the domains according to the following rule.[3]

> If for an $a \in D_x$ there exists a variable $y \in V(\mathsf{A})$ and a binary relation $\dot{R} \in \widetilde{\sigma}$ such that $(x, y) \in \dot{R}^{\mathsf{A}}$ and $(a, b) \notin \dot{R}^{\mathsf{B}}$ for all $b \in D_y$, then delete a from D_x.

The rule is easily seen to be sound: For every mapping $h\colon V(\mathsf{A}) \to V(\mathsf{B})$ with $h(x) = a$ and $h(y) \in D_y$ it holds that $(x, y) \in \dot{R}^{\mathsf{A}}$ but $(h(x), h(y)) \notin \dot{R}^{\mathsf{B}}$. Hence, no homomorphism can map x to a. A pair of binary structures augmented with a set of domains is *arc consistent* if the above rule cannot be applied and all domains are nonempty. We say that arc consistency *can be established* for A and B, if there exists a set of domains such that A and B augmented with these domains is arc consistent.

The k-Consistency Heuristic

Arc consistency iteratively rules out assignments for one variable that cannot be extended to a partial homomorphism on two variables. This concept can naturally be extended to higher dimensions. To do this we need to extend the definition of domains to tuples of variables. Let $\mathcal{D}^{k-1}$ be a family of ℓ-partial assignments ($\ell \leq k-1$) from $V(\mathsf{A})$ to $V(\mathsf{B})$. A partial mapping $\alpha\colon V(\mathsf{A}) \to V(\mathsf{B})$ is *consistent* with $\mathcal{D}^{k-1}$ if for every $X \subseteq V(\mathsf{A})$ with $|X| < k$ the restriction of α to variables from X is contained in $\mathcal{D}^{k-1}$. We say that $\mathcal{D}^{k-1}$ is a $(k-1)$-*dimensional domain* for a HOM-instance (A, B) if every homomorphism $h\colon \mathsf{A} \to \mathsf{B}$ is consistent with $\mathcal{D}^{k-1}$. Note that for $k = 2$ the 1-dimensional domain $\mathcal{D}^1$ can be translated to variable domains defined above by setting $D_x = \{a \mid (x \mapsto a \in \mathcal{D}^1)\}$.

[3] We use the following notion: For a signature $\sigma = \{\dot{P}_1, \ldots, \dot{P}_t, \dot{E}_1, \ldots, \dot{E}_\ell\}$ with unary relations $\dot{P}_i$ and binary relations $\dot{E}_j$ we let $\widetilde{\sigma} = \sigma \cup \{\widetilde{E}_1, \ldots, \widetilde{E}_\ell\}$ and define $\widetilde{E}_i^{\mathsf{A}} = \{(y, x) \mid (x, y) \in \dot{E}_i^{\mathsf{A}}\}$ for every σ-structure A.

The k-consistency heuristic is a constraint propagation technique to iteratively shrink $(k-1)$-dimensional domains. Generally, it is defined for structures of arbitrary arity but it takes only relations of arity at most k into account. Initially, the domain $\mathcal{D}^{k-1}$ contains all partial homomorphism $h\colon \mathsf{A} \to \mathsf{B}$ with $|h| < k$. The domain is subsequently restricted by the following inference rule:

> If for an $h \in \mathcal{D}^{k-1}$ there exists a variable $y \in V(\mathsf{A})$ such that for all $b \in V(\mathsf{B})$ the set $h \cup \{y \mapsto b\}$ is not a partial homomorphism consistent with $\mathcal{D}^{k-1}$, then delete h from $\mathcal{D}^{k-1}$.

Two structures equipped with a nonempty $(k-1)$-dimensional domain are *k-consistent*, if the above rule cannot be applied and hence every $(\ell-1)$-partial homomorphism in $\mathcal{D}^{k-1}$ can be extended to a consistent ℓ-partial homomorphism.[4] We say that k-consistency *can be established* for two structures if there exists a nonempty $(k-1)$-dimensional domain such that the structures together with the domain are k-consistent. For $k = 2$ the initialization of $\mathcal{D}^1$ is the same as ensuring node consistency and the inference rule is just a reformulation of the inference rule for arc consistency. Hence, 2-consistency can be established if and only if arc consistency can be established.

The next level, 3-consistency, shrinks 2-dimensional domains and also involves ternary relations of A and B. However, the restriction of 3-consistency to binary structures has gained considerable attention in the past and is known under the name *path consistency*. Higher levels of consistency are more of theoretical interest. In general, n-consistency can be established on CSP-instances with n variables, if and only if the instance is satisfiable. Hence, n-consistency can be used to *solve* the CSP, although this is prohibitively expensive. On restricted classes of structures there might be a constant k such that k-consistency solves the CSP on this class of structures. That is, k-consistency can be established if and only if the CSP is satisfiable. Dalmau, Kolaitis and Vardi [DKV02] showed that is the case if the

[4] In many textbooks the term "strongly k-consistent" is used for what we call only "k-consistent" here.

treewidth of the homomorphic core of A is bounded by k. Atserias, Bulatov and Dalmau [ABD07] complemented this result and showed that if the treewidth of the homomorphic core of A is not bounded by k, then there are structures B such that k-consistency does not solve the CSP on A and B. This gives a precise characterization (in terms of structure A) of the CSP-instances where k-consistency can be used to solve the CSP in polynomial time.

2.2. Propagation Algorithms

In order to analyze the algorithmic complexity of the k-consistency heuristic in its full generality, we consider the following decision problem.

k-Cons	
Input:	Two binary relational structures A and B.
Question:	Can k-consistency be established for A and B?

We define a *k-consistency algorithm* to be an algorithm that solves k-Cons. This definition is the most general since a k-consistency algorithm could be an arbitrary Turing machine that reads A and B, does some weird computation and either accepts or rejects the input. Hence, lower bounds on the computational complexity of k-Cons immediately imply lower bounds on the worst-case running time of any possible algorithm. One of our main results concerns the computational complexity of k-Cons and is discussed in Section 2.4. Note that we do not require an algorithm to explicitly output a consistent domain. The main reason for this is to rule out pathological examples. For example, the size of every $(k-1)$-dimensional domain for two structures with empty relations is $\Omega(|V(\mathsf{A})|^{k-1}|V(\mathsf{B})|^{k-1})$.

To gain insights into the structure of practical k-consistency algorithms we also consider a more restricted model of computation. Every arc-, path- and k-consistency procedure described in the literature tries to find a locally consistent instance by iteratively propagating new constraints using the inference rule. The main differences

between these algorithms are the underlying data structure, which includes the representation of $\mathcal{D}^{k-1}$, and the order in which they apply the inference rule. This motivates the definition of propagation algorithms. Intuitively, a propagation algorithm decides k-Cons by iteratively applying the inference rule for k-consistency. To formalize this notion we introduce a simple inference system to derive inconsistent local assignments. Given two σ-structures A and B, every line of our derivation system is a partial mapping from $V(\mathsf{A})$ to $V(\mathsf{B})$. The axioms are all partial mappings $p\colon V(\mathsf{A}) \to V(\mathsf{B})$ that are not partial homomorphisms. We have the following derivation rule to derive a new inconsistent assignment p. For all partial mappings $p'_i \subseteq p$, $x \in V(\mathsf{A})$ and $V(\mathsf{B}) = \{a_1, \ldots, a_n\}$:

$$\frac{p'_1 \cup \{x \mapsto a_1\} \quad \cdots \quad p'_n \cup \{x \mapsto a_n\}}{p} \tag{2.1}$$

A *CSP-derivation* of p is a sequence $(p_1, \ldots, p_\ell = p)$ such that every p_i is either an axiom or derived from lines p_j, $j < i$, via the derivation rule (2.1). A *CSP-refutation* is a CSP-derivation of $\emptyset$. Every derivation of p can naturally be seen as a directed acyclic graph where the nodes are labeled with lines from the refutation, one node of in-degree 0 is labeled with p and all nodes of out-degree 0 are labeled with axioms. If p_i is derived from $p_{j_1}, \ldots, p_{j_n}$ using (2.1), then there is an arc from p_i to each $p_{j_1}, \ldots, p_{j_n}$.

Given a CSP-refutation P, we let $\mathrm{Prop}(P)$ be the set of propagated mappings $p \in P$, which are derived via (2.1). We define the *width* of a derivation P to be $\mathrm{width}(P) = \max_{p \in \mathrm{Prop}(P)} |p|$.[5] Furthermore, $\mathrm{depth}(P)$ denotes the *depth* of P which is the number of edges on the longest path in the dag associated with P. This measure characterizes the maximum number of nested propagation steps in P. The next lemma follows directly from the definitions as the derivation rule (2.1) is just a reformulation of the "inference rule" for k-consistency from Section 2.1. For the sake of completeness a proof is nevertheless given.

[5] Note that this implies $|p| \leq \mathrm{width}(P) + 1$ for all axioms p used in the derivation P. However, the size of the axioms can always be bounded by the maximum arity of the relations in A and B.

Lemma 2.1. *Given two relational structures* A *and* B*. There is a CSP-refutation of width* $k-1$ *if and only if* k*-consistency cannot be established.*

Proof. Let $P = (p_1, \ldots, p_\ell)$ by a CSP-refutation of width $k-1$. Assume for contradiction that k-consistency can be established and let $\mathcal{D}^{k-1}$ be a $(k-1)$-dimension domain such that the structures augmented with this domain are k-consistent. We show by induction over $i \in [\ell]$ that p_i is not consistent with $\mathcal{D}^{k-1}$. First, if p_i is an axiom, then p_i is not a partial homomorphism by definition and hence $p_i \notin \mathcal{D}^{k-1}$. If p_i is derived from $p'_1 \cup \{x \mapsto a_1\}, \ldots, p'_n \cup \{x \mapsto a_n\}$ according to rule (2.1), then by induction hypothesis no $p'_j \cup \{x \mapsto a_j\}$ is consistent with $\mathcal{D}^{k-1}$. Since $p'_j \subseteq p_i$ it follows that all $p_i \cup \{x \mapsto a_j\}$ are not consistent with $\mathcal{D}^{k-1}$ either. Hence, $p_i \notin \mathcal{D}^{k-1}$ by the inference rule for k-consistency. It follows that $p_\ell = \emptyset$ is not consistent with $\mathcal{D}^{k-1}$ and thus no mapping is consistent with $\mathcal{D}^{k-1}$. This contradicts the assumption that $\mathcal{D}^{k-1} \neq \emptyset$.

For the other direction suppose that the k-consistency procedure iteratively deletes partial assignments from the set $\mathcal{D}^{k-1}$ of all l-partial homomorphisms ($l < k$) according to the propagation rule and ends up with the empty set. It follows inductively, that there exists a derivation of p for all l-partial mappings ($l < k$) that are deleted from $\mathcal{D}^{k-1}$ by the k-consistency procedure. Since initially $\emptyset \in \mathcal{D}^{k-1}$, it must be deleted at some time and is thus derivable in our system. □

Recall that a (sequential/parallel) propagation algorithm was supposed to be an algorithm that iteratively applies the inference rule for k-consistency in order to solve k-Cons. Hence, if k-consistency cannot be established for two given structures, then a propagation algorithm produces (in passing) a CSP-refutation P. Furthermore, the total number of propagation steps performed by this algorithm is $|\operatorname{Prop}(P)|$ and the maximum number of nested propagation steps (which have to be computed sequentially) is $\operatorname{depth}(P)$.

Definition 2.2. Let A and B be two relational structures such that k-consistency cannot be established. We define the *propagation amount*

$$\mathrm{prop}^k(\mathsf{A},\mathsf{B}) = \min_P |\operatorname{Prop}(P)|$$

and the *propagation depth*

$$\mathrm{depth}^k(\mathsf{A},\mathsf{B}) = \min_P \operatorname{depth}(P)$$

where the minimum is taken over all CSP-refutations P of width at most $k-1$.

The propagation amount and the propagation depth of two structures are crucial measures for the analysis of propagation algorithms since the number of propagation steps performed by any sequential propagation algorithm is lower bounded by $\mathrm{prop}^k(\mathsf{A},\mathsf{B})$ and the number of sequential propagation steps that have to be performed by any parallel propagation algorithm is lower bounded by $\mathrm{depth}^k(\mathsf{A},\mathsf{B})$. Note that in general

$$\mathrm{depth}^k(\mathsf{A},\mathsf{B}) \leq \mathrm{prop}^k(\mathsf{A},\mathsf{B}) \leq |V(\mathsf{A})|^{k-1}|V(\mathsf{B})|^{k-1}.$$

The main theorem of Chapter 3 is that this bound is tight and hence every sequential or parallel propagation algorithm for k-consistency has to perform $\Omega(|V(\mathsf{A})|^{k-1}|V(\mathsf{B})|^{k-1})$ nested propagation steps. In the last part of this section we describe algorithms for arc consistency and k-consistency which achieve the best known worst-case running time.

Upper Bounds for k-Consistency

Theorem 2.3. *There is a propagation algorithm that decides whether k-consistency can be established for two given structures A and B with worst-case running time $O(\|\mathsf{A}\|^k \cdot \|\mathsf{B}\|^k) = O(N^{2k})$.*

Proof. To prove Theorem 2.3 we provide a reduction to a set of Horn clauses of size $O(\|\mathsf{A}\|^k \cdot \|\mathsf{B}\|^k)$. The theorem then follows by applying any linear-time solver for Horn formulas.

The Horn clauses are defined over Boolean variables $\mathsf{X}[p]$ for partial assignments $p\colon V(\mathsf{A}) \to V(\mathsf{B})$, $|p| \leq k$, with the intended meaning "p is inconsistent." The following set of Horn clauses ensures that $\mathsf{X}[p]$ has to be true if and only if some $p' \subseteq p$ has a CSP-derivation of width at most $k-1$.

$$\mathsf{X}[p], \qquad |p| \leq k,\ p \text{ is not a partial homomorphism}; \tag{2.2}$$

$$\mathsf{X}[p] \text{ :- } \mathsf{X}[p'], \qquad |p| \leq k,\ p' \subset p; \tag{2.3}$$

$$\mathsf{X}[p] \text{ :- } \mathsf{X}[p \cup \{y \mapsto a_1\}], \ldots, \mathsf{X}[p \cup \{y \mapsto a_n\}], \qquad |p| < k,\ y \in V(\mathsf{A}) \text{ and } V(\mathsf{B}) = \{a_1, \ldots, a_n\}. \tag{2.4}$$

The first set of clauses (2.2) ensures that $\mathsf{X}[p]$ has to be true for all axioms p. The other clauses (2.3) and (2.4) ensure that $\mathsf{X}[p]$ has to be true if p can be derived via (2.1). Finally, we add the clause

$$\text{:- } \mathsf{X}[\emptyset] \tag{2.5}$$

and conclude that the entire Horn formula is unsatisfiable if and only if $\emptyset$ has a CSP-derivation width at most $k-1$. Note that Horn-Sat-solvers iteratively propagate truth values for the variables on the left hand side of the clauses. Especially, setting the variable $\mathsf{X}[p]$ in (2.4) to true corresponds to applying the inference rule. Hence, applying a Horn-Sat-solver to this Horn formula also fits into our definition of "propagation based algorithms." □

Upper Bounds for Arc Consistency

For $k > 2$ the upper bound provided in Theorem 2.3 matches the worst-case running time of the best known k-consistency algorithms. For $k = 2$ (arc consistency) the k-consistency test can be implemented more efficiently as the following theorem shows.

Theorem 2.4. *There is a propagation algorithm that decides whether arc consistency can be established for two given structures* A *and* B

with worst-case running time $O(\|\mathsf{A}\| \cdot \|\mathsf{B}\|) = O(N^2)$.

Proof. We fix a signature $\sigma = \{P_1, \ldots, P_s, E_1, \ldots, E_t\}$ with unary relations P_i and binary relations E_i. Analogously to the proof of Theorem 2.3 we provide a reduction to a set of Horn clauses of size $O(\|\mathsf{A}\| \cdot \|\mathsf{B}\|)$.[6] As before, we have variables $\mathsf{X}[\emptyset]$ and $\mathsf{X}[x \mapsto a]$ with the intended meaning that these partial mappings are inconsistent. Additionally, we use variables $\mathsf{X}[x \mapsto a,\ y \mapsto *]$, for all $x, y \in V(\mathsf{A})$ and $a \in V(\mathsf{B})$, to denote that for all b the mapping $\{x \mapsto a,\ y \mapsto b\}$ is inconsistent. The Horn formula consists of the following set of clauses.

$$\mathsf{X}[x \mapsto a], \tag{2.6}$$
$$\text{if } P_i^{\mathsf{A}}(x) \neq P_i^{\mathsf{B}}(a) \text{ for some } P_i \in \sigma;$$
$$\mathsf{X}[x \mapsto a] \text{ :- } \mathsf{X}[x \mapsto a,\ y \mapsto *], \tag{2.7}$$
$$\text{for all } E \in \widetilde{\sigma},\ (x, y) \in E^{\mathsf{A}} \text{ and } a \in V(\mathsf{B});$$
$$\mathsf{X}[x \mapsto a,\ y \mapsto *] \text{ :- } \mathsf{X}[y \mapsto b_1], \ldots, \mathsf{X}[y \mapsto b_s], \tag{2.8}$$
$$\text{for all } E \in \widetilde{\sigma},\ (x, y) \in E^{\mathsf{A}} \text{ and}$$
$$a \in V(\mathsf{B}) \text{ with } N^{E^{\mathsf{B}}}(a) = \{b_1, \ldots, b_s\};$$
$$\mathsf{X}[\emptyset] \text{ :- } \mathsf{X}[x \mapsto a_1], \ldots, \mathsf{X}[x \mapsto a_n] \tag{2.9}$$
$$V(\mathsf{B}) = \{a_1, \ldots, a_n\},\ x \in V(\mathsf{A});$$
$$\text{:- } \mathsf{X}[\emptyset]. \tag{2.10}$$

The structure of the clauses is similar to the one in the proof of Theorem 2.3: the clauses (2.6) state that the one-dimensional axioms $x \mapsto a$ are derivable and the clauses (2.7)–(2.9) state that partial mappings are derivable via enforcing (2.1). Finally, (2.10) expresses that the empty clause is not derivable. The main difference to the Horn formula for 2-consistency in the proof of Theorem 2.3 is that axioms p of size $|p| = 2$ are not explicitly expressed in variables $\mathsf{X}[p]$ but rather implicitly used in the clauses (2.8). The premise of (2.8) only ranges over the corresponding neighbors of a and is therefore

[6] The idea of reducing a CSP instance to a Horn formula to obtain upper bounds for arc consistency was first made explicit by Kasif [Kas90].

closer to the view on the inference rule for arc consistency as presented in Section 2.1.

The size of the clause set can be estimated as follows: (2.6) contains $O(|V(\mathsf{A})| \cdot |V(\mathsf{B})|)$ clauses of size $O(1)$, (2.7) contains $O(\|\mathsf{A}\| \cdot |V(\mathsf{B})|)$ clauses of size $O(1)$ and (2.9) contains $O(|V(\mathsf{A})|)$ clauses of size $O(|V(\mathsf{B})|)$. Finally, the size of the clause set (2.8) can be estimated by

$$\sum_{E \in \widetilde{\sigma}} \sum_{(x,y) \in E^{\mathsf{A}}} \sum_{a \in V(\mathsf{B})} |N^{E^{\mathsf{B}}}(a)| = \sum_{E \in \widetilde{\sigma}} |E^{\mathsf{A}}| \cdot |E^{\mathsf{B}}| = O(\|\mathsf{A}\| \cdot \|\mathsf{B}\|).$$

Thus, the overall size of this clause set is bounded by $O(\|\mathsf{A}\| \cdot \|\mathsf{B}\|)$. Since this set can also be computed in time $O(\|\mathsf{A}\| \cdot \|\mathsf{B}\|)$, Theorem 2.4 follows by solving the Horn-Sat instance in linear time. □

Remark 2.5. For general binary structures Theorem 2.4 states the best known worst-case running time for arc consistency algorithms. The upper bound can be slightly improved if the underlying structures contains only one binary relation E. In that case Berkholz and Verbitsky [BV13] showed that the arc consistency test can be implemented in $O(|V(\mathsf{A})| \cdot |E^{\mathsf{B}}| + |V(\mathsf{B})| \cdot |E^{\mathsf{A}}|)$.

Parallel Algorithms

In general, the decision problem k-Cons is complete for PTIME under LOGSPACE-reductions. This was first shown for arc consistency ($k = 2$) by Kasif [Kas90] and later extended to every constant k by Kolaitis and Panttaja [KP03]. Hence, unless PTIME = NC, k-consistency is not efficiently parallelizable. However, there have been some approaches in the literature to design parallel algorithms for k-consistency in order to gain some speedup in running time (e. g. [SH87; Sus+91]). The common feature of these approaches is that they use a polynomial number of processors ($n^{O(k)}$) to apply the propagation rule in parallel. That is, every processor is responsible for computing one instance of the propagation rule. Using this approach we get the following lemma.

Lemma 2.6. *There is a CRCW-PRAM that takes two structures* A, B *and computes* k-Cons *in time* $O(\mathrm{depth}^k(\mathsf{A}, \mathsf{B}))$ *using* $n^{O(k)}$ *processors.*

Corollary 2.7. *Using a polynomial number of processors, the* k*-consistency test can be tested in time* $O(|V(\mathsf{A})|^{k-1} \cdot |V(\mathsf{B})|^{k-1})$.

2.3. A Game Characterization of Local Consistency

We have already seen the definition of k-consistency as it appears in most textbooks.[7] Furthermore, we have presented an equivalent derivation system (Lemma 2.1) that captures the structure of propagation algorithms. In this paragraph we introduce a third view on the k-consistency heuristic in terms of a combinatorial pebble game.

The *existential* k*-pebble game* was first introduced by Kolaitis and Vardi [KV95] to study the power of Datalog. It is played by two players *Spoiler* and *Duplicator* on two relational structures A and B. There are k pairs of pebbles $(p_1, q_1), \ldots, (p_k, q_k)$ and during the game Spoiler moves the pebbles $p_1, \ldots, p_k$ to elements of $V(\mathsf{A})$ and Duplicator moves the pebbles $q_1, \ldots, q_k$ to elements of $V(\mathsf{B})$. At the beginning of the game, Spoiler places pebbles $p_1, \ldots, p_k$ on elements of $V(\mathsf{A})$ and Duplicator answers by putting pebbles $q_1, \ldots, q_k$ on elements of $V(\mathsf{B})$. In each further round Spoiler picks some pebbles and places one of them, say p_i, on another element in $V(\mathsf{A})$. Duplicator answers by picking up the same pebbles and moving the corresponding pebble q_i to one element in $V(\mathsf{B})$. Spoiler wins the game if he can reach a position where the mapping defined by $p_i \mapsto q_i$ is not a partial homomorphism from A to B.

The connection between the existential k-pebble game and the k-consistency heuristic was established by Kolaitis and Vardi [KV00]. They showed that a winning strategy for Duplicator (as formally defined in Definition 3.3) corresponds to a consistent domain $\mathcal{D}^{k-1}$. Going a different way, we now show that there is also a tight correspondence between Spoiler's strategy and CSP-refutations.

[7] On page 27.

Lemma 2.8. *Let* A *and* B *be two relational structures. There is a CSP-refutation for* A *and* B *of width* $k-1$ *and depth* d *if and only if Spoiler has a strategy to win the existential* k*-pebble game on* A *and* B *within* d *rounds.*

Proof. For one direction assume that there is a CSP-refutation P of depth d and width $k-1$. We show by induction over the depth that every partial mapping p of depth i occurring in the refutation defines a position of pebbles from which Spoiler can win the existential k-pebble game within i rounds. It follows that Spoiler can win the game from $\emptyset$ (all pebbles off the board) within d rounds. All mappings of depth $i = 0$ are axioms and thus not partial homomorphisms. Hence, Spoiler wins immediately. For the induction step assume that p has depth $i > 0$. Therefore, $|p| < k$ and p is derived from $p'_1 \cup \{x \mapsto a_1\}, \ldots, p'_n \cup \{x \mapsto a_n\}$ ($p'_j \subseteq p$) each of depth $< i$. Spoiler can now reach one of these positions within one round by placing the remaining pebble on x. Depending on Duplicator's choice (some $a_j \in V(\mathsf{B})$) Spoiler moves to $p'_j \cup \{x \mapsto a_j\}$ by picking up the pebbles in $p \setminus p'_j$. By induction hypothesis, Spoiler can win from $p'_j \cup \{x \mapsto a_j\}$ within $< i$ rounds and hence he can win from p within i rounds.

To prove the other direction we show by induction over the number of rounds that if Spoiler has an i-round winning strategy from a position p, then some $p' \subseteq p$ has a CSP-derivation of depth i. Since we assume that Spoiler has a d-round winning strategy from $\emptyset$, the lemma follows. For $i = 0$ the 0-round winning positions are precisely the axioms in our derivation system. Assume that Spoiler has an i-round winning strategy from p. In the next round in his strategy Spoiler first picks up pebble pairs (at least one if $|p| = k$). Let $p' \subseteq p$ be the new position and note that $|p'| < k$. By the definition of the game Spoiler also has an i-round winning strategy from p'. Let $x \in V(\mathsf{A})$ be the element on which the next pebble is set. Since Spoiler has a strategy to win against every possible choice of Duplicator, we know that $p' \cup \{x \mapsto a_1\}, \ldots, p' \cup \{x \mapsto a_n\}$ are positions from which Spoiler can win the game within $i-1$ rounds. For all these positions there is a $p_j \subseteq p' \cup \{x \mapsto a_j\}$ that has a derivation of depth at most $i-1$ by induction hypothesis. If for some j it holds that $p_j \subseteq p' \subseteq p$ we

are done. Otherwise, all p_j are of the form $p_j = p'_j \cup \{x \mapsto a_j\}$ with $p'_j \subseteq p' \subseteq p$. Thus, p has a derivation of depth at most i by applying the derivation rule (2.1). □

2.4. Lower Bounds for Local Consistency

Our main results in the area of Constraint Satisfaction (Part I) are lower bounds for propagation algorithms (Theorem 1) and for the computational complexity of k-Cons (Theorem 2). We start discussing the former one.

Lower Bounds for Propagation Algorithms

Our aim is to prove lower bounds for sequential and parallel propagation algorithms for k-consistency. As described in the previous section we do so by proving lower bounds on the propagation amount $\mathrm{prop}^k(\mathsf{A}, \mathsf{B})$ and the propagation depth $\mathrm{depth}^k(\mathsf{A}, \mathsf{B})$. Recall that in general

$$\mathrm{depth}^k(\mathsf{A}, \mathsf{B}) \leq \mathrm{prop}^k(\mathsf{A}, \mathsf{B}) \leq |V(\mathsf{A})|^{k-1}|V(\mathsf{B})|^{k-1}.$$

The next theorem shows that the upper bound is tight. The proof of this theorem will take the whole Chapter 3.

Theorem 1. *For every integer $k \geq 2$ there exists a constant $\varepsilon > 0$ and two positive integers n_0, m_0 such that for every $n \geq n_0$ and $m \geq m_0$ there exist two binary structures A_n and B_m with $|V(\mathsf{A}_n)| = n$ and $|V(\mathsf{B}_m)| = m$ such that $\mathrm{depth}^k(\mathsf{A}_n, \mathsf{B}_m) \geq \varepsilon n^{k-1} m^{k-1}$.*

Corollary 2.9. *There is a sequence of binary structures $\mathsf{A}_i, \mathsf{B}_i$ with $|V(\mathsf{A})| = |V(\mathsf{B})| = n$ such that every (sequential or parallel) propagation algorithm for k-consistency needs at least $\Omega(n^{2(k-1)})$ sequential propagation steps.*

We are aware of two particular cases that have been discovered earlier. First, the case $k = 2$ can be shown by rather simple examples that occurred very early in the AI-community. We will discuss this

case in Section 3.1. Second, for $k = 3$ Ladkin and Maddux [LM94] showed that there is a fixed finite binary structure B and an infinite sequence of binary structures A_i such that $\text{depth}^3(\mathsf{A}_i, \mathsf{B}) = \Omega(|V(\mathsf{A}_i)|^2)$. They used this result to argue that every parallel propagation algorithm for path consistency needs at least a quadratic number of steps. This is tight only for fixed structures B, Theorem 1 extends their result to the case when B is also given as input.

Computational Complexity of Local Consistency

Our second result for k-consistency addresses the decision complexity of the k-consistency test. Independent of the fact that the propagation approach is the intended, and up to now the only, way to implement a k-consistency test it could well be that there is an algorithm of any kind whatsoever that decides k-consistency faster. To address this question we are interested in the computational complexity of the decision problem k-Cons. Our main result is the following.

Theorem 2. *For every fixed $k \geq 15$ and any $\varepsilon > 0$, the winner of the existential k-pebble game on two given binary structures A and B cannot be determined in time $O((\|\mathsf{A}\| + \|\mathsf{B}\|)^{\frac{k-2}{12}-\varepsilon})$ on deterministic multi-tape Turing machines.*

As a consequence, we get an $\Omega((\|\mathsf{A}\| + \|\mathsf{B}\|)^{\frac{k-2}{12}-\varepsilon})$ lower bound on the running time for k-consistency tests on multi-tape Turing machines by Lemma 2.1 and Lemma 2.8.

Corollary 2.10. *It cannot be decided in time $n^{o(k)}$ whether k-consistency can be established on CSP-instances of size n. This holds for any computation model that can be polynomially simulated by Turing Machines.*

Chapter 4 is devoted to the proof of Theorem 2. It relies on many concepts and gadgets already introduced in Chapter 3 for the proof of Theorem 1. We prove Theorem 2 by a reduction from the k-pebble game of Kasai, Adachi and Iwata [KAI79], called KAI-game, to the existential $(\mathsf{k}+1)$-pebble game. Our result then follows from an $n^{\Omega(\mathsf{k})}$

lower bound for this game [AIK84], which in turn follows from the deterministic time hierarchy theorem.

For $k = 2$, Kasif [Kas90] showed that k-Cons is complete for PTIME under LOGSPACE reductions. As a consequence it follows that arc consistency is not efficiently parallelizable unless PTIME = NC. Kolaitis and Panttaja [KP03] extended this result to every fixed $k \geq 2$. Moreover, they established that the problem is complete for EXPTIME if k is part of the input.

Theorem 2.11 ([KP03])**.** *It is* EXPTIME*-complete to decide whether k-consistency can be established, given two structures* A, B *and an integer k.*

This remarkable result, and the techniques used to prove it, are the main inspiration for the proof of Theorem 2. To prove EXPTIME-completeness Kolaitis and Panttaja reduced the KAI-game to the existential pebble game. In their reduction the number of pebbles used in the existential pebble game depends on the size of the KAI-game instance and is not bounded by any function of the number of pebbles used in the KAI-game. Thus, their reduction fails to prove a lower bound for fixed k and it was left as an open question if such a lower bound can be proven. In the proof of Theorem 2 we reduce the k-pebble KAI-game to the existential $(\mathsf{k}+1)$-pebble game, and thus keep the parameter small. Some constructions in the proof can be seen as further development to those given by Kolaitis and Panttaja.[8]

Parameterized by the number of pebbles k, k-Cons is known to be hard for the parameterized complexity class W[1]. This follows directly from the fact that a graph G contains a k-clique if and only if Duplicator has a winning strategy for the existential k-pebble game on the complete graph on k vertices and G. Thus, the existence of an algorithm of running time $f(k)n^c$ for any computable function f and constant c would imply W[1] = FPT, an unlikely event in parameterized complexity theory. Our proof of Theorem 2 implies that k-Cons is indeed complete for the complexity class XP and hence, since XP $\neq$ FPT, it shows that k-Cons $\notin$ FPT without relying on unproven assumptions.

[8] Most notably the *switch*, see Remark 3.15 on page 62.

Corollary 2.12. *Given two structures* A, B *and a parameter* k. *Deciding whether* k*-consistency can be established on* A *and* B *is* XP*-complete under fpt-reductions.*

Finally, the parameterized complexity of k-consistency has also been investigated by Gaspers and Szeider [GS11], who considered a similar problem based on the following observation. They argued that the task of checking whether the instance is already k-consistent is inherent in every k-consistency test and thus lower bounds its complexity. This motivates them to analyze the following parameterized problem (translated to our notation): "Given two structures and a parameter k, are the structures k-consistent?" They showed that this problem is complete for the parameterized complexity class co-W[2]. Hence, assuming FPT $\neq$ co-W[2], the problem is not solvable in time $O(f(k)n^c)$ for some computable function f and constant c. Note that the problem in Corollary 2.12 concerns the stronger statement "Given two structures and a parameter k, can k-consistency be established?" The outcome of this decision problem matches the outcome of a k-consistency algorithm and thus characterizes the complexity of the k-consistency test precisely.

3. Lower Bounds for Propagation Algorithms

3.1. Warm up: Lower Bounds for Arc Consistency

As a simple base case we start with a lower bound on the propagation depth for arc consistency. For this we define two structures A_n and B_m on n and m vertices. The structure A_n is an edge-colored directed cycle of length n. One edge gets the color *thick* all other edges get the color *thin*. The structure B_m is a directed path of thick edges where at each vertex a thin loop is attached. See Figure 3.1 for an example this pair of structures. Similar examples of this kind occurred very early in the literature [DP85] and were also used for benchmarking arc consistency implementations [Bes+05]. The next lemma states that Spoiler can win the existential 2-pebble game but needs at least $\Omega(nm)$ rounds. The $k = 2$ case of Theorem 1 follows immediately.

Lemma 3.1. *Spoiler has a winning strategy in the existential 2-pebble game on* A_n *and* B_m *and the maximal number of rounds in any strategy is at least* $\left\lfloor \frac{1}{2}nm \right\rfloor$.

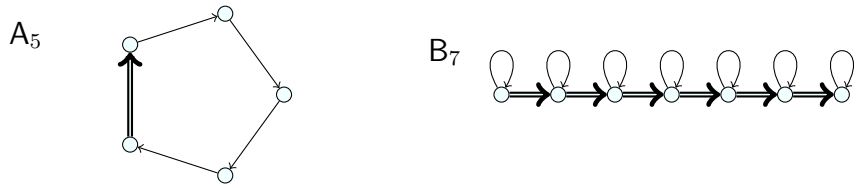


Figure 3.1.: The structures A_5 and B_7.

Proof. Spoiler can win the existential 2-pebble game with the following strategy. At the beginning Spoiler puts his pebbles on the two endpoints of the thick edge in A_n. Duplicator has to answer by pebbling one of the thick edges in B_m. Depending on whether the pebbles in B_m are closer to the start or to the end of the path, Spoiler starts moving his pebbles counter-clockwise or clockwise along the edges of the cycle. Assume the latter, i. e. Spoiler picks up the pebbles alternately and always pebbles the outgoing neighbor of the current pebble on the board. Duplicator has to pebble the same kind of outgoing neighbors in B_m. Thus, whenever Spoiler pebbles a thin edge Duplicator stays with both pebbles on the same vertex and whenever Spoiler pebbles a thick edge Duplicator moves one step further on the path. Finally, Duplicator has to pebble the last vertex on the path and when Spoiler pebbles the thick edge at this position, Duplicator loses.

The lower bound follows from the observation that this is also the fastest way for Spoiler to win. Duplicator's counter-strategy is to start in the middle of the path and to answer as described above whenever Spoiler pebbles along an edges. If Spoiler pebbles a vertex that is non-adjacent to the currently pebbled vertex, then Duplicator moves the corresponding pebble to one vertex in the middle of the path. Following that strategy Duplicator can force Spoiler to perform at least $\left\lfloor \frac{1}{2}nm \right\rfloor$ steps to win the game. □

Remark 3.2. An important feature in this construction is that the edges are colored and the second structure contains loops. Berkholz and Verbitsky [BV13] showed that this can be avoided: a similar lower bound holds for uncolored acyclic directed graphs.

Before we are able to prove Theorem 1 for $k > 2$, we need some definitions and tools for playing the existential pebble game. They will also be useful in Chapter 4 and Chapter 6.

3.2. Toolbox for the Existential Pebble Game

To argue about winning strategies on large structures we need to combine strategies on smaller parts of the structures to a strategy for the whole structure. The easier part is combine Spoiler's strategies. As in [Gro99] and [KP03], we say that Spoiler *can reach* position p from position q if he has a strategy in the game such that starting from position q he wins the game, or position p occurs in the game after some finite number of rounds. Since this relation is transitive, we can combine such strategies to show that Spoiler can reach some position p from $\emptyset$; if p does not define a partial homomorphism, this gives us a winning strategy for Spoiler. We also extend this notion to sets of positions $\mathcal{P}$ and say that Spoiler can reach a position in $\mathcal{P}$ from q if he has a strategy such that starting from position q he wins the game, or some position $p \in \mathcal{P}$ occurs after finitely many rounds.

Arguing about Duplicator's strategies is more difficult. We start with the standard definition of a winning strategy for Duplicator. By $\mathrm{Dom}(h)$ we denote the domain of a (partial) function h.

Definition 3.3 ([KV95])**.** A *winning strategy* for Duplicator in the existential k-pebble game on structures A and B is a nonempty family $\mathcal{H}$ of partial homomorphisms from A to B satisfying the following properties:

closure If $h \in \mathcal{H}$ and $g \subset h$ then $g \in \mathcal{H}$.

extension For every $g \in \mathcal{H}$, $|g| < k$, and every $x \in V(\mathsf{A})$ there is an $h \in \mathcal{H}$ with $g \subseteq h$ and $x \in \mathrm{Dom}(h)$.

The set $\mathcal{H}$ is the set of winning positions for Duplicator, that's why they are all partial homomorphisms. Non-emptiness and the closure property ensure that $\mathcal{H}$ contains the start position $\emptyset$. Furthermore, the closure property guarantees that the current position remains a winning position for Duplicator when Spoiler picks up pebbles. The extension property ensures that Duplicator has an appropriate answer if Spoiler puts a free pebble on x. It is easy to see that if there is a total homomorphism h from A to B, then the powerset $\wp(h)$ is a winning strategy in the existential k-pebble game on A and B.

Next we introduce *critical strategies* that are nearly winning strategies. The difference is that Duplicator does not always have a corresponding answer, and this happens precisely in a situation when the current position belongs to a set of predefined *critical positions*.

Definition 3.4. A *critical strategy* for Duplicator in the existential k-pebble game on structures A and B is a nonempty family $\mathcal{H}$ of partial homomorphisms from A to B together with a set $\mathrm{crit}(\mathcal{H}) \subseteq \mathcal{H}$ of *critical positions* satisfying the following properties:

- All critical positions are $(k-1)$-partial homomorphisms.
- If $h \in \mathcal{H}$ and $g \subset h$, then $g \in \mathcal{H}$.
- For every $g \in \mathcal{H} \setminus \mathrm{crit}(\mathcal{H})$, $|g| < k$, and every $x \in V(\mathsf{A})$ there is an $h \in \mathcal{H}$ with $g \subseteq h$ and $x \in \mathrm{Dom}(h)$.

A critical strategy is nearly a winning strategy in the sense that Duplicator wins unless the game reaches a critical position. Note that a critical strategy with $\mathrm{crit}(\mathcal{H}) = \emptyset$ is a winning strategy and every critical strategy in the $(k+1)$-pebble game is a winning strategy in the k-pebble game. Let $\widehat{\mathcal{H}} := \mathcal{H} \setminus \mathrm{crit}(\mathcal{H})$. As for winning strategies, the union of critical strategies is also a critical strategy. The following lemma enables us to construct a winning strategy out of critical strategies. The proof follows directly from the definitions.

Lemma 3.5. *If $\mathcal{H}_1, \ldots, \mathcal{H}_l$ are critical strategies on the same structures and for all $i \in [l]$ and all $p \in \mathrm{crit}(\mathcal{H}_i)$ there exists a $j \in [l]$ such that $p \in \widehat{\mathcal{H}}_j$, then $\bigcup_{i \in [l]} \mathcal{H}_i$ is a winning strategy on these structures.*

If Spoiler has a winning strategy in the existential k-pebble game, we will use critical strategies to prove lower bounds on the number of rounds Spoiler needs to win.

Lemma 3.6. *If $\mathcal{H}_1, \ldots, \mathcal{H}_l$ is a sequence of critical strategies on the same structures and for all $i < l$ and all $p \in \mathrm{crit}(\mathcal{H}_i)$ it holds that $p \in \widehat{\mathcal{H}}_j$ for some $j \leq i+1$, then Duplicator wins the l-round existential k-pebble game.*

Proof. Starting with $i = 1$, Duplicator answers according to the extension property of $\mathcal{H}_i$, if the current position p is non-critical in $\mathcal{H}_i$. Otherwise, p is non-critical in $\mathcal{H}_j$ for some $j \leq i + 1$ and Duplicator answers according to the extension property of $\mathcal{H}_j$. This allows Duplicator to survive for at least l rounds. □

Gadgets

We construct the two structures A and B out of smaller structures, called *gadgets*. Since the structures we construct are vertex colored graphs, we focus on this kind of structures from now on. In Part II we present a similar notion of gadgets for 3-CNF formulas.

Every gadget Q consists of two graphs Q_S and Q_D for Spoiler's and Duplicator's side, respectively. This means that Q_S will be subgraph of A and Q_D will be subgraph of B in the end. The gadgets contain *boundary vertices* $\mathrm{bd}(Q_S) \subseteq V(Q_S)$ and $\mathrm{bd}(Q_D) \subseteq V(Q_D)$, which are the vertices shared with other gadgets. That is, vertices in $V(Q_S) \setminus \mathrm{bd}(Q_S)$ ($V(Q_D) \setminus \mathrm{bd}(Q_D)$) are only adjacent to vertices in $V(Q_S)$ ($V(Q_D)$) in the final graph A (B). A *boundary function* of a strategy $\mathcal{H}$ on a gadget Q is a mapping $\beta\colon \mathrm{bd}(Q_S) \to \mathrm{bd}(Q_D)$ such that $\beta(z) = h(z)$ for all $h \in \mathcal{H}$ and all $z \in \mathrm{bd}(Q_S) \cap \mathrm{Dom}(h)$. We say that two strategies $\mathcal{G}$ and $\mathcal{H}$ on gadgets Q and Q' are *connectable*, if they have boundary functions $\beta_\mathcal{G}$ and $\beta_\mathcal{H}$ and it holds that $\beta_\mathcal{G}(z) = \beta_\mathcal{H}(z)$ for all $z \in \mathrm{bd}(Q_S) \cap \mathrm{bd}(Q'_S)$. If $\mathcal{G}$ and $\mathcal{H}$ are two connectable strategies, we define the *composition*

$$\mathcal{G} \uplus \mathcal{H} = \{g \cup h \mid g \in \mathcal{G}, h \in \mathcal{H}\}.$$

Lemma 3.7. *Let $\mathcal{G}$ and $\mathcal{H}$ be two connectable critical strategies on gadgets $Q = (Q_S, Q_D)$ and $Q' = (Q'_S, Q'_D)$, respectively. The composition $\mathcal{G} \uplus \mathcal{H}$ is a critical strategy on $Q_S \cup Q'_S$ and $Q_D \cup Q'_D$ with* $\mathrm{crit}(\mathcal{G} \uplus \mathcal{H}) = \mathrm{crit}(\mathcal{G}) \cup \mathrm{crit}(\mathcal{H})$. □

Playing according to the strategy $\mathcal{G} \uplus \mathcal{H}$ on Q and Q' means that Duplicator uses strategy $\mathcal{G}$ on Q and strategy $\mathcal{H}$ on Q'. The requirements on the boundary ensure that strategy $\mathcal{G}$ equals strategy $\mathcal{H}$ on the intersection of Q and Q'. We use the operator $\uplus$ to construct

global critical strategies for the whole graph out of critical strategies on the gadgets.

3.3. Overview of the Construction

In this section we prove Theorem 1 for $k \geq 3$. In order to ease notation and to be consistent with previous work [Ber12a; Ber13], we let

$$\mathsf{k} := k - 1 \geq 2.$$

To prove the theorem we construct two vertex colored graphs A_n and B_m with $O(n)$ and $O(m)$ vertices such that Spoiler needs $\Omega(n^{\mathsf{k}} m^{\mathsf{k}})$ rounds to win the existential $(\mathsf{k}+1)$-pebble game. We color the vertices of both graphs such that the colors partition the vertex set into independent sets, i. e. every vertex gets one color and there is no edge between vertices of the same color. The basic building blocks in our construction are sets of vertices which allow us to store $n^{\mathsf{k}} m^{\mathsf{k}}$ partial homomorphisms with k pebbles.

Figure 3.2.: Vertex blocks to encode $n^{\mathsf{k}} m^{\mathsf{k}}$ partial homomorphisms. Two vertices x_j^i and $x_{j'}^{i'}$ get the same color iff $i = i'$.

We introduce vertices x_j^i ($i \in [\mathsf{k}]$, $j \in [n]$) in A_n and vertices x_j^i ($i \in [\mathsf{k}]$, $j \in [m] \cup \{0\}$) in B_m. For every $i \in [\mathsf{k}]$ the vertices x_j^i form a *block* and are colored with the same color (say P_{x^i}), which is different from any other color in the entire construction. The vertices x_0^i in structure B_m play a special role in our construction and are visualized by $\circ$ instead of $\bullet$ in the pictures. However, they are colored with the same color P_{x^i} as the other vertices x_j^i.

Whenever Spoiler pebbles a vertex x_j^i, Duplicator has to answer with some $x_{j'}^i$ because of the vertex colors. Lets ignore the $\circ$ vertices x_0^i

for a moment. Since there are nm positions for one pebble pair in one block, we get $n^{\mathsf{k}}m^{\mathsf{k}}$ positions if every block has exactly one pebble pair on $\bullet$ vertices. The $\circ$ vertices are used by Duplicator whenever Spoiler does not play the intended way. That is, if Spoiler pebbles a vertex in block i that he is not supposed to pebble now, then Duplicator answers with x_0^i. The construction will have the property that this is always a good situation for Duplicator.

Formally, we define mappings $\mathfrak{a}\colon [\mathsf{k}] \to [n]$ and $\mathfrak{b}\colon [\mathsf{k}] \to [m]$. The expression "$(\mathfrak{a}, \mathfrak{b})$ on x" refers to the pebble position $\{(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$.[1] Whenever $(\mathfrak{a}, \mathfrak{b})$ is on x, Duplicator answers according to the mapping

$$h(x^i_j) = \begin{cases} x^i_{\mathfrak{b}(i)}, \text{ if } j = \mathfrak{a}(i), \\ x^i_0, \text{ otherwise.} \end{cases}$$

That is, whenever Spoiler pebbles a vertex x^i_j that he is not supposed to pebble now (i. e. $j \neq \mathfrak{a}(i)$), then Duplicator pebbles the $\circ$ vertex of the same block. Since the blocks form independent sets in both graphs, such positions are partial homomorphisms.

We also need to name positions where Duplicator answers with x^i_0 for every vertex in block i. We let T be the set of blocks where this happens. For $\mathfrak{a}\colon [\mathsf{k}] \to [n]$, $\mathfrak{b}\colon [\mathsf{k}] \to [m]$ and $T \subseteq [\mathsf{k}]$ we call $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ a *configuration*. The configuration $\mathfrak{q}$ is *valid* if $T = \emptyset$ and *invalid* otherwise. For every configuration $\mathfrak{q}$ and a set of x^i_j vertices as in Figure 3.2 we define the homomorphism

$$h^x_{\mathfrak{q}}(x^i_j) = \begin{cases} x^i_{\mathfrak{b}(i)}, \text{ if } j = \mathfrak{a}(i) \text{ and } i \notin T, \\ x^i_0, \text{ otherwise.} \end{cases}$$

By $h^x_{\mathbf{0}}$ we denote the homomorphism $h^x_{\mathbf{0}}(x^i_j) := x^i_0$ for all $i \in [\mathsf{k}], j \in [n]$, which is equal to $h^x_{\mathfrak{q}}$ for every invalid configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, [\mathsf{k}])$. We say that a position of (at most $\mathsf{k}+1$) pebble pairs on these ver-

[1] At this point we want to stress that the vertex "x^i_j" may occur in both structures. However, it will become clear from the context which vertex is meant and with a slight abuse of notation we alway assume that $x^i_{\mathfrak{a}(i)} \in V(\mathsf{A}_n)$ and $x^i_{\mathfrak{b}(i)} \in V(\mathsf{B}_n)$.

tices is *invalid* if it is a subset of $h^x_{\mathfrak{q}}$ for some invalid configuration $\mathfrak{q}$. Furthermore, a position of k pebble pairs is *valid* if it has the form $\{(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$ for a valid configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$. As above, we call this valid position "$\mathfrak{q}$ on x." Note that valid positions are not invalid.[2]

In the entire construction there is one unique copy of the x^i_j-vertices in Figure 3.2, which we denote by x. Our goal is to force Spoiler to pebble every valid position on x before he wins the game. He is supposed to do so in a specific predefined order. To fix this order we define a bijection α between valid configurations $(\mathfrak{a}, \mathfrak{b}, \emptyset)$ and the numbers $0, \ldots, n^{\mathsf{k}}m^{\mathsf{k}} - 1$.

$$\alpha(\mathfrak{q}) := m^{\mathsf{k}} \sum_{i=1}^{\mathsf{k}} (\mathfrak{a}(i) - 1) n^{\mathsf{k}-i} + \sum_{i=1}^{\mathsf{k}} (\mathfrak{b}(i) - 1) m^{\mathsf{k}-i}.$$

Thus, $\alpha(\mathfrak{q})$ is the rank of the tuple $(\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \mathfrak{b}(1), \ldots, \mathfrak{b}(\mathsf{k}))$ in the lexicographical order. If $\alpha(\mathfrak{q}) < n^{\mathsf{k}}m^{\mathsf{k}} - 1$, we define the successor $\mathfrak{q}^+ = (\mathfrak{a}^+, \mathfrak{b}^+, \emptyset)$ to be the unique valid configuration satisfying $\alpha(\mathfrak{q}^+) = \alpha(\mathfrak{q}) + 1$. In the sequel we introduce gadgets to make sure that:

- Spoiler can reach the position $\alpha^{-1}(0)$ on x from $\emptyset$,
- Spoiler can reach $\alpha^{-1}(i+1)$ on x from $\alpha^{-1}(i)$ on x and
- Spoiler wins from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}} - 1)$ on x.

If we have these properties, we know that Spoiler has a winning strategy in the $(\mathsf{k}+1)$-pebble game. To show that Spoiler needs at least $n^{\mathsf{k}}m^{\mathsf{k}}$ rounds we argue that this is essentially the only way for Spoiler to win the game.

We start with an overview of the gadgets and how they are glued together to form the structures A_n and B_m. Recall that there is a set of "x-vertices" (as described above and depicted in Figure 3.2) which gets a central position in our construction. The boundary[3] of

[2] There are pebble positions on the x^i_j vertices that are neither valid nor invalid. However, such positions will not occur in our strategies.

[3] See the definition on page 45.

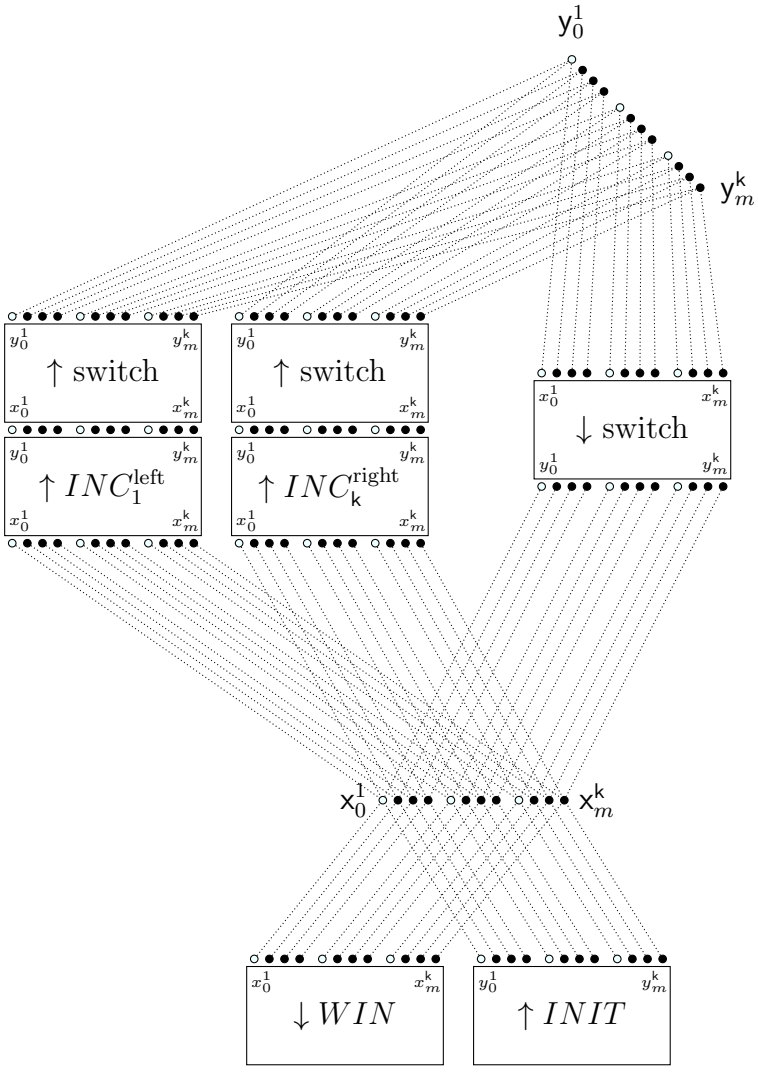


Figure 3.3.: The graph B_m. The boundaries of the gadgets are connected as indicated by the dotted lines (which need to be contracted). The arrows point from the input to the output vertices of the gadgets.

our gadgets consists of *input* vertices and *output* vertices. For every gadget the set of input (output) vertices is a copy of the vertex set in Figure 3.2 and we write x_j^i (y_j^i) to name them. This enables us to glue together the gadgets at their input and output vertices. The overall construction for the graph B_m is shown in Figure 3.3. The schema for A_n is similar, it contains the Spoiler's side of the corresponding gadgets which are glued together the same way as in B_m (just replace m by n and drop the $\circ$ vertices). There are four types of gadgets: the initialization gadget, the winning gadget, several increment gadgets and the switch.

The *initialization gadget* ensures that Spoiler can reach $\alpha^{-1}(0)$ on x, i. e. the pebble position $\{(\mathsf{x}_1^1, \mathsf{x}_1^1), \ldots, (\mathsf{x}_1^k, \mathsf{x}_1^k)\}$. This gadget has only output boundary vertices and is used by Spoiler at the beginning of the game. There are *increment gadgets* $\mathrm{INC}_i^{\mathrm{left}}$ and $\mathrm{INC}_i^{\mathrm{right}}$ for all $i \in [\mathsf{k}]$. Their boundary consists of input vertices x_j^i and output vertices y_j^i. The input vertices of every increment gadget are identified with the x vertices as depicted in Figure 3.3. The increment gadgets (all together) ensure that Spoiler can reach $\alpha^{-1}(j+1)$ from $\alpha^{-1}(j)$ on x. More precisely, for every valid position $\mathfrak{q}$ on x, $\alpha(\mathfrak{q}) < n^{\mathsf{k}} m^{\mathsf{k}} - 1$, there is one increment gadget INC (out of $2k$) such that Spoiler can reach $\mathfrak{q}^+$ on the output of INC from $\mathfrak{q}$ on the input. Every increment gadget is followed by a copy of the *switch*. The input of $2\mathsf{k}$ switches is identified with the output of the $2\mathsf{k}$ increment gadgets and the output of these switches is identified with the input of one additional switch (see Figure 3.3). The output of this switch is in turn identified with the unique block of x-vertices. The switches are used to perform the transition in the game from $\alpha^{-1}(j)$ on x to $\alpha^{-1}(j+1)$ on x. Spoiler can pebble a valid position through one switch: from $\mathfrak{q}$ on the input of a switch Spoiler can reach $\mathfrak{q}$ on the output of that switch. Hence, Spoiler can simply pebble the incremented position $\alpha^{-1}(j+1)$ from the output of an increment gadget through two switches to the x-block. The switches are the most important tool for Duplicator. This is because Spoiler can *only* pebble a position through the switch if he picks up all $\mathsf{k}+1$ pebbles pairs and plays with them on the switch. In such a situation Duplicator switches his global strategy from "I play according to $\alpha^{-1}(j)$ on x" to "I play according to $\alpha^{-1}(j+1)$ on

x." This change of the strategy on the x-vertices is possible since all pebbles are locked inside one switch.

Finally, the *winning gadget* ensures that from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on x Spoiler wins the game. The winning gadget has only input vertices, which are identified with the x-vertices. From $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on the input, Spoiler can win the game by playing on this gadget. On the other hand, Duplicator does not lose from any other (valid or invalid) configuration on x by playing on this gadget.

3.4. The Gadgets

We now describe the gadgets in detail and provide strategies for Spoiler and Duplicator on them. In the next section we combine these partial strategies to prove Theorem 1.

The Winning Gadget

Lets start with the simplest among all gadgets: the winning gadget. It contains input vertices x^i_j, $i \in [\mathsf{k}]$, $j \in [n]$ in structure A_n and x^i_j, $i \in [\mathsf{k}]$, $j \in [m] \cup \{0\}$ in B_m. The whole gadget with Spoiler's side WIN_S and Duplicator's side WIN_D is shown in Figure 3.4. On Spoiler's side there is just one additional vertex a, which is connected to x^i_n for all $i \in [\mathsf{k}]$. On Duplicator's side there are k additional vertices $a^i, i \in [\mathsf{k}]$. Every a^i is connected to all input vertices except x^i_m. We use one new vertex color to color the vertex a and all vertices a_i. From the position $\{(x^1_n, x^1_m), \ldots, (x^k_n, x^k_m)\}$ ("$\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ on x") Spoiler wins the game by placing the $(\mathsf{k}+1)$st pebble on a. Duplicator has to answer with some a_i (because of the coloring). Since there is an edge between x^i_n and a in WIN_S but none between x^i_m and a_i in WIN_D, Spoiler wins immediately. This proves the following Lemma.

Lemma 3.8. *Spoiler wins the existential* $(\mathsf{k}+1)$*-pebble game on* WIN *from* $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}}-1)$ *on the input.*

It is also not hard to see that for any other position where at least one pebble pair (x^j_n, x^j_m) is missing Duplicator can survive by choosing

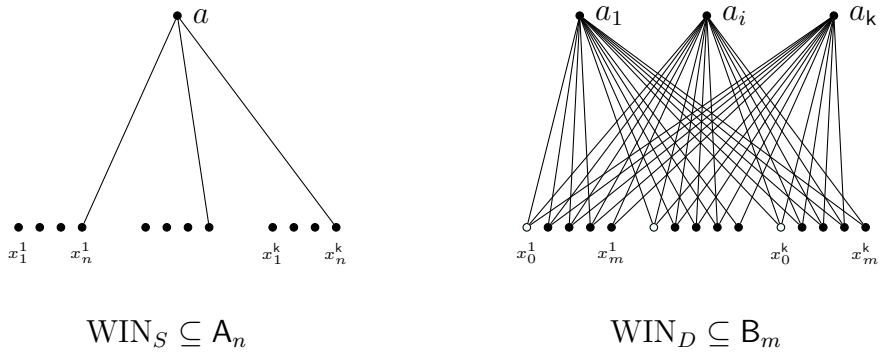


Figure 3.4.: The winning gadget.

a_j. Let $\mathfrak{q}_{\text{win}} = \alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}} - 1)$ be the configuration $(\mathfrak{a}_{\text{win}}, \mathfrak{b}_{\text{win}}, \emptyset)$ with $\mathfrak{a}_{\text{win}}(i) := n$ and $\mathfrak{b}_{\text{win}}(i) := m$ for all $i \in [\mathsf{k}]$. The next lemma formalizes Duplicator's strategy.

Lemma 3.9. *For every configuration $\mathfrak{q} \neq \mathfrak{q}_{win}$ there is a winning strategy for Duplicator in the existential $(\mathsf{k}+1)$-pebble game on* WIN *with boundary function $h^x_{\mathfrak{q}}$.*[4]

Proof. Let $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$. Since $\mathfrak{q} \neq \mathfrak{q}_{\text{win}}$ there is an index $j \in [\mathsf{k}]$ such that $\mathfrak{a}(j) \neq n$ or $\mathfrak{b}(j) \neq m$ or $j \in T$. Hence, $h^x_{\mathfrak{q}}(x^j_n) \neq x^j_m$ by the definition of $h^x_{\mathfrak{q}}$. Let $h := h^x_{\mathfrak{q}} \cup \{(a, a_j)\}$. Note that h is a homomorphism from WIN_S to WIN_D since it preserves vertex colors and maps edges to edges. The winning strategy for Duplicator is $\wp(h)$ which has $h^x_{\mathfrak{q}}$ as boundary function by definition.[5] □

The Increment Gadgets

The increment gadgets were used to reach $\alpha^{-1}(i+1)$ from $\alpha^{-1}(i)$ on x. Recall that we identify every valid configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ with the tuple $(\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \mathfrak{b}(1), \ldots, \mathfrak{b}(\mathsf{k})) \in [n]^k \times [m]^k$ and define $\alpha(\mathfrak{q})$

[4] Recall the definition of $h^x_{\mathfrak{q}}$ on page 47.
[5] Recall that $\wp(h) := \{g \mid g \subseteq h\}$ denotes the power set.

to be the rank (from 0 to $n^k m^k - 1$) of this tuple in lexicographical order. Let $\mathfrak{q}$ be a valid configuration with $\alpha(\mathfrak{q}) < n^{\mathsf{k}} m^{\mathsf{k}} - 1$ and successor $\mathfrak{q}^+ = (\mathfrak{a}^+(1), \ldots, \mathfrak{a}^+(\mathsf{k}), \mathfrak{b}^+(1), \ldots, \mathfrak{b}^+(\mathsf{k}))$. We use two types of increment gadgets, *left* and *right*, depending on whether the left-hand side of the tuple changes after incrementation or not. If $\mathfrak{a}(i) = \mathfrak{a}^+(i)$ for all $i \in [\mathsf{k}]$, then a right increment gadget will be used by Spoiler to reach $\mathfrak{q}^+$ on the output from $\mathfrak{q}$ on the input. If otherwise $\mathfrak{a}(i) \neq \mathfrak{a}^+(i)$ for some $i \in [\mathsf{k}]$ (and hence $\mathfrak{b}(i) = m$ and $\mathfrak{b}^+(i) = 1$ for all $i \in [\mathsf{k}]$), then Spoiler uses a left increment gadget. There are k increment gadgets of each type. Spoiler uses them depending on the position where last change in the tuple occurs. If

$$\begin{aligned} \mathfrak{q} &= (\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \quad \mathfrak{b}(1), \ldots, \mathfrak{b}(\ell) < m, \quad m, \ldots, m) \text{ and hence} \\ \mathfrak{q}^+ &= (\mathfrak{a}(1), \ldots, \mathfrak{a}(\mathsf{k}), \quad \mathfrak{b}(1), \ldots, \mathfrak{b}(\ell) + 1, \quad 1, \ldots, 1), \end{aligned}$$

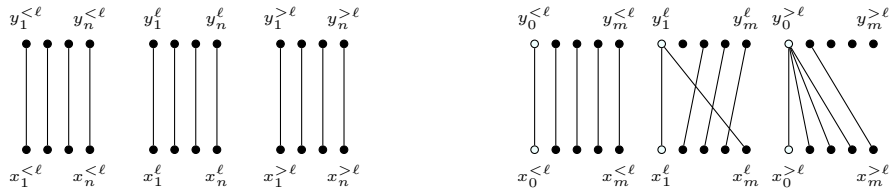
then Spoiler uses the increment gadget $\mathrm{INC}_\ell^{\mathrm{right}}$ to reach $\mathfrak{q}^+$ on the output from $\mathfrak{q}$ on the input. If

$$\begin{aligned} \mathfrak{q} &= (\mathfrak{a}(1), \ldots, \mathfrak{a}(\ell) < n, \quad n, \ldots, n, \quad m, \ldots, m) \text{ and hence} \\ \mathfrak{q}^+ &= (\mathfrak{a}(1), \ldots, \mathfrak{a}(\ell) + 1, \quad 1, \ldots, 1, \quad 1, \ldots, 1), \end{aligned}$$

then Spoiler uses $\mathrm{INC}_\ell^{\mathrm{left}}$. Thus, for every valid configuration $\mathfrak{q}$ with $\alpha(\mathfrak{q}) < n^{\mathsf{k}} m^{\mathsf{k}} - 1$ there is exactly one increment gadget Spoiler uses and in that case we say that this gadget is applicable to $\mathfrak{q}$. The next definition formalizes this.

Definition 3.10. The right increment gadget $\mathrm{INC}_\ell^{\mathrm{right}}$ is *applicable* to a configuration $\mathfrak{q}$ if $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ is valid, $\mathfrak{b}(\ell) < m$ and $\mathfrak{b}(i) = m$ for all $i > \ell$. A left increment gadget $\mathrm{INC}_\ell^{\mathrm{left}}$ is applicable to $\mathfrak{q}$ if $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ is valid, $\mathfrak{a}(\ell) < n$, $\mathfrak{a}(i) = n$ for all $i > \ell$ and $\mathfrak{b}(i) = m$ for all $i \in [\mathsf{k}]$.

It follows from the definition that for every valid configuration $\mathfrak{q}$ with $\alpha(\mathfrak{q}) < n^{\mathsf{k}} m^{\mathsf{k}} - 1$ there is exactly one increment gadget that is applicable to $\mathfrak{q}$. Furthermore if there is some applicable increment gadget for $\mathfrak{q}$, then $\alpha(\mathfrak{q}) < n^{\mathsf{k}} m^{\mathsf{k}} - 1$ and $\mathfrak{q}^+$ is defined. We define $T_\ell^{\mathrm{right}}(\mathfrak{q}) \subseteq [\mathsf{k}]$ and $T_\ell^{\mathrm{left}}(\mathfrak{q}) \subseteq [\mathsf{k}]$ to be the set of blocks that contradict



$\mathrm{INC}^{\mathrm{right}}_{\ell\ S} \subseteq \mathsf{A}_n$ $\mathrm{INC}^{\mathrm{right}}_{\ell\ D} \subseteq \mathsf{B}_m$

Figure 3.5.: The right increment gadget $\mathrm{INC}^{\mathrm{right}}_{\ell}$

the applicability of $\mathrm{INC}^{\mathrm{right}}_{\ell}$ ($\mathrm{INC}^{\mathrm{left}}_{\ell}$, resp.) to $\mathfrak{q}$. That is, $i \in T^{\mathrm{right}}_{\ell}(\mathfrak{q})$ for a configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ if one of the following conditions is satisfied:

- $i = \ell$ and $\mathfrak{b}(i) = m$,
- $i > \ell$ and $\mathfrak{b}(i) \neq m$,
- $i \in T$.

Similarly, $i \in T^{\mathrm{left}}_{\ell}(\mathfrak{q})$ if

- $i = \ell$ and $\mathfrak{a}(\ell) = n$,
- $i > \ell$ and $\mathfrak{a}(i) \neq n$,
- $\mathfrak{b}(i) \neq m$ or
- $i \in T$.

Therefore, $T^{\mathrm{right}}_{\ell}(\mathfrak{q}) = \emptyset$ ($T^{\mathrm{left}}_{\ell}(\mathfrak{q}) = \emptyset$) if and only if $\mathrm{INC}^{\mathrm{right}}_{\ell}$ ($\mathrm{INC}^{\mathrm{left}}_{\ell}$) is applicable to $\mathfrak{q}$. We now start describing the gadgets. Every increment gadget contains input vertices x^i_j, output vertices y^i_j and no further vertices. The right increment gadget $\mathrm{INC}^{\mathrm{right}}_{\ell}$ is shown in Figure 3.5. While we describe the gadget it might be useful to have the situation in mind when we want to use this gadget. Hence, assume that the current pebble position is

$$\{(x^1_{\mathfrak{a}(1)}, x^1_{\mathfrak{b}(1)}), \ldots, (x^{\ell}_{\mathfrak{a}(\ell)}, x^{\ell}_{\mathfrak{b}(\ell)}), (x^{\ell+1}_{\mathfrak{a}(\ell+1)}, x^{\ell+1}_{m}), \ldots, (x^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}, x^{\mathsf{k}}_{m})\}$$

and Spoiler wants to reach the incremented position

$$\{(y^1_{\mathfrak{a}(1)}, y^1_{\mathfrak{b}(1)}), \ldots, (y^\ell_{\mathfrak{a}(\ell)}, y^\ell_{\mathfrak{b}(\ell)+1}), (y^{\ell+1}_{\mathfrak{a}(\ell+1)}, y^{\ell+1}_1), \ldots, (y^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}, y^{\mathsf{k}}_1)\}.$$

In Spoiler's side of the gadget all blocks have the same shape: every x^i_j is connected to y^i_j. In Duplicator's side the first $\ell - 1$ blocks look the same, x^i_j is connected to y^i_j for all $i < \ell$ and $0 \leq j \leq m$. This ensures that Spoiler can reach $(y^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{b}(i)})$ from $(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)})$, for all $i < \ell$, by placing the remaining $(\mathsf{k}+1)$st pebble on $y^i_{\mathfrak{a}(i)}$. Block ℓ on Duplicator's side looks different (this is the block where we want to increment $\mathfrak{b}(\ell)$ to $\mathfrak{b}(\ell)+1$). The input vertex x^ℓ_i is connected to y^ℓ_{i+1} for every $i \in [m-1]$. Furthermore, we connect x^ℓ_m to the special vertex y^ℓ_0. From the position $(x^\ell_{\mathfrak{a}(\ell)}, x^\ell_{\mathfrak{b}(\ell)})$, where $\mathfrak{b}(\ell) < m$, Spoiler can reach $(y^\ell_{\mathfrak{a}(\ell)}, y^\ell_{\mathfrak{b}(\ell)+1})$ by placing the remaining pebble on $y^\ell_{\mathfrak{a}(\ell)}$. Every other block $i > \ell$ on Duplicator's side contains an edge between x^i_m and y^i_1 to ensure that Spoiler can reach $(y^i_{\mathfrak{a}(i)}, y^i_1)$ from $(x^i_{\mathfrak{a}(i)}, x^i_m)$ in the same way. Furthermore, the other input vertices x^i_j, $j < m$, are connected to the special vertex y^i_0.

Lemma 3.11. *Let $\ell \in [\mathsf{k}]$ and $\mathfrak{q}$ be a configuration such that $\mathrm{INC}^{\mathrm{right}}_\ell$ is applicable to $\mathfrak{q}$. Then Spoiler can reach $\mathfrak{q}^+$ on the output from $\mathfrak{q}$ on the input of the right increment gadget $\mathrm{INC}^{\mathrm{right}}_\ell$.*

Proof. Let $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ with successor $\mathfrak{q}^+ = (\mathfrak{a}^+, \mathfrak{b}^+, \emptyset)$. Note that $\mathfrak{a} = \mathfrak{a}^+$ by Definition 3.10. We have to show that Spoiler can reach $\{(y^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{b}^+(i)}) \mid i \in [\mathsf{k}]\}$ from $\{(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$. By definition of the gadget, $y^i_{\mathfrak{a}(i)}$ is a neighbor of $x^i_{\mathfrak{a}(i)}$ on Spoiler's side. On Duplicator's side, $y^i_{\mathfrak{b}^+(i)}$ is a neighbor of $x^i_{\mathfrak{b}(i)}$. Furthermore, $y^i_{\mathfrak{b}^+(i)}$ is the only neighbor among $y^i_0, \ldots, y^i_m$, which are all the vertices having the same color as $y^i_{\mathfrak{a}(i)}$ (see Figure 3.5). In such a situation Spoiler can reach $\mathfrak{q}^+$ at the output by the following procedure. First, Spoiler places the remaining pebble on $y^1_{\mathfrak{a}(1)}$. Since this vertex is adjacent to $x^1_{\mathfrak{a}(1)}$, Duplicator has to answer with a vertex of the same color (i. e. y^1_j for some $0 \leq j \leq m$) that is adjacent to $x^1_{\mathfrak{b}(1)}$. The only vertex satisfying this property is $y^1_{\mathfrak{b}^+(1)}$. Thus, the new pebble position is $(y^1_{\mathfrak{a}(1)}, y^1_{\mathfrak{b}^+(1)})$ and Spoiler can pick up the pebble pair from

$(x^1_{\mathfrak{a}(1)}, x^1_{\mathfrak{b}(1)})$. On the second block Spoiler proceeds the same way: he pebbles $y^2_{\mathfrak{a}(2)}$, forces the position $(y^2_{\mathfrak{a}(2)}, y^2_{\mathfrak{b}^+(2)})$ and picks up the pebbles from $(x^2_{\mathfrak{a}(2)}, x^2_{\mathfrak{b}(2)})$. By iterating this procedure, Spoiler can reach the position $\{(y^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{b}^+(i)}) \mid i \in [\mathsf{k}]\}$. □

We have shown that Spoiler can increment a configuration using an applicable right increment gadget. The next step is to show that this is essentially everything Spoiler can do with this gadget. To show this, we have to ensure the following two assertions. First, from a valid configuration on the input of an increment gadget applicable to it Spoiler can *only* reach the incremented configuration on the output. Second, if the gadget is not applicable to the configuration on the input, then Spoiler cannot reach a valid configuration on the output. We provide Duplicator with appropriate counter strategies to ensure these assertions. In general, Duplicator's counter strategy is to pebble a $\circ$ vertex whenever she has the possibility to do so. On the one hand, if Spoiler pebbles a vertex that is neither equal nor adjacent to another pebbled vertex in A_n, then Duplicator can answer with the $\circ$ vertex in the corresponding block. On the other hand, if Spoiler pebbles an edge $\{x^i_j, y^i_j\}$, then Duplicator may answer with the $\circ$ vertex provided there is also an edge in her graph. The next lemma formalizes Duplicator's strategy.

Lemma 3.12. *Let $\ell \in [\mathsf{k}]$ and $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ be a configuration.*

1. *If $\mathrm{INC}^{\mathrm{right}}_\ell$ is applicable to $\mathfrak{q}$, then there is a winning strategy for Duplicator with boundary function $h^x_{\mathfrak{q}}$ on the input and $h^y_{\mathfrak{q}^+}$ on the output.*

2. *If $\mathrm{INC}^{\mathrm{right}}_\ell$ is not applicable to $\mathfrak{q}$, then there is a winning strategy for Duplicator with boundary function $h^x_{\mathfrak{q}}$ on the input and $h^y_{\mathfrak{q}_{\mathrm{inv}}}$ on the output for an invalid configuration $\mathfrak{q}_{\mathrm{inv}}$.*

Proof. To prove the first statement let $\mathfrak{q}$ be a valid configuration, hence $T = \emptyset$, such that $\mathrm{INC}^{\mathrm{right}}_\ell$ is applicable to $\mathfrak{q}$ and $\mathfrak{q}^+ = (\mathfrak{a}^+, \mathfrak{b}^+, \emptyset)$ be the incremented position. We claim that $h := h^x_{\mathfrak{q}} \cup h^y_{\mathfrak{q}^+}$ is a homomorphism from $\mathrm{INC}^{\mathrm{right}}_{\ell\ S}$ to $\mathrm{INC}^{\mathrm{right}}_{\ell\ D}$. It follows that $\mathcal{H} := \wp(h)$ is

a winning strategy with the desired boundary function. Since $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}^+}$ preserve vertex colors, it remains to verify that all edges were mapped to edges. Hence, we have to check that for all $i \in [\mathsf{k}]$ and $j \in [n]$ there is an edge between $h(x^i_j)$ and $h(y^i_j)$ in Duplicator's graph. Recall that $\mathfrak{a} = \mathfrak{a}^+$ whenever a right increment gadget is applicable to $\mathfrak{q}$. By definition

$$h(x^i_j) = \begin{cases} x^i_{\mathfrak{b}(i)}, \text{ if } j = \mathfrak{a}(i), \\ x^i_0, \text{ otherwise,} \end{cases} \qquad h(y^i_j) = \begin{cases} y^i_{\mathfrak{b}^+(i)}, \text{ if } j = \mathfrak{a}(i), \\ y^i_0, \text{ otherwise.} \end{cases}$$

Since there is an edge between $\circ$ vertices of the corresponding blocks it follows that $h(x^i_j)$ and $h(y^i_j)$ are adjacent for all $j \neq \mathfrak{a}(i)$. By the choice of $\mathfrak{q}$ and $\mathfrak{q}^+$ we have $\mathfrak{b}^+(i) = \mathfrak{b}(i)$ for all $i < \ell$, $\mathfrak{b}^+(\ell) = \mathfrak{b}(\ell) + 1$, $\mathfrak{b}(i) = m$ and $\mathfrak{b}^+(i) = 1$ for all $i > \ell$. Thus, by the definition of the gadget, there is an edge between $h(x^i_{\mathfrak{a}(i)}) = x^i_{\mathfrak{b}(i)}$ and $h(y^i_{\mathfrak{a}(i)}) = y^i_{\mathfrak{b}^+(i)}$.

For the second statement recall that $T^{\text{right}}_\ell(\mathfrak{q})$ (defined on page 54) is the set of blocks that do not satisfy the applicability condition. Assume that $\text{INC}^{\text{right}}_\ell$ is not applicable to $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ and hence $T^{\text{right}}_\ell(\mathfrak{q}) \neq \emptyset$. Note that on Duplicator's side of the gadget every vertex x^i_j has exactly one neighbor $y^i_{j'}$. We have designed the gadget such that the unique neighbor of $x^i_{\mathfrak{b}(i)}$ is y^i_0 if and only if $i \in T^{\text{right}}_\ell(\mathfrak{q})$. That is, if block i contradicts the applicability condition, then Duplicator has the chance to move to the $\circ$ vertex y^i_0 when Spoiler moves upwards. Hence, Duplicator can avoid a valid configuration on the output. Formally, Duplicator's strategy is $\wp(h^x_{\mathfrak{q}} \cup h^y_{\mathfrak{q}_{\text{inv}}})$ where $\mathfrak{q}_{\text{inv}} = (\mathfrak{a}, \mathfrak{b}_{\text{inv}}, T^{\text{right}}_\ell(\mathfrak{q}))$ with

$$\mathfrak{b}_{\text{inv}}(i) = \begin{cases} \text{arbitrary, if } i \in T^{\text{right}}_\ell(\mathfrak{q}), \\ j, \text{ such that } y^i_j \text{ is the neighbor of } x^i_{\mathfrak{b}(i)}, \text{ otherwise.} \end{cases} \quad \square$$

This concludes the strategies on the right increment gadget. In the remaining part of this paragraph we describe similar strategies for the left increment gadget. The gadget $\text{INC}^{\text{left}}_\ell$ is shown in Figure 3.6. It ensures that from a configuration $\mathfrak{q}$ (where $\text{INC}^{\text{left}}_\ell$ is applicable) on the input Spoiler can reach the incremented position $\mathfrak{q}^+$ on the

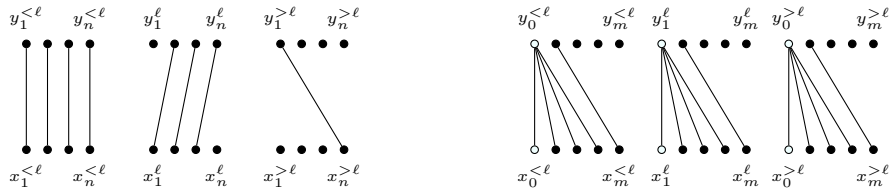


Figure 3.6.: The left increment gadget $\mathrm{INC}^{\mathrm{left}}_{\ell}$

output. That is, starting from a position

$$\{(x^1_{\mathfrak{a}(1)}, x^1_m), \ldots, (x^{\ell}_{\mathfrak{a}(\ell)}, x^{\ell}_m), (x^{\ell+1}_n, x^{\ell+1}_m), \ldots, (x^{\mathsf{k}}_n, x^{\mathsf{k}}_m)\}$$

Spoiler can reach

$$\{(y^1_{\mathfrak{a}(1)}, y^1_1), \ldots, (y^{\ell}_{\mathfrak{a}(\ell)+1}, y^{\ell}_1), (y^{\ell+1}_1, y^{\ell+1}_1), \ldots, (y^{\mathsf{k}}_1, y^{\mathsf{k}}_1)\}.$$

To achieve this, there are edges $\{x^i_j, y^i_j\}$ (for $i < \ell, j \in [n]$), $\{x^{\ell}_j, y^{\ell}_{j+1}\}$ (for $j \in [n-1]$) and $\{x^i_n, y^i_1\}$ (for $i > \ell$) in Spoiler's graph. In Duplicator's graph every vertex x^i_m is adjacent to y^i_1, all other input vertices x^i_j, $j < m$, are connected to the $\circ$ vertex y^i_0. The next lemma describes Spoiler's strategy, which is similar to the strategy on the right increment gadget (Lemma 3.11).

Lemma 3.13. *Let $\ell \in [\mathsf{k}]$, $\mathfrak{q}$ be a configuration such that $\mathrm{INC}^{\mathrm{left}}_{\ell}$ is applicable to $\mathfrak{q}$. Spoiler can reach $\mathfrak{q}^+$ on the output from $\mathfrak{q}$ on the input of the left increment gadget $\mathrm{INC}^{\mathrm{left}}_{\ell}$.*

Proof. Since $\mathrm{INC}^{\mathrm{left}}_{\ell}$ is applicable to $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ we have that $\mathfrak{a}(\ell) < n$, $\mathfrak{a}(i) = n$ for all $i > \ell$ and $\mathfrak{b}(i) = m$ for all $i \in [\mathsf{k}]$. Furthermore, $\mathfrak{q}^+ = (\mathfrak{a}^+, \mathfrak{b}^+, \emptyset)$ satisfies $\mathfrak{a}^+(i) = \mathfrak{a}(i)$ for all $i < \ell$, $\mathfrak{a}^+(\ell) = \mathfrak{a}(\ell) + 1$, $\mathfrak{a}^+(i) = 1$ for all $i > \ell$ and $\mathfrak{b}^+(i) = 1$ for all $i \in [\mathsf{k}]$. By definition of the gadget all edges $\{x^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{a}^+(i)}\}$ are present in Spoiler's graph. Moreover, $y^i_{\mathfrak{b}^+(i)} = y^i_1$ is the only neighbor of $x^i_{\mathfrak{b}(i)} = x^i_m$ in the corresponding

block. Hence, Spoiler can reach $\mathfrak{q}^+$ on the output by pebbling along these edges in the same way as described in Lemma 3.11. □

Again, Duplicator does not lose when Spoiler increments a position. Furthermore, if the increment gadget is not applicable to the current configuration, then Spoiler does not reach any valid position on the output. The next lemma ensures this and is the analogue of Lemma 3.12 for left increment gadgets.

Lemma 3.14. *Let $\ell \in [\mathsf{k}]$ and $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ be a configuration.*

1. *If $\mathrm{INC}_\ell^{\mathrm{left}}$ is applicable to $\mathfrak{q}$, then there is a winning strategy for Duplicator with boundary function $h^x_{\mathfrak{q}}$ on the input and $h^y_{\mathfrak{q}^+}$ on the output.*

2. *If $\mathrm{INC}_\ell^{\mathrm{left}}$ is not applicable to $\mathfrak{q}$, then there is a winning strategy for Duplicator with boundary function $h^x_{\mathfrak{q}}$ on the input and $h^y_{\mathfrak{q}_{\mathrm{inv}}}$ on the output for an invalid configuration $\mathfrak{q}_{\mathrm{inv}}$.*

Proof. Assume that $\mathrm{INC}_\ell^{\mathrm{left}}$ is applicable to $\mathfrak{q}$. As in the proof of Lemma 3.12 it is straightforward to check that $h = h^x_{\mathfrak{q}} \cup h^y_{\mathfrak{q}^+}$ is a homomorphism from Spoiler's side of the gadget to Duplicator's side: Every edge $\{x^i_j, x^i_{j'}\}$ is mapped to either $\{x^i_0, y^i_0\}$ or $\{x^i_{\mathfrak{b}(i)}, y^i_{\mathfrak{b}^+(i)}\} = \{x^i_m, y^i_1\}$. Hence, $\wp(h)$ is a winning strategy for Duplicator with the desired boundary function. To prove the second statement assume that $\mathrm{INC}_\ell^{\mathrm{left}}$ is not applicable to $\mathfrak{q}$ and hence $T_\ell^{\mathrm{left}}(\mathfrak{q}) \neq \emptyset$. Duplicator plays according to $h^x_{\mathfrak{q}}$ on the input vertices. If Spoiler pebbles some vertex y^i_j for an $i \in T_\ell^{\mathrm{left}}(\mathfrak{q})$ we consider two cases. The first case is that $\mathfrak{b}(i) \neq m$ or $i \in T$. Then y^i_0 is the neighbor of every $h^x_{\mathfrak{q}}(x^i_j)$ and Duplicator can safely move to y^i_0 (as in the proof of Lemma 3.12). The second case is that ($i = \ell$ and $\mathfrak{a}(\ell) = n$) or ($i > \ell$ and $\mathfrak{a}(i) \neq n$). In this situation Duplicator can answer with y^i_0 since on the one hand if $h^x_{\mathfrak{q}}(x^i_j) = x^i_0$, then there is an edge $\{x^i_0, y^i_0\}$ on Duplicator's side. On the other hand, if $h^x_{\mathfrak{q}}(x^i_j) \neq x^i_0$, then by definition $x^i_j = x^i_{\mathfrak{a}(i)}$. Since there is no edge between $x^i_{\mathfrak{a}(i)}$ and the ith block of the output vertices in Spoiler's graph, the choice of y^i_0 extends to a partial homomorphism. If Spoiler pebbles y^i_j for some $i \notin T_\ell^{\mathrm{left}}(\mathfrak{q})$, then

Duplicator answers with y_1^i as in part 1 of this lemma. Formally, Duplicator's strategy is $\wp(h^x_{\mathfrak{q}} \cup h^y_{\mathfrak{q}_{\text{inv}}})$ where $\mathfrak{q}_{\text{inv}} = (\mathfrak{a}_{\text{inv}}, \mathfrak{b}_{\text{inv}}, T^{\text{right}}_\ell(\mathfrak{q}))$ with $\mathfrak{b}_{\text{inv}}(i) = 1$ for all $i \in [\mathsf{k}]$ and

$$\mathfrak{a}_{\text{inv}}(i) = \begin{cases} \text{arbitrary, if } i \in T^{\text{right}}_\ell(\mathfrak{q}), \\ j, \text{ such that } y^i_j \text{ is the neighbor of } x^i_{\mathfrak{a}(i)}, \text{ otherwise.} \end{cases} \quad \square$$

The Switch

As for the increment gadgets, the switch has input vertices $x^1_1, \dots, x^{\mathsf{k}}_n$ in Spoiler's graph and $x^1_0, \dots, x^{\mathsf{k}}_m$ in Duplicator's graph, and output vertices $y^1_1, \dots, y^{\mathsf{k}}_n$ and $y^1_0, \dots, y^{\mathsf{k}}_m$, respectively.

For Spoiler the switch ensures that he can reach a valid configuration $\mathfrak{q}$ on the output from $\mathfrak{q}$ on the input (Lemma 3.16(i)). For Duplicator there are several strategies. She has a winning strategy called *output strategy*, where any position is on the output and $h^x_{\mathbf{0}}$ is on the input (Lemma 3.16(ii)).[6] This ensures that Spoiler cannot move backwards and reach $\mathfrak{q}$ on the input from $\mathfrak{q}$ on the output. Next, for every invalid $\mathfrak{q}$ Duplicator has a winning strategy where $h^x_{\mathfrak{q}}$ is on the input and $h^y_{\mathbf{0}}$ is on the output (Lemma 3.16(iii)). Thus, she has a strategy such that from invalid positions on the input Spoiler can only reach a position on the output that maps every vertex to some $\circ$ vertex y^i_0. These strategies are called *restart strategies*. We will see later that Spoiler has to restart the game, that is, he has to pick up all pebbles and start playing on the initialization gadget, if he reaches a position that is contained in a restart strategy. To ensure that Spoiler picks up all pebbles when reaching $\mathfrak{q}$ on the output from $\mathfrak{q}$ on the input, Duplicator has a critical *input strategy* with $\mathfrak{q}$ on the input and $h^y_{\mathbf{0}}$ on the output. The critical positions are either contained in an output strategy (where $\mathfrak{q}$ is on the output) or in a restart strategy (Lemma 3.16(iv)). If Duplicator plays according to this input strategy, the only way for Spoiler to bring $\mathfrak{q}$ from the input to the output is to pebble a critical position inside the switch and force Duplicator to switch to the corresponding output strategy.

[6] Recall that $h^x_{\mathbf{0}}(x^i_j) := x^i_0$ (see page 47).

Figure 3.7.: Subgraph of the switch. On Spoiler's side, all inner-block edges are present and the inter-block edges are indicated. For the first block on Duplicator's side, all inner-block edges are drawn. Note that there is no edge between $a^i_{s,l}$ and $b^i_{0,l'}$.

Remark 3.15. For the reader familiar with the literature it is worth noting that the switch presented here is an extension of the "multiple input one-way switch" defined in [Ber12a; Ber13]. The difference is that the old switch can only be used for the case $n = 1$. We will need this special case in Chapter 4. However, many strategies and technical definitions can directly be extended to this more general setting. The switch in [Ber12a; Ber13] was in turn a further development of the work from Kolaitis and Panttaja [KP03], who constructed a switch for the special case $n = 1$ and $m = 2$.

In order to define the *switch* we construct the two graphs: M_S for Spoiler's side and M_D for Duplicator's side. Let

$$\begin{aligned} V(M_S) =& \{x^i_j, a^i_j, b^i_j, y^i_j \mid i \in [\mathsf{k}], j \in [n]\}, \\ E(M_S) =& \big\{\{x^i_j, a^i_j\}, \{a^i_j, b^i_j\}, \{b^i_j, y^i_j\} \mid i \in [\mathsf{k}], j \in [n]\big\} \\ & \cup \big\{\{a^i_j, a^{i'}_{j'}\}, \{b^i_j, b^{i'}_{j'}\}, \{a^i_j, b^{i'}_{j'}\} \mid i, i' \in [\mathsf{k}]; i \neq i'; j, j' \in [n]\big\} \end{aligned}$$

That is, within one block $i \in [\mathsf{k}]$ of M_S the vertices $a^i_1, a^i_2, \ldots$ are pairwise connected to $b^i_1, b^i_2, \ldots$ and between two blocks i and i' every vertex a^i_j and b^i_j from block i is connected to every vertex $a^{i'}_{j'}$ and $b^{i'}_{j'}$ from block i'. For Duplicator's side of the graph, we define for $i \in [\mathsf{k}]$:

$$\begin{aligned} X^i &= \{x^i_s \mid 0 \leq s \leq m\}, & Y^i &= \{y^i_s \mid 0 \leq s \leq m\} \\ A^i_+ &= \{a^i_{s,l} \mid s \in [m], l \in [\mathsf{k}]\}, & A^i &= A^i_+ \cup \{a^i_0\} \\ B^i_+ &= \{b^i_{s,l} \mid s \in [m], l \in [\mathsf{k}]\}, & B^i &= B^i_+ \cup \{b^i_{0,l} \mid l \in [\mathsf{k}]\}. \end{aligned}$$

The set of vertices of M_D is

$$V(M_D) = \bigcup_{i \in [\mathsf{k}]} \left(X^i \cup A^i \cup B^i \cup Y^i\right).$$

The graphs consist of k blocks, where the i-th block contains all vertices with upper index i. Furthermore there are four types of variables (drawn in one row in Figure 3.3) the input vertices x, the output vertices y, the vertices a and b (with several indices). Every block of every type of vertices gets a unique color. That is, all x^i_j (y^i_j, a^i_j, b^i_j) in

M_S get the same color as the vertices X^i (Y^i, A^i, B^i, resp.) in M_D. This ensures that Duplicator always has to answer with vertices of the same type in the same block.

Now we describe the edges in M_D. We first define the inner-block edges E^i, which are also shown in Figure 3.3, and then the inter-block edges $E^{i,j}$:

$$
\begin{aligned}
E^i = &\big(\{x_0^i\} \times A^i\big) && \text{(E1)}\\
&\cup \big\{\{x_s^i, a_{s,l}^i\} \mid s \in [m]; l \in [\mathsf{k}]\big\} && \text{(E2)}\\
&\cup \big(\{a_0^i\} \times B^i\big) && \text{(E3)}\\
&\cup \big\{\{a_{s,l}^i, b_{s,l}^i\} \mid s \in [m]; l \in [\mathsf{k}]\big\} && \text{(E4)}\\
&\cup \big\{\{a_{s,l}^i, b_{0,l'}^i\} \mid s \in [m]; l, l' \in [\mathsf{k}]; l \neq l'\big\} && \text{(E5)}\\
&\cup \big\{\{b_{s,l}^i, y_s^i\} \mid s \in [m]; l \in [\mathsf{k}]\big\} && \text{(E6)}\\
&\cup \big\{\{b_{0,l}^i, y_s^i\} \mid s \in [m] \cup \{0\}; l \in [\mathsf{k}]\big\}, && \text{(E7)}\\
E^{i,j} = &\big\{\{a_{s,l}^i, a_{s',l'}^j\}, \mid s, s' \in [m]; l, l' \in [\mathsf{k}]; l \neq l'\big\} && \text{(E8)}\\
&\cup \big\{\{b_{s,l}^i, b_{s',l'}^j\} \mid s \in [m], s' \in [m] \cup \{0\}; l, l' \in [\mathsf{k}]; l \neq l'\big\} && \text{(E9)}\\
&\cup \big\{\{b_{0,l}^i, b_{0,l'}^j\} \mid l, l' \in [\mathsf{k}]\big\} && \text{(E10)}\\
&\cup \big\{\{a_{s,l}^i, b_{s',l'}^j\} \mid s \in [m]; s' \in [m] \cup \{0\}; l, l' \in [\mathsf{k}]; l \neq l'\big\} && \text{(E11)}\\
&\cup \big\{\{a_0^i, a_{s,l}^j\} \mid s \in [m]; l \in [\mathsf{k}]\big\} && \text{(E12)}\\
&\cup \big\{\{a_0^i, b_{s,l}^j\} \mid s \in [m] \cup \{0\}; l \in [\mathsf{k}]\big\} && \text{(E13)}
\end{aligned}
$$

Finally, $E(M_D) = \bigcup_{i \in [\mathsf{k}]} E^i \cup \bigcup_{i,j \in [\mathsf{k}]; i \neq j} E^{i,j}$. The next lemma states the main properties of the switch. For this, recall the definition of critical strategies (Definition 3.4 on page 44). The first statement (i) states that Spoiler can pebble a valid position from the input to the output. Duplicator uses the critical input strategies (iv) to ensure that Spoiler has to pebble a critical position inside the switch while he pebbles the valid position through the switch. Duplicator's output strategy (ii) ensures that Spoiler cannot move backwards (i. e., reach q on the input from q on the output). The restart strategy (iii) makes sure that Spoiler cannot pebble an invalid position through the switch.

Lemma 3.16. *For every configuration* $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$*, the following statements hold in the existential* $(\mathsf{k}+1)$*-pebble game on the switch:*

(i) If $\mathfrak{q}$ *is valid, then Spoiler can reach* $\mathfrak{q}$ *on the output from* $\mathfrak{q}$ *on the input.*

(ii) Duplicator has a winning strategy $\mathcal{H}_{\mathfrak{q}}^{out}$ *with boundary function* $h_{\mathbf{0}}^{x} \cup h_{\mathfrak{q}}^{y}$*.*

(iii) If $\mathfrak{q}$ *is invalid, then Duplicator has a winning strategy* $\mathcal{H}_{\mathfrak{q}}^{restart}$ *with boundary function* $h_{\mathfrak{q}}^{x} \cup h_{\mathbf{0}}^{y}$*.*

(iv) If $\mathfrak{q}$ *is valid, then Duplicator has a critical strategy* $\mathcal{H}_{\mathfrak{q}}^{in}$ *with boundary function* $h_{\mathfrak{q}}^{x} \cup h_{\mathbf{0}}^{y}$ *and sets of restart critical positions* $\mathcal{C}_{\mathfrak{q},t}^{restart\text{-}crit}$ *(for* $t \in [\mathsf{k}]$*) and output critical positions* $\mathcal{C}_{\mathfrak{q}}^{out\text{-}crit}$ *such that:*

a) $\operatorname{crit}(\mathcal{H}_{\mathfrak{q}}^{in}) = \bigcup_{t\in[\mathsf{k}]} \mathcal{C}_{\mathfrak{q},t}^{restart\text{-}crit} \cup \mathcal{C}_{\mathfrak{q}}^{out\text{-}crit}$,

b) $\mathcal{C}_{\mathfrak{q},t}^{restart\text{-}crit} \subseteq \mathcal{H}_{(\mathfrak{a},\mathfrak{b},\{t\})}^{restart}$ *and*

c) $\mathcal{C}_{\mathfrak{q}}^{out\text{-}crit} \subseteq \mathcal{H}_{\mathfrak{q}}^{out}$.

Proof. Let $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$ be an arbitrary configuration. We first construct the strategy for Spoiler to prove (i). Starting from position $\{(x^1_{\mathfrak{a}(1)}, x^1_{\mathfrak{b}(1)}), \dots, (x^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}, x^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k})})\}$, Spoiler places the $(\mathsf{k}+1)$st pebble on $a^1_{\mathfrak{a}(1)}$. Duplicator has to answer with $a^1_{\mathfrak{b}(1),l_1}$ for some $l_1 \in [\mathsf{k}]$, mapping the edge $\{x^1_{\mathfrak{a}(1)}, a^1_{\mathfrak{a}(1)}\}$ to some edge in (E2). Next, Spoiler picks up the pebble from $x^1_{\mathfrak{a}(1)}$ and puts it on $a^2_{\mathfrak{a}(2)}$. Again, Duplicator has to answer with $a^2_{\mathfrak{b}(2),l_2}$ for some $l_2 \in [\mathsf{k}] \setminus \{l_1\}$. The index l_2 has to be different from l_1 because there is an edge between $a^1_{\mathfrak{a}(1)}$ and $a^2_{\mathfrak{a}(2)}$, but none between $a^1_{\mathfrak{b}(1),l_1}$ and $a^2_{\mathfrak{b}(2),l_1}$ in (E8). Following that scheme, Spoiler can reach the position $\{(a^1_{\mathfrak{a}(1)}, a^1_{\mathfrak{b}(1),l_1}), \dots, (a^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}, a^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k}),l_{\mathsf{k}}})\}$ for pairwise distinct $l_1, l_2, \cdots, l_{\mathsf{k}}$. Now, Spoiler pebbles $b^1_{\mathfrak{a}(1)}$ with the free pebble and Duplicator has to answer with a vertex in B^1 (due to the vertex-colors) that is adjacent to all $a^1_{\mathfrak{b}(1),l_1}, \dots, a^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k}),l_{\mathsf{k}}}$. This is only the case for $b^1_{\mathfrak{b}(1),l_1}$ (due to (E4) and (E11)), since every vertex of the form b^1_{0,l_i} is not adjacent to the vertex $a^i_{\mathfrak{b}(i),l_i}$ according

to (E5) and (E11). Furthermore, $b^1_{\mathfrak{b}(1),l_1}$ is the only vertex of the form $b^1_{s,l}$ (for $s > 0$) that is adjacent to $a^i_{\mathfrak{b}(i),l_i}$. In the next step Spoiler picks up the pebble from $a^1_{\mathfrak{a}(1)}$ and puts it on $b^2_{\mathfrak{a}(2)}$. Duplicator has to answer with a vertex that is adjacent to all vertices $b^1_{\mathfrak{b}(1),l_1}, a^2_{\mathfrak{b}(2),l_2}, \ldots, a^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k}),l_{\mathsf{k}}}$. Because of the missing edges in (E5), (E11) and (E9) (!) the only vertex with this property is $b^2_{\mathfrak{b}(2),l_2}$. Again, Spoiler picks up the pebble from $a^2_{\mathfrak{a}(2)}$ and puts it on $b^3_{\mathfrak{a}(3)}$. By the same argument as before, Duplicator has to answer with $b^3_{\mathfrak{b}(3),l_3}$, which is the only vertex adjacent to all of $b^1_{\mathfrak{b}(1),l_1}, b^2_{\mathfrak{b}(2),l_2}, a^3_{\mathfrak{b}(3),l_3}, \ldots, a^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k}),l_{\mathsf{k}}}$. Thus, Spoiler can reach $\{(b^1_{\mathfrak{a}(1)}, b^1_{\mathfrak{b}(1),l_1}), \ldots, (b^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}, b^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k}),l_{\mathsf{k}}})\}$ and from there he reaches $\{(y^1_{\mathfrak{a}(1)}, y^1_{\mathfrak{b}(1)}), \ldots, (y^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}, y^{\mathsf{k}}_{\mathfrak{b}(\mathsf{k})})\}$ by successively pebbling the edges $\{b^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{a}(i)}\}$.

In order to derive the winning strategies for Duplicator in (ii) and (iii) we consider several total homomorphisms from Spoiler's to Duplicator's side. Consider the edges (E1), (E3) and (E7) connecting $\bullet$ vertices with $\circ$ vertices in one block of Duplicator's side. They can be used by Duplicator to pebble a $\circ$ vertex when Spoiler moves upwards. This is the crucial ingredient for Duplicator's output strategies (ii). The first homomorphism is used when Spoiler plays the above strategy to get a valid position through the switch and has already taken all his pebbles from the input vertices. If he tries to pebble input vertices again, then Duplicator can move to x^i_0 and plays according to the following homomorphism:

$$
\begin{aligned}
h^{\mathrm{out}}_{\mathfrak{q},\sigma}(x^i_j) &= x^i_0 \\
h^{\mathrm{out}}_{\mathfrak{q},\sigma}(a^i_{\mathfrak{a}(i)}) &= a^i_{\mathfrak{b}(i),\sigma(i)} & h^{\mathrm{out}}_{\mathfrak{q},\sigma}(a^i_j) &= a^i_0,\ j \neq \mathfrak{a}(i) \\
h^{\mathrm{out}}_{\mathfrak{q},\sigma}(b^i_{\mathfrak{a}(i)}) &= b^i_{\mathfrak{b}(i),\sigma(i)} & h^{\mathrm{out}}_{\mathfrak{q},\sigma}(b^i_j) &= b^i_{0,\sigma(j)},\ j \neq \mathfrak{a}(i) \\
h^{\mathrm{out}}_{\mathfrak{q},\sigma}(y^i_{\mathfrak{a}(i)}) &= y^i_{\mathfrak{b}(i)} & h^{\mathrm{out}}_{\mathfrak{q},\sigma}(y^i_j) &= y^i_0,\ j \neq \mathfrak{a}(i)
\end{aligned}
$$

where $\sigma \in S_{\mathsf{k}}$ is some permutation on $[\mathsf{k}]$. The next homomorphism is used by Duplicator when there is some valid or invalid configuration $\mathfrak{q}$ at the output of the switch.

$$h^{\mathrm{out}}_{\mathfrak{q}}(x^i_j) = x^i_0$$

$$h^{\text{out}}_{\mathfrak{q}}(a^i_j) = a^i_0$$
$$h^{\text{out}}_{\mathfrak{q}}(b^i_j) = b^i_{0,j}$$
$$h^{\text{out}}_{\mathfrak{q}}(y^i_j) = h^y_{\mathfrak{q}}(y^i_j)$$

Since $h^{\text{out}}_{\mathfrak{q}}$ and all $h^{\text{out}}_{\mathfrak{q},\sigma}$ are total,

$$\mathcal{H}^{\text{out}}_{\mathfrak{q}} := \begin{cases} \wp(h^{\text{out}}_{\mathfrak{q}}), & \mathfrak{q} \text{ is invalid,} \\ \wp(h^{\text{out}}_{\mathfrak{q}}) \cup \bigcup_{\sigma \in S_{\mathsf{k}}} \wp(h^{\text{out}}_{\mathfrak{q},\sigma}), & \text{otherwise,} \end{cases}$$

is a winning strategy for Duplicator satisfying (ii).

If a homomorphism maps all the $a^i_{\mathfrak{a}(i)}$ vertices to A^i_+, then it has to map all b^i vertices to B^i_+. This is due to the missing edges in (E5), (E11) and has also been used in Spoiler's strategy above. On the other hand, if for at least one $i \in [\mathsf{k}]$ all a^i_j are mapped to a^i_0, then every b^i_j can be mapped to $b^i_{0,l}$, where l is chosen such that $a^j_{\mathfrak{b}(j),l}$ is not in the image of the homomorphism for every j. Duplicator benefits from this, because she can now map the y^i_j vertices arbitrarily using the edges (E7). This behavior is used in the following restart strategies. Note that a homomorphism mapping some a^i_j to a^i_0 also maps x^i_j to x^i_0, hence restart strategies require invalid input positions. For invalid $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, T)$, let $\mathcal{H}^{\text{restart}}_{\mathfrak{q}} := \{\wp(h) \mid h \in H^{\text{restart}}_{\mathfrak{q}}\}$, where $H^{\text{restart}}_{\mathfrak{q}}$ is the set of total homomorphisms h satisfying the constraints $h(x^i_j) = h^x_{\mathfrak{q}}(x^i_j)$ and $h(y^i_j) = y^i_0$. This set clearly satisfies (iii). As an example fix some $t \in T$ and let $g \in H^{\text{restart}}_{\mathfrak{q}}$ be the following homomorphism:

$$g(x^i_j) = h^x_{\mathfrak{q}}(x^i_j),$$
$$g(a^i_j) = a^i_{\mathfrak{b}(i),i}, \text{ if } j = \mathfrak{a}(i) \text{ and } i \notin T,\ g(a^i_j) = a^i_0, \text{ otherwise,}$$
$$g(b^i_j) = b^i_{0,t},$$
$$g(y^i_j) = y^i_0.$$

It remains to consider the critical input strategies (iv). They formalize the following behavior of Duplicator at the time when Spoiler wants to pebble a configuration $\mathfrak{q}$ through the switch as in (i). Fix a valid

configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$. If Spoiler pebbles $a^i_{\mathfrak{a}(i)}$ or $b^i_{\mathfrak{a}(i)}$, Duplicator answers within A^i_+ or $B^i \setminus B^i_+$, respectively. This allows her to answer on the boundary according to the boundary function defined in (iv). However, she may run into trouble when Spoiler places k pebbles on $a^i_{\mathfrak{a}(i)}$ and $b^i_{\mathfrak{a}(i)}$ vertices, because they extend to a $(\mathsf{k}+1)$-clique on Spoiler's side, but not on Duplicator's side (on the blocks A^i_+ and $B^i \setminus B^i_+$). These positions form the critical positions where Duplicator switches to an output or restart strategy. If all k pebbles are on $a^1_{\mathfrak{a}(1)}, \ldots, a^{\mathsf{k}}_{\mathfrak{a}(\mathsf{k})}$, as in Spoiler's strategy (i), then Duplicator switches to the output strategy (i. e., she plays according to a homomorphism $h^{\text{out}}_{\mathfrak{q},\sigma}$). In all other cases she switches to a restart strategy. For all $\ell \in [\mathsf{k}]$ and permutations σ on $[\mathsf{k}]$ we define partial homomorphism $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ as follows:

$$
\begin{aligned}
h^{\text{in}}_{\mathfrak{q},\sigma,\ell}(x^i_j) &= h^x_{\mathfrak{q}}(x^i_j) \\
h^{\text{in}}_{\mathfrak{q},\sigma,\ell}(a^i_{\mathfrak{a}(i)}) &= a^i_{\mathfrak{b}(i),\sigma(i)},\ i \neq \sigma^{-1}(\ell) \\
h^{\text{in}}_{\mathfrak{q},\sigma,\ell}(a^i_{\mathfrak{a}(i)}) &= \text{undefined},\ i = \sigma^{-1}(\ell) \\
h^{\text{in}}_{\mathfrak{q},\sigma,\ell}(a^i_j) &= a^i_0,\ j \neq \mathfrak{a}(i) \\
h^{\text{in}}_{\mathfrak{q},\sigma,\ell}(b^i_j) &= b^i_{0,\ell} \\
h^{\text{in}}_{\mathfrak{q},\sigma,\ell}(y^i_j) &= y^i_0
\end{aligned}
$$

We need to check that $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ defines a homomorphism from $M_S \setminus \{a^{\sigma^{-1}(\ell)}_{\mathfrak{a}(i)}\}$ to M_D. For most parts this is easy to verify. The important part is to check that we do not map edges to the missing pairs in the edge sets (E5), (E8) and (E11) where we require that the indices l and l' have to be different. The constraints of (E8) are fulfilled because of the permutation σ. The constraints of (E5) and (E11) are satisfied because we have chosen ℓ such that no vertex maps to $a^i_{s,\ell}$ for all $i \in [\mathsf{k}]$ and $s \in [m]$. This also shows that the partial homomorphism cannot be extended to a total homomorphism (where $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ is defined on $a^i_{\mathfrak{a}(i)}$ for $i = \sigma^{-1}(\ell)$). Now we define a partial homomorphism $h^{\text{in}}_{\mathfrak{q},\sigma}$

for every permutation $\sigma \in S_{\mathsf{k}}$.

$$\begin{aligned} h^{\text{in}}_{\mathfrak{q},\sigma}(x^i_j) &= h^x_{\mathfrak{q}}(x^i_j), \\ h^{\text{in}}_{\mathfrak{q},\sigma}(a^i_{\mathfrak{a}(i)}) &= a^i_{\mathfrak{b}(i),\sigma(i)}, \\ h^{\text{in}}_{\mathfrak{q},\sigma}(a^i_j) &= a^i_0,\ j \neq \mathfrak{a}(i), \\ h^{\text{in}}_{\mathfrak{q},\sigma}(b^i_j) &= \text{undefined}, \\ h^{\text{in}}_{\mathfrak{q},\sigma}(y^i_j) &= y^i_0. \end{aligned}$$

Again it is not hard to see that $h^{\text{in}}_{\mathfrak{q},\sigma}$ defines a partial homomorphism from M_S to M_D. We cannot extend this partial homomorphism to a total homomorphism, because if we map $b^i_{\mathfrak{a}(i)}$ to some $b^i_{0,l}$ we will map to a missing edge in (E5) or (E11). Otherwise, if we chose some $b^i_{\mathfrak{b}(i),l}$, we will map the edge $\{b^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{a}(i)}\}$ in M_S to the non-edge $\{b^i_{\mathfrak{b}(i),l}, y^i_0\}$ in M_D. Duplicator's input strategy is the family of all subsets of all mappings $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ and $h^{\text{in}}_{\mathfrak{q},\sigma}$. We are ready to define the critical positions. For all $\sigma \in S_{\mathsf{k}}$ let

$$h^{\text{out-crit}}_{\mathfrak{q},\sigma} := \{(a^i_{\mathfrak{a}(i)}, a^i_{\mathfrak{b}(i),\sigma(i)}) \mid i \in [\mathsf{k}]\}$$

and for all $\sigma \in S_{\mathsf{k}}$ and $t, u \in [\mathsf{k}]$ and $s \in [n]$

$$h^{\text{restart-crit}}_{\mathfrak{q},\sigma,t,u,s} := \{(a^i_{\mathfrak{a}(i)}, a^i_{\mathfrak{b}(i),\sigma(i)}) \mid i \in [\mathsf{k}] \setminus \{t\}\} \cup \{(b^u_s, b^u_{0,\sigma(t)})\}.$$

Now we can define the sets used in (iv):

$$\begin{aligned} \mathcal{H}^{\text{in}}_{\mathfrak{q}} &= \{\wp(h^{\text{in}}_{\mathfrak{q},\sigma}) \mid \sigma \in S_{\mathsf{k}}\} \cup \{\wp(h^{\text{in}}_{\mathfrak{q},\sigma,\ell}) \mid \sigma \in S_{\mathsf{k}}, \ell \in [\mathsf{k}]\}, \\ \mathcal{C}^{\text{out-crit}}_{\mathfrak{q}} &= \{h^{\text{out-crit}}_{\mathfrak{q},\sigma} \mid \sigma \in S_{\mathsf{k}}\}, \\ \mathcal{C}^{\text{restart-crit}}_{\mathfrak{q},t} &= \{h^{\text{restart-crit}}_{\mathfrak{q},\sigma,t,u,s} \mid \sigma \in S_{\mathsf{k}}, u \in [\mathsf{k}], s \in [n]\}, \\ \text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}}) &= \bigcup_{t \in [\mathsf{k}]} \mathcal{C}^{\text{restart-crit}}_{\mathfrak{q},t} \cup \mathcal{C}^{\text{out-crit}}_{\mathfrak{q}}. \end{aligned}$$

First note that $h^{\text{out-crit}}_{\mathfrak{q},\sigma} \subset h^{\text{in}}_{\mathfrak{q},\sigma}$ and $h^{\text{restart-crit}}_{\mathfrak{q},\sigma,t,u,s} \subset h^{\text{in}}_{\mathfrak{q},\sigma,\sigma(t)}$. It holds that $\text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}}) \subseteq \mathcal{H}^{\text{in}}_{\mathfrak{q}}$. It easily follows from the definitions, that $h^{\text{out-crit}}_{\mathfrak{q},\sigma} \subset h^{\text{out}}_{\mathfrak{q},\sigma}$. Furthermore, every $h^{\text{restart-crit}}_{\mathfrak{q},\sigma,t,u,s}$ can be extended to a

homomorphism $g \in \mathcal{H}^{\text{restart}}_{(\mathfrak{a},\mathfrak{b},\{t\})}$ by defining

$$\begin{aligned}
g(x^i_j) &= h^x_{(\mathfrak{a},\mathfrak{b},\{t\})}(x^i_j),\\
g(a^i_{\mathfrak{a}(i)}) &= h^{\text{restart-crit}}_{\mathfrak{q},\sigma,t,u,s}(a^i_{\mathfrak{a}(i)}) = a^i_{\mathfrak{b}(i),\sigma(i)}, \text{ if } i \neq t,\\
g(a^t_{\mathfrak{a}(t)}) &= a^t_0,\\
g(a^i_j) &= a^i_0, \text{ if } j \neq \mathfrak{a}(i),\\
g(b^i_j) &= b^i_{\sigma(t)},\\
g(y^i_j) &= y^i_0.
\end{aligned}$$

This proves statement b) and c) from (iv). It remains to show that $\mathcal{H}^{\text{in}}_{\mathfrak{q}}$ is a critical strategy with critical positions $\text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}})$.

Claim 3.17. For all $g \in \mathcal{H}^{\text{in}}_{\mathfrak{q}}$ with $|g| \leq \mathsf{k}$, either $g \in \text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}})$ or for all $z \in V(M_S)$ there exist an $h \in \mathcal{H}^{\text{in}}_{\mathfrak{q}}$, such that $g \subseteq h$ and $z \in \text{Dom}(h)$.

Proof. As g is a partial homomorphism from $\mathcal{H}^{\text{in}}_{\mathfrak{q}}$ (which only contains subsets of $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ and $h^{\text{in}}_{\mathfrak{q},\sigma}$), we can fix some $\sigma \in S_{\mathsf{k}}$ and $\ell \in [\mathsf{k}]$ such that g is a subset of the following mapping

$$\begin{aligned}
x^i_{\mathfrak{a}(i)} &\mapsto x^i_{\mathfrak{b}(i)}, & x^i_j &\mapsto x^i_0, \text{ if } j \neq \mathfrak{a}(i),\\
a^i_{\mathfrak{a}(i)} &\mapsto a^i_{\mathfrak{b}(i),\sigma(i)}, & a^i_j &\mapsto a^i_0, \text{ if } j \neq \mathfrak{a}(i),\\
b^i_j &\mapsto b^i_{0,\ell},\\
y^i_j &\mapsto y^i_0.
\end{aligned}$$

Let $B_S := \{b^i_j \mid i \in [\mathsf{k}], j \in [n]\} \subseteq V(M_S)$.
Case 1: $|\text{Dom}(g) \cap \{a^i_{\mathfrak{a}(i)} \mid i \in [\mathsf{k}]\}| = \mathsf{k}$**.** In this case, $g = h^{\text{out-crit}}_{\mathfrak{q},\sigma}$ and hence, $g \in \text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}})$.
Case 2: $|\text{Dom}(g) \cap \{a^i_{\mathfrak{a}(i)} \mid i \in [\mathsf{k}]\}| = \mathsf{k} - 1$**.** If $\text{Dom}(g) \cap B_S \neq \emptyset$, then $g = h^{\text{restart-crit}}_{\mathfrak{q},\sigma,\sigma^{-1}(l),u,s}$ for some $u \in [\mathsf{k}]$ and $s \in [n]$. Thus, we can assume that $\text{Dom}(g) \cap B_S = \emptyset$ and show for all z that g satisfies the

extension property. If $z = a^i_j$, then $h^{\text{in}}_{\mathfrak{q},\sigma}$ extends g. If $z = x^i_j$,$z = b^i_j$ or $z = y^i_j$, then $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ extends g.

Case 3: $|\operatorname{Dom}(g) \cap \{a^i_{\mathfrak{a}(i)} \mid i \in [\mathsf{k}]\}| \leq \mathsf{k} - 2$**.** Let j_1 and j_2 be two distinct indices such that $a^{j_1}_{\mathfrak{a}(j_1)}, a^{j_2}_{\mathfrak{a}(j_2)} \notin \operatorname{Dom}(g)$. Furthermore, we can without loss of generality assume that $\sigma(j_1) = \ell$. For $z \neq a^{j_1}_{\mathfrak{a}(j_1)}$ the homomorphism $h^{\text{in}}_{\mathfrak{q},\sigma,\ell}$ extends g. If $z = a^{j_1}_{\mathfrak{a}(j_1)}$, then $h^{\text{in}}_{\mathfrak{q},\sigma',\ell}$ extends g, where $\sigma' := \{(i,\sigma(i)) \mid i \in [\mathsf{k}] \setminus \{j_1,j_2\}\} \cup \{(j_1,\sigma(j_2)),(j_2,\sigma(j_1))\}$. ⊣

□

The Initialization Gadget

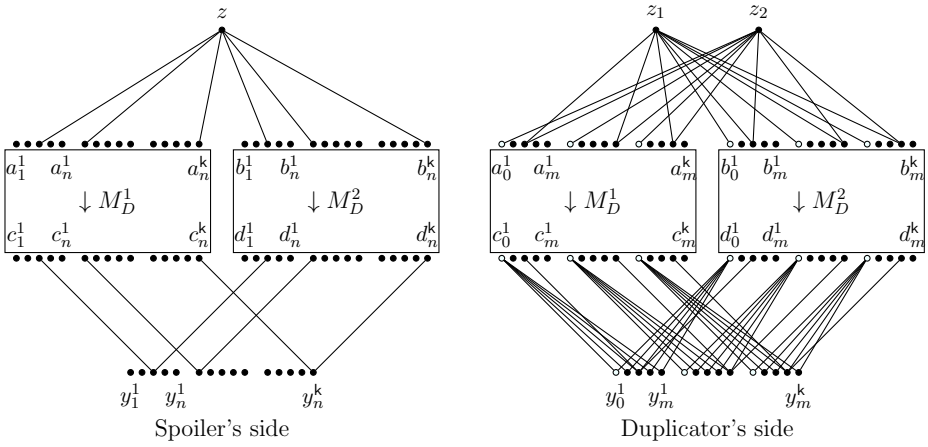


Figure 3.8.: The initialization gadget (for $\mathsf{k} = 3$, $m = 4$, $n = 5$, $\mathfrak{a}(1) = 3$, $\mathfrak{a}(2) = 1$, $\mathfrak{b}(3) = 5$, $\mathfrak{b}(1) = 2$, $\mathfrak{b}(2) = 4$, $\mathfrak{b}(3) = 3$.)

At the beginning of the game we want that Spoiler can reach the start configuration $\alpha^{-1}(0)$ on x, which is the pebble position $\{(\mathsf{x}^1_1,\mathsf{x}^1_1),\ldots,(\mathsf{x}^k_1,\mathsf{x}^k_1)\}$. To ensure this, we introduce an initialization gadget and identify its output vertices y^i_j with the block of x^i_j vertices. The main property is that Spoiler can force the start configuration on the output of the gadget. Another additional property is that

from any position on the output of that gadget Duplicator does not lose. This property causes the main difficulties and is needed because other positions than the start position occur on the x vertices during the course of the game. In Chapter 4 we also need to initialize the game with other configurations than $\alpha^{-1}(0)$. For this, we define the initialization gadget more generally for every valid configuration $\mathfrak{q}$.

The initialization gadget $\text{INIT}^{\mathfrak{q}}$ is built out of two switches M^1 and M^2, vertices z in Spoiler's graph and z_1, z_2 in Duplicator's graph. The three vertices z, z_1, z_2 share one unique vertex color. Additionally, there are output boundary vertices y^i_j of the usual form. The vertices z, z_1, z_2 and the boundary vertices are connected to M^1 and M^2 as shown in Figure 3.8 for a specific valid configuration $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$. Lemma 3.18 (i)–(iii) provides the strategies on $\text{INIT}^{\mathfrak{q}}$. The main property is that Spoiler can reach the start position $\mathfrak{q}$ at the boundary (i) and Duplicator has a corresponding counter strategy (ii) in this situation. Furthermore, if an arbitrary position occurs at the boundary during the game, Duplicator has a strategy to survive (iii).

Lemma 3.18. *For every valid configuration* $\mathfrak{q} = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ *the following holds in the existential* $(\mathsf{k}+1)$*-pebble game on* $\text{INIT}^{\mathfrak{q}}$*:*

(i) Spoiler can reach $\mathfrak{q}$ *on the output.*

(ii) There is a winning strategy $\mathcal{I}^{init}$ *for Duplicator with boundary function* $h^y_{\mathfrak{q}}$*.*

(iii) For every (valid or invalid) configuration $\mathfrak{q}'$ *there is a critical strategy* $\mathcal{I}^{init}_{\mathfrak{q}'}$ *with boundary function* $h^y_{\mathfrak{q}'}$ *and* $\text{crit}(\mathcal{I}^{init}_{\mathfrak{q}'}) \subseteq \mathcal{I}^{init}$*.*

Spoiler's strategy is quite simple. First he pebbles z. Duplicator has to answer with either z_1 or z_2. Then Spoiler can reach $\{(x^i_{\mathfrak{a}(i)}, x^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$ by pebbling through either M^1 or M^2. To construct the strategies for Duplicator, we can combine the strategies of the switches M^1 and M^2 such that she plays an input strategy on one switch and a restart or output strategy on the other switch. Assume that Spoiler reaches a critical position on the switch where Duplicator plays the input strategy, say M^1. Duplicator can now flip the strategies such that she plays a restart or output strategy on M^1,

depending on which kind of critical position Spoiler has reached, and an input strategy on M^2.

Proof of Lemma 3.18. We start with developing the strategy for Spoiler (i). First, Spoiler pebbles z. Duplicator has to respond with either z_1 or z_2. Depending on Duplicator's choice, Spoiler can reach either $\{(a^i_{\mathfrak{a}(i)}, a^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$ or $\{(b^i_{\mathfrak{a}(i)}, b^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$. By Lemma 3.16.(i) Spoiler reaches $\{(c^i_{\mathfrak{a}(i)}, c^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$ ($\{(d^i_{\mathfrak{a}(i)}, d^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$) and from there he can reach the position $\{(y^i_{\mathfrak{a}(i)}, y^i_{\mathfrak{b}(i)}) \mid i \in [\mathsf{k}]\}$. For Duplicator's strategies we start with a discussion of possible moves outside of the switches. At the top of the gadget Duplicator can map z to z_1 and is then forced to answer with $h^a_{\mathfrak{q}}$ at the input of M^1 and for some $R \subseteq [\mathsf{k}]$ with $h^b_{(\mathfrak{a},\mathfrak{b},R)}$ at the input of M^2. On the other hand, Duplicator can map z to z_2 and play according to $h^a_{(\mathfrak{a},\mathfrak{b},R)}$ and $h^b_{\mathfrak{q}}$. At the bottom of the switch the following three combinations define partial homomorphisms for all configurations $\mathfrak{q}'$:

$$h^c_{\mathbf{0}} \cup h^d_{\mathbf{0}} \cup h^y_{\mathfrak{q}'}$$
$$h^c_{\mathfrak{q}} \cup h^d_{\mathbf{0}} \cup h^y_{\mathfrak{q}}$$
$$h^c_{\mathbf{0}} \cup h^d_{\mathfrak{q}} \cup h^y_{\mathfrak{q}}$$

Now we can combine these partial strategies with the strategies on the switches described in Lemma 3.16. In strategy $\mathcal{I}^{\text{in-}i}_{t,\mathfrak{q}'}$ Duplicator plays an input strategy on switch i, a restart strategy on the other switch and according to an arbitrary configuration $\mathfrak{q}'$ on the y-block. These strategies were combined to the critical strategy $\mathcal{I}^{\text{init}}_{\mathfrak{q}'}$ described in (iii).[7]

$$\mathcal{I}^{\text{in-1}}_{t,\mathfrak{q}'} := \wp(\{(z, z_1)\}) \uplus \mathcal{H}^{\text{in}}_{\mathfrak{q}}\langle M^1\rangle \uplus \mathcal{H}^{\text{restart}}_{(\mathfrak{a},\mathfrak{b},\{t\})}\langle M^2\rangle \uplus \wp(h^y_{\mathfrak{q}'})$$
$$\mathcal{I}^{\text{in-2}}_{t,\mathfrak{q}'} := \wp(\{(z, z_2)\}) \uplus \mathcal{H}^{\text{restart}}_{(\mathfrak{a},\mathfrak{b},\{t\})}\langle M^1\rangle \uplus \mathcal{H}^{\text{in}}_{\mathfrak{q}}\langle M^2\rangle \uplus \wp(h^y_{\mathfrak{q}'})$$

[7] In the definition of the combined strategies we use the operator $\uplus$ as defined in the paragraph before Lemma 3.7. The careful reader might notice that we do not connect "connectable strategies" on gadgets and thus cannot apply Lemma 3.7 literally. However, by defining the edges outside of the switches as additional gadget one could arrange the definitions (with an ugly overload of notation) to fit.

$$\mathcal{I}^{\text{init}}_{\mathfrak{q}'} := \bigcup_{t\in[\mathsf{k}]} (\mathcal{I}^{\text{in-1}}_{t,\mathfrak{q}'} \cup \mathcal{I}^{\text{in-2}}_{t,\mathfrak{q}'})$$

All critical positions of $\mathcal{I}^{\text{in-}i}_{t,\mathfrak{q}'}$ are restart or output critical positions on the switch M^i. By Lemma 3.16.(iv).(b) every restart critical position of $\mathcal{I}^{\text{in-1}}_{t,\mathfrak{q}'}$ is contained in one of the strategies $\mathcal{I}^{\text{in-2}}_{t,\mathfrak{q}'}$ as non-critical position. Hence, the only critical positions $\text{crit}(\mathcal{I}^{\text{init}}_{\mathfrak{q}'})$ of the combined strategy are output critical positions on the switches. These output critical positions will be contained in the strategies $\mathcal{I}^{\text{init-}i}$ where Duplicator plays an output strategy on switch i. Together with $\mathcal{I}^{\text{init}}_{\mathfrak{q}}$ they form the winning strategy $\mathcal{I}^{\text{init}}$ from (ii).

$$\begin{aligned}
\mathcal{I}^{\text{init-1}} &:= \wp(\{(z,z_2)\}) \uplus \mathcal{H}^{\text{out}}_{\mathfrak{q}}\langle M^1\rangle \uplus \mathcal{H}^{\text{in}}_{\mathfrak{q}}\langle M^2\rangle \uplus \wp(h^y_{\mathfrak{q}})\\
\mathcal{I}^{\text{init-2}} &:= \wp(\{(z,z_1)\}) \uplus \mathcal{H}^{\text{in}}_{\mathfrak{q}}\langle M^1\rangle \uplus \mathcal{H}^{\text{out}}_{\mathfrak{q}}\langle M^2\rangle \uplus \wp(h^y_{\mathfrak{q}})\\
\mathcal{I}^{\text{init}} &:= \mathcal{I}^{\text{init-1}} \cup \mathcal{I}^{\text{init-2}} \cup \mathcal{I}^{\text{init}}_{\mathfrak{q}}
\end{aligned}$$

$\mathcal{I}^{\text{init}}$ is a union of critical strategies with boundary function $h^y_{\mathfrak{q}}$. To prove that $\mathcal{I}^{\text{init}}$ is indeed a winning strategy on the gadget, we apply Lemma 3.5 and show that every critical position of one strategy is contained as non-critical position in another strategy. Critical positions are inside the input strategy $\mathcal{H}^{\text{in}}_{\mathfrak{q}}$ on one of the switches. By Lemma 3.16.(iv) they are either contained in an output or restart strategy on the corresponding switch. Hence, all restart critical positions on M^1 and M^2 are contained in $\mathcal{I}^{\text{init}}_{\mathfrak{q}}$ and all output critical positions on M^1 (M^2) are contained in $\mathcal{I}^{\text{init-1}}$ ($\mathcal{I}^{\text{init-2}}$). Recall the notation $\widehat{\mathcal{S}} := \mathcal{S} \setminus \text{crit}(\mathcal{S})$, by Lemma 3.16.(iv) we get:

$$\begin{aligned}
\text{crit}(\mathcal{I}^{\text{in-2}}_{R,\mathfrak{q}'}) = \text{crit}(\mathcal{I}^{\text{init-1}}) &= \text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}}\langle M^2\rangle)\\
&\subseteq \mathcal{H}^{\text{out}}_{\mathfrak{q}}\langle M^2\rangle \cup \bigcup_{t\in[\mathsf{k}]} \mathcal{H}^{\text{restart}}_{(\mathfrak{q},\{t\})}\langle M^2\rangle\\
&\subseteq \widehat{\mathcal{I}}^{\text{init-2}} \cup \bigcup_{t\in[\mathsf{k}]} \widehat{\mathcal{I}}^{\text{in-1}}_{\{t\},\mathfrak{q}},\\
\text{crit}(\mathcal{I}^{\text{in-1}}_{R,\mathfrak{q}'}) = \text{crit}(\mathcal{I}^{\text{init-2}}) &= \text{crit}(\mathcal{H}^{\text{in}}_{\mathfrak{q}}\langle M^1\rangle)
\end{aligned}$$

$$\subseteq \mathcal{H}^{\text{out}}_{\mathfrak{q}}\langle M^1\rangle \cup \bigcup_{t\in[\mathsf{k}]} \mathcal{H}^{\text{restart}}_{(\mathfrak{q},\{t\})}\langle M^1\rangle$$
$$\subseteq \widehat{\mathcal{I}}^{\text{init-1}} \cup \bigcup_{t\in[\mathsf{k}]} \widehat{\mathcal{I}}^{\text{in-2}}_{\{t\},\mathfrak{q}}.$$

Hence, $\operatorname{crit}(\mathcal{I}^{\text{init}}_{\mathfrak{q}'}) \subseteq \mathcal{I}^{\text{init}}$ and $\mathcal{I}^{\text{init}}$ is a winning strategy by Lemma 3.5. □

3.5. Proof of the Lower Bound

By describing all the gadgets in the previous section we have finished the definition of the structures A_n and B_m. Note that the overall construction uses a constant number of gadgets and the size of the vertex set in every gadget is linear in n on Spoiler's side and linear in m on Duplicator's side. It follows that $|V(\mathsf{A}_n)| = O(n)$ and $|V(\mathsf{B}_m)| = O(m)$. It remains to prove the lower bound on the number of rounds Spoiler needs to win the existential $(\mathsf{k}+1)$-pebble game. For this it suffices to prove the following two lemmas.

Lemma 3.19. *Spoiler has a winning strategy in the existential* $(\mathsf{k}+1)$*-pebble game on* A_n *and* B_m*.*

Lemma 3.20. *There is a sequence of critical strategies for Duplicator* $\mathcal{G}^{\text{start}}, \mathcal{F}_1, \mathcal{G}_1, \mathcal{F}_2, \mathcal{G}_2, \ldots, \mathcal{G}_{n^{\mathsf{k}}m^{\mathsf{k}}-2}, \mathcal{F}_{n^{\mathsf{k}}m^{\mathsf{k}}-1}$ *such that*

$$\begin{aligned}
\operatorname{crit}(\mathcal{G}^{\text{start}}) &\subseteq \widehat{\mathcal{F}}_1, \\
\operatorname{crit}(\mathcal{G}_i) &\subseteq \widehat{\mathcal{F}}_{i+1} \cup \widehat{\mathcal{G}}^{\text{start}}, && 1 \le i \le n^{\mathsf{k}}m^{\mathsf{k}} - 2, \\
\operatorname{crit}(\mathcal{F}_i) &\subseteq \widehat{\mathcal{G}}_i \cup \widehat{\mathcal{G}}^{\text{start}}, && 1 \le i \le n^{\mathsf{k}}m^{\mathsf{k}} - 2.
\end{aligned}$$

Proof of Theorem 1. For the existential 2-pebble game the theorem follows from Lemma 3.1. For $k \ge 3$ consider the structures A_n and B_m (for $\mathsf{k} = k-1$) defined above. By Lemma 3.19 Spoiler wins the existential k-pebble game on A_n and B_m. From Lemma 3.20 it follows via Lemma 3.6 that Spoiler needs at least $\Omega(n^{k-1}m^{k-1})$ rounds to win the game. To get structures with exactly n and m vertices we take

the largest n', m' such that $|V(\mathsf{A}_{n'})| \leq n$, $|V(\mathsf{B}_{m'})| \leq m$ and fill up the structures with an appropriate number of isolated vertices. □

Proof of Lemma 3.19. To show that Spoiler has a winning strategy it suffices to prove the following three statements:

(1) Spoiler can reach the position $\alpha^{-1}(0)$ on x from $\emptyset$,

(2) Spoiler can reach $\alpha^{-1}(i+1)$ on x from $\alpha^{-1}(i)$ on x and

(3) Spoiler wins from $\alpha^{-1}(n^{\mathsf{k}}m^{\mathsf{k}} - 1)$ on x.

The first assertion (1) follows from Spoiler's strategy on the initialization gadget (Lemma 3.18). Statement (3) can be obtained by playing on the winning gadget (Lemma 3.8). For (2), Spoiler starts with the position $\mathfrak{q} = \alpha^{-1}(i)$ on x. Since $i < n^{\mathsf{k}}m^{\mathsf{k}} - 1$ there is exactly one increment gadget applicable to $\mathfrak{q}$. Spoiler uses either Lemma 3.11 or Lemma 3.13 to reach $\mathfrak{q}^+ = \alpha^{-1}(i+1)$ on the output of that gadget. By applying Lemma 3.16.(i) twice, Spoiler can pebble $\mathfrak{q}^+$ through the two switches to the x vertices. □

Proof of Lemma 3.20. To define the sequence of critical strategies we use the partial critical strategies on the gadgets (as defined in the lemmas 3.9, 3.12, 3.14, 3.16 and 3.18) and combine them according to Lemma 3.7. We name the unique switch whose output boundary is identified with the x-vertices "the single switch" (see Figure 3.3). The vertex block at the input of that switch (and on the output of all other switches) is denoted by y. There are three types of strategies: $\mathcal{G}^{\text{start}}$, $\mathcal{F}_i$ and $\mathcal{G}_i$.

To define $\mathcal{G}_i$ we let $\mathfrak{q} = \alpha^{-1}(i)$. Duplicator plays according to $h^{\mathsf{x}}_{\mathfrak{q}}$ on x and according to $h^{\mathsf{y}}_{\mathbf{0}}$ on y. She plays according to this strategy in the case when Spoiler reaches "$\mathfrak{q}$ on x". The critical strategy $\mathcal{G}_i$ is the combination (using the $\uplus$ operator from Lemma 3.7) of the following (pairwise connectable) strategies on the gadgets:

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathfrak{q}}$ on the initialization gadget (L. 3.18).
- The winning strategy with boundary $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}^+}$ on the increment gadget applicable to $\mathfrak{q}$ (L. 3.12/3.14).

- The critical input strategy $\mathcal{H}^{\text{in}}_{\mathfrak{q}^+}$ on the switch following the applicable increment gadget (L. 3.16).
- The winning strategy with boundary $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}_{\text{inv}}}$ on the other increment gadgets not applicable to $\mathfrak{q}$ (L. 3.12/3.14).
- The restart winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_{\text{inv}}}$ on the switches following the inapplicable increment gadgets (L. 3.16). Here, $\mathfrak{q}_{\text{inv}}$ is the invalid configuration on the output the corresponding increment gadget.
- The output winning strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}}$ on the single switch (L. 3.16).

If in the above setting Spoiler increments $\mathfrak{q}$ through the applicable increment gadget and moves $\mathfrak{q}^+ = \alpha^{-1}(i+1)$ through the subsequent switch, then Duplicator switches to the strategy $\mathcal{F}_{i+1}$.

To define $\mathcal{F}_i$ we fix $\mathfrak{q} = \alpha^{-1}(i)$. In this strategy, Duplicator plays according to $h^{\mathsf{x}}_{\mathbf{0}}$ on x and according to $h^{\mathsf{y}}_{\mathfrak{q}}$ on y. This critical strategy is the combination of the following strategies on the gadgets.

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathbf{0}}$ on the initialization gadget (L. 3.18).
- The winning strategy with boundary $h^x_{\mathbf{0}}$ and $h^y_{\mathbf{0}}$ on the increment gadgets (L. 3.12/3.14).
- The output strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}}$ on the switches following the increment gadgets (L. 3.16).
- The critical input strategy $\mathcal{H}^{\text{in}}_{\mathfrak{q}}$ on the single switch (L. 3.16).

The critical positions in the strategies $\mathcal{G}_i$ and $\mathcal{F}_i$ are within the switches and the initialization gadget. Recall that by Lemma 3.16.(iv) the critical positions on the switch can be divided into restart critical positions and output critical positions. Furthermore, all output critical positions of $\mathcal{G}_i$ (which are inside the switch following the applicable increment gadget) are contained as non-critical positions in $\mathcal{F}_{i+1}$. All output critical position in $\mathcal{F}_i$ (which are inside the single switch) are contained as non-critical positions in $\mathcal{G}_i$. Now we define $\mathcal{G}^{\text{start}}$ that contains all other critical positions of $\mathcal{G}_i$ and $\mathcal{F}_i$. The critical strategy $\mathcal{G}^{\text{start}}$ is the union of several other global strategies (cf. Lemma 3.5). The first one is $\mathcal{G}^{\text{init}}$ which contains in particular the winning strategy

$\mathcal{I}^{\text{init}}$ on the initialization gadget. Thus, by Lemma 3.18, it contains every critical position on the initialization gadget as non-critical position. Additionally, $\mathcal{G}^{\text{init}}$ combines the following strategies on the gadgets for the start configuration $\mathfrak{q} = \alpha^{-1}(0)$:

- The winning strategy $\mathcal{I}^{\text{init}}$ on the initialization gadget.
- The winning strategy with boundary $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}^+}$ on the increment gadget applicable to $\mathfrak{q}$.
- The critical input strategy $\mathcal{H}^{\text{in}}_{\mathfrak{q}^+}$ on the switch following the applicable increment gadget.
- The winning strategy with boundary $h^x_{\mathfrak{q}}$ and $h^y_{\mathfrak{q}_{\text{inv}}}$ on the other increment gadgets not applicable to $\mathfrak{q}$.
- The restart winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_{\text{inv}}}$ on the switches following those inapplicable increment gadgets.
- The output winning strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}}$ on the single switch.

Note that the output critical positions of $\mathcal{G}^{\text{init}}$ are contained as non-critical positions in $\mathcal{F}_1$. Since $\mathcal{G}^{\text{init}}$ handles the critical positions on the initialization gadget and we discussed the output critical positions on the switches, it remains to consider the restart critical positions of the strategies. For this we construct a strategy $\mathcal{G}^{\text{restart}}_i$ to handle the restart critical positions of $\mathcal{G}_i$ (for $i \geq 1$) and of $\mathcal{G}^{\text{init}}$ (for $i = 0$). Furthermore, we define for every $i \geq 1$ a strategy $\mathcal{F}^{\text{restart}}_i$ to handle the restart critical positions of $\mathcal{F}_i$.

For $0 \leq i \leq n^{\mathsf{k}}m^{\mathsf{k}} - 2$ and $t \in [\mathsf{k}]$ we let $\mathfrak{q} = \alpha^{-1}(i) = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ and $\mathfrak{q}_t := (\mathfrak{a}, \mathfrak{b}, \{t\})$ and define $\mathcal{G}^{\text{restart}}_{i,t}$ to be the combination of the following strategies on the gadgets.

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathfrak{q}_t}$ on the initialization gadget (L. 3.18).
- The winning strategy with boundary $h^x_{\mathfrak{q}_t}$ and $h^y_{\mathfrak{q}_{\text{inv}}}$ on the increment gadgets. Note that, since $\mathfrak{q}_t$ is invalid, no increment gadget is applicable to $\mathfrak{q}_t$.
- The restart winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_{\text{inv}}}$ on the switches following the increment gadgets. Again, $\mathfrak{q}_{\text{inv}}$ is the invalid configuration

at the output of the preceding increment gadget.

- The output winning strategy $\mathcal{H}^{\text{out}}_{\mathfrak{q}_t}$ on the single switch.

Finally, we let $\mathcal{G}^{\text{restart}}_i := \bigcup_{i\in[\mathsf{k}]} \mathcal{G}^{\text{restart}}_{i,t}$. Note that by Lemma 3.16.(iv) every restart critical position of $\mathcal{G}_i$ is contained in $\mathcal{G}^{\text{restart}}_i$ and every restart critical position of $\mathcal{G}^{\text{init}}$ is contained in $\mathcal{G}^{\text{restart}}_0$. Now we define for $1 \leq i \leq n^{\mathsf{k}}m^{\mathsf{k}}-2$, $t \in [\mathsf{k}]$, $\mathfrak{q} = \alpha^{-1}(i) = (\mathfrak{a}, \mathfrak{b}, \emptyset)$ and $\mathfrak{q}_t := (\mathfrak{a}, \mathfrak{b}, \{t\})$ the strategy $\mathcal{F}^{\text{restart}}_{i,t}$ analogously. It consists of the following partial strategies.

- The critical strategy $\mathcal{I}^{\text{init}}_{\mathbf{0}}$ on the initialization gadget (L. 3.18).
- The winning strategy with boundary $h^x_{\mathbf{0}}$ and $h^y_{\mathbf{0}}$ on the increment gadgets.
- The restart winning strategy $\mathcal{H}^{\text{restart}}_{\mathbf{0}}$ on the switches following the increment gadgets.
- The restart winning strategy $\mathcal{H}^{\text{restart}}_{\mathfrak{q}_t}$ on the single switch.

In the end we let $\mathcal{F}^{\text{restart}}_i$ be the union of all $\mathcal{F}^{\text{restart}}_{i,t}$. Note that every restart critical position of $\mathcal{F}_i$ is contained as non-critical position in $\mathcal{F}^{\text{restart}}_i$. Finally, we let

$$\mathcal{G}^{\text{start}} := \mathcal{G}^{\text{init}} \cup \bigcup_{0\leq i\leq n^{\mathsf{k}}m^{\mathsf{k}}-2} \mathcal{G}^{\text{restart}}_i \cup \bigcup_{1\leq i\leq n^{\mathsf{k}}m^{\mathsf{k}}-2} \mathcal{F}^{\text{restart}}_i.$$

To conclude the proof note that the critical positions of $\mathcal{G}^{\text{restart}}_i$ and $\mathcal{F}^{\text{restart}}_i$ are inside the initialization gadget and hence contained in $\widehat{\mathcal{G}}^{\text{init}}$. Thus they are not critical positions of $\mathcal{G}^{\text{start}}$. It follows that $\text{crit}(\mathcal{G}^{\text{start}}) = \text{crit}(\mathcal{G}^{\text{init}}) \subseteq \widehat{\mathcal{F}}_1$. □

4. Computational Complexity of Local Consistency

The section is devoted to the lower bound on the time complexity for k-consistency tests (Theorem 2). We prove this result by a reduction from the pebble game of Kasai, Adachi and Iwata (introduced in Section 4.1 below) to the existential pebble game. The reduction is stated in our Main Lemma (Lemma 4.4) which is used to prove Theorem 2 in Section 4.2. We present the reduction in Section 4.3 and finally prove the Main Lemma in Section 4.4.

4.1. Games that Characterize Complexity Classes

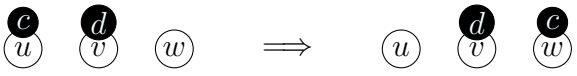


Figure 4.1.: KAI-game: Moving pebble c according to rule (u, v, w, c, d).

Kasai, Adachi and Iwata [KAI79] introduced a simple combinatorial pebble game that nicely simulates Turing machines. The authors showed in [KAI79] and [AIK84] that playing various variants of this game is complete for different complexity classes. Here we stick to the k-pebble variant, that is restricted to a fixed set of $[\mathsf{k}] = \{1, \ldots, \mathsf{k}\}$ of pebbles. An instance of the k-pebble KAI-game is a tuple $(X, R, \mathfrak{s}, \gamma)$, where X is the set of nodes, $R = R' \times \{(c, d) \in [\mathsf{k}]^2 \mid c \neq d\}$ with $R' \subseteq [X]^3$ the set of rules, $\mathfrak{s}\colon [\mathsf{k}] \to X$ the start position and $\gamma \in X$

the goal. A rule is of the form (u, v, w, c, d), with distinct pebbles c, d, pairwise distinct nodes u, v, w and the intended meaning that if pebble c is on u and pebble d is on v and there is no pebble on w then one player can move pebble c from u to w (see Figure 4.1). This is a slightly more wasteful notion than the original one used in [KAI79], where the relation $R' \subseteq X^3$ (instead of $R' \times \{(c, d) \in [\mathsf{k}]^2 \mid c \neq d\}$) is given as input. However, this technical modification does not affect the purpose of the game and increases the size of an instance only by a constant factor if k is fixed, and by a polynomial factor if k is part of the input. A *position* of the KAI-game is an injective mapping $\mathfrak{p}\colon [\mathsf{k}] \to X$. A rule $r = (u, v, w, c, d) \in R$ is *applicable* to a position $\mathfrak{p}$ if $\mathfrak{p}(c) = u$, $\mathfrak{p}(d) = v$ and $\mathfrak{p}(z) \neq w$ for all $z \in [\mathsf{k}]$. Furthermore, if r is applicable to $\mathfrak{p}$ then $r(\mathfrak{p})$ denotes the position defined as $r(\mathfrak{p})(c) = w$ and $r(\mathfrak{p})(z) = \mathfrak{p}(z)$, for all $z \in [\mathsf{k}] \setminus \{c\}$. The set of all rules in R applicable to a position $\mathfrak{p}$ is denoted by $\mathrm{appl}(\mathfrak{p})$, and $T_r(\mathfrak{p}) \subseteq [\mathsf{k}]$ is the set of KAI-pebbles i such that $\mathfrak{p}(i)$ contradicts the applicability condition of rule r: $T_{(u,v,w,c,d)}(\mathfrak{p}) := \{i \in [\mathsf{k}] \mid (i = c \text{ and } \mathfrak{p}(i) \neq u) \text{ or } (i = d \text{ and } \mathfrak{p}(i) \neq v) \text{ or } \mathfrak{p}(i) = w\}$. Thus, $r \in \mathrm{appl}(\mathfrak{p})$ iff $T_r(\mathfrak{p}) = \emptyset$.

The k-pebble KAI-game is played by two players and proceeds in rounds. In the first round Player 1 starts with position $\mathfrak{s}$ and chooses a rule $r \in \mathrm{appl}(\mathfrak{s})$. The new position is $\mathfrak{p} = r(\mathfrak{s})$. In the next round Player 2 chooses a rule $r \in \mathrm{appl}(\mathfrak{p})$ and applies it to $\mathfrak{p}$. Then it is Player 1's turn and so on. Player 1 wins the game if he reaches a position $\mathfrak{p}$, where $\mathfrak{p}(z) = \gamma$ for one $z \in [\mathsf{k}]$ or where Player 2 is unable to move. Player 2 wins if she has a strategy ensuring that Player 1 cannot reach such a position. The next definition formalizes winning strategies for Player 2. They contain sets of positions $\mathcal{K}_i, i \in \{1, 2\}$ where it is Player i's turn and a mapping κ that tells Player 2 for every position which rule to choose next.

Definition 4.1. A *winning strategy* for Player 2 in the KAI-game on $(X, \{r_1, \ldots, r_m\}, \mathfrak{s}, \gamma)$ is a triple $\mathcal{K} = (\mathcal{K}_1, \mathcal{K}_2, \kappa)$ where $\mathcal{K}_1 \subseteq \{\mathfrak{p} \mid \mathfrak{p}\colon [\mathsf{k}] \to X\}$ and $\mathcal{K}_2 \subseteq \{\mathfrak{p} \mid \mathfrak{p}\colon [\mathsf{k}] \to X \setminus \{\gamma\}\}$ are sets of positions and $\kappa\colon \mathcal{K}_2 \to [m]$ is a mapping such that the following holds:

- $\mathfrak{s} \in \mathcal{K}_1$.

- For every $\mathfrak{p} \in \mathcal{K}_1$ and every $r_i \in \mathrm{appl}(\mathfrak{p})$: $r_i(\mathfrak{p}) \in \mathcal{K}_2$.
- For every $\mathfrak{p} \in \mathcal{K}_2$: $r_{\kappa(\mathfrak{p})} \in \mathrm{appl}(\mathfrak{p})$ and $r_{\kappa(\mathfrak{p})}(\mathfrak{p}) \in \mathcal{K}_1$.

Kasai, Adachi and Iwata [KAI79] showed that the problem of determining the winner of the k-pebble KAI-game is PTIME-complete (for every fixed $k \geq 3$) under LOGSPACE-reductions and complete for EXPTIME if k is part of the input. Furthermore, they proved the following unconditional lower bound.

Theorem 4.2 ([AIK84])**.** *For every fixed* $\mathsf{k} \geq 6$ *and any* $\varepsilon > 0$*, the winner of the* k*-pebble KAI-game on a given instance* I *cannot be determined in time* $O(\|I\|^{\frac{\mathsf{k}-1}{4}-\varepsilon})$ *on deterministic multi-tape Turing machines, where* $\|I\|$ *is the size of the input* I*.*

The proof of this theorem essentially relies on the deterministic time hierarchy theorem, which states that multi-tape Turing machines of running time n^k cannot be simulated within time $n^{k-\varepsilon}$. On the other hand, Turing machines of running time n^k can be simulated within the $(4k+1)$-pebble KAI-game and hence the lower bound follows. In terms of parametrized complexity, their argument also leads to XP-completeness of the k-pebble KAI-game with parameter k as pointed out in [DF99].

We write KAI-game instead of k-pebble KAI-game if the number of pebbles k is part of the input. We need this variant in Part II where we also consider the following acyclic version. The underlying directed graph of a KAI-game instance $I = (X, R, \mathfrak{s}, \gamma)$ has vertex set X and contains an arc (u, w) if $(u, v, w, c, d) \in R$ for some v, c, d. That is, there is an arc pointing from u to w if there is some rule allowing to move a pebble from u to w. An instance of the KAI-game is *acyclic* if the underlying directed graph is acyclic. The *acyclic* KAI-game is the KAI-game where the input is required to be acyclic. Note that every play of the game on acyclic instances lasts at most $|X|$ rounds, because no node can be visited twice. This constitutes the main difference to the non-acyclic version where an exponential number of rounds is possible. The next theorem establishes the complexity of these two versions.

Theorem 4.3 ([KAI79])**.** *Determining the winner in the KAI-game is complete for* EXPTIME *and determining the winner in the acyclic KAI-game is complete for* PSPACE*.*

4.2. Proof of the Lower Bound by a Reduction

Now we state our Main Lemma and prove Theorem 2. The proof of the Main Lemma is deferred to Section 4.4.

Lemma 4.4 (Main Lemma)**.** *There is a reduction from the* k*-pebble KAI-game to the existential* $(\mathsf{k}+1)$*-pebble game that computes for every instance* $I = (X, R, \mathfrak{s}, \gamma)$ *two vertex-colored simple graphs* G_S *and* G_D *such that the following constraints hold:*

- *Player 1 has a winning strategy in the* k*-pebble KAI-game on* I *if and only if Spoiler has a winning strategy in the existential* $(\mathsf{k}+1)$*-pebble game on* G_S *and* G_D*.*
- $|V(G_S)| + |V(G_D)| = O(|X| \cdot |R| \cdot \mathsf{k}^2)$*.*
- $|E(G_S)| + |E(G_D)| = O(\mathsf{k}^4(|X|^2|R| + |X| \cdot |R|^2))$*.*
- *The reduction is computable in time* $(O(\|I\|^3)$ *and in logarithmic space.*

Lemma 4.4 implies the result of Kolaitis and Panttaja [KP03] that k-Cons is EXPTIME-complete when k is part of the input (Theorem 2.11). Furthermore, it follows that k-Cons is PTIME-complete under LOGSPACE-reductions for every fixed $k \geq 4$ (as shown in [KP03]). Since the reduction is also an fpt-reduction, it follows that determining the winner in the existential k-pebble game is complete for the parameterized complexity class XP (Corollary 2.12). Now we are ready to proof Theorem 2 which is restated below.

Reminder of Theorem 2. *For every fixed* $k \geq 15$ *and any* $\varepsilon > 0$*, the winner of the existential* k*-pebble game on two given binary*

structures A *and* B *cannot be determined in time* $O((\|\mathsf{A}\|+\|\mathsf{B}\|)^{\frac{k-2}{12}-\varepsilon})$ *on deterministic multi-tape Turing machines.*

Proof of Theorem 2. Let $k \geq 15$ be a fixed integer and $\varepsilon > 0$. Assume that $\mathbb{A}$ is an algorithm that determines the winner of the existential k-pebble game on structures A and B in time $O((\|\mathsf{A}\| + \|\mathsf{B}\|)^{\frac{k-2}{12}-\varepsilon})$. Let $\mathbb{B}$ be the algorithm that first applies the reduction from Lemma 4.4 to a given instance I of the $(k-1)$-pebble KAI-game and then executes $\mathbb{A}$. Since $\|G_S\| + \|G_D\| = O(\|I\|^3)$, $\mathbb{B}$ has running time $O(\|I\|^3 + \|I\|^{3\left(\frac{k-2}{12}-\varepsilon\right)})$, and thus solves the k'-pebble KAI-game in time $O(\|I\|^{\frac{k'-1}{4}-\varepsilon'})$ for $k' = k-1$ and $\varepsilon' > 0$. This contradicts Theorem 4.2. □

4.3. The Reduction

Let $([n], R, \mathfrak{s}, \gamma)$ be an instance of the k-pebble KAI-game and $m := |R|$. We use the variables n and m for historical reasons (to be consistent with [Ber12a; Ber13]) and keep them fixed during this section. However, n and m have been used differently in Section 3.3 and 3.4 (to index vertices in the structures A_n and B_m). To avoid ambiguity we denote the old n, m by n_{old}, m_{old} from now on. Whenever we use a gadget from Section 3.4, we set $n_{\text{old}} = 1$ and $m_{\text{old}} = n$.

As in [KP03], the main idea is to simulate every play of the KAI-game within the existential pebble game such that Spoiler imitates the moves of Player 1 and Duplicator imitates the moves of Player 2. We construct two vertex-colored simple graphs, G_S and G_D. We use $|V(G_S)|$ colors to color every vertex of Spoiler's graph G_S differently and partition the vertices of Duplicator's graph with these colors. Thus, whenever Spoiler pebbles a vertex in G_S there is a corresponding set of vertices in G_D Duplicator can pebble.

To encode a position of the KAI-game in the existential $(\mathsf{k}+1)$-pebble game we introduce the vertices $\{\mathsf{x}^1, \ldots, \mathsf{x}^{\mathsf{k}}\}$ in Spoiler's graph and $\{\mathsf{x}^i_l \mid i \in [\mathsf{k}], 0 \leq l \leq n\}$ in Duplicator's graph. For each i, all the vertices $\{\mathsf{x}^i\} \cup \{\mathsf{x}^i_0, \ldots, \mathsf{x}^i_n\}$ are colored with the same unique color, denoted by c_{x^i}. The vertices x^i_0 play a special role in the construction,

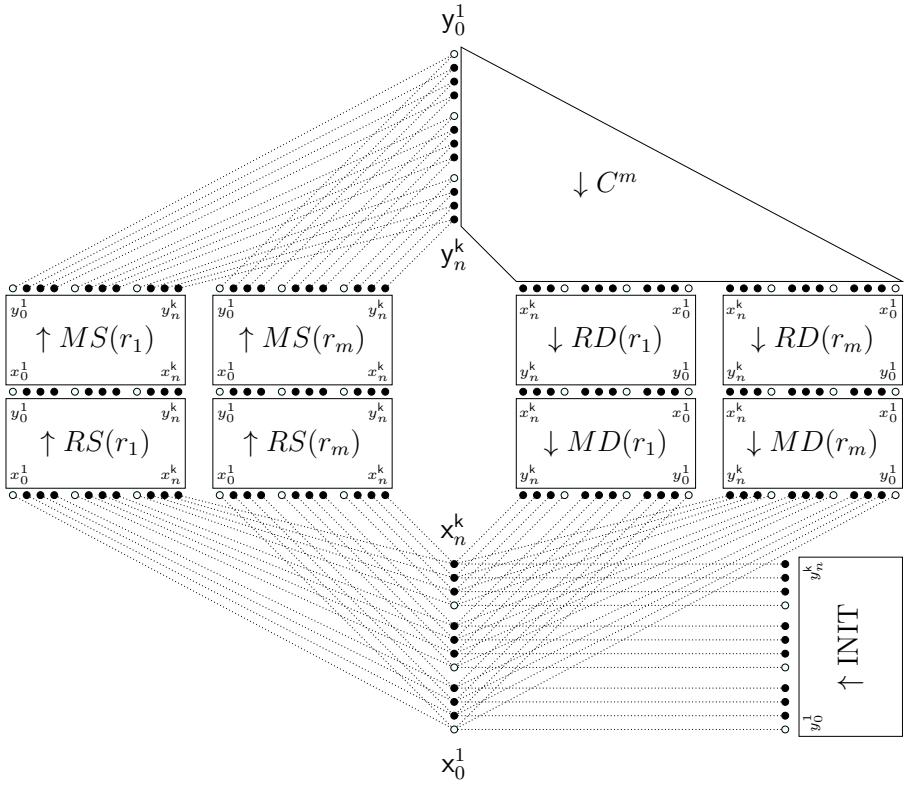


Figure 4.2.: The graph G_D. The dotted edges are only for visual purposes and need to be contracted. The gadgets $MS(r_i)$ and $MD(r_i)$ are distinct copies of the switch. The special vertices x_j^i and y_j^i are also depicted in Figure 4.3.

so we draw ∘ for vertices with subindex 0 in the figures and • for all other vertices. Furthermore, we introduce vertices y^i,y^i_l in the same way. Note that these vertex blocks have exactly the same shape as in the previous section (cf. Figure 3.2) for $n_{\text{old}} = 1$ and $m_{\text{old}} = n$. The only (notational) difference is that we can drop the lower indices in G_S.

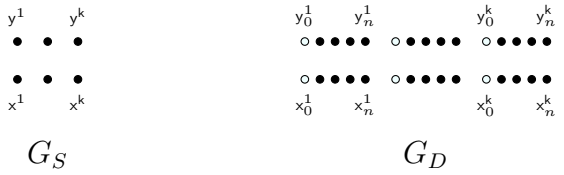


Figure 4.3.: Vertex blocks to encode positions in the KAI-game.

If $\mathfrak{p}\colon [\mathsf{k}] \to [n]$ is a position of the k pebbles in the KAI-game and it is Player 1's turn, then $\{(\mathsf{x}^i, \mathsf{x}^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ is the corresponding position in the existential pebble game. If it is Player 2's turn, then $\{(\mathsf{y}^i, \mathsf{y}^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ is the corresponding position. During the course of the game Spoiler pebbles some vertex x^i asking, "Where does KAI-pebble i lie?" Due to the coloring Duplicator has to answer with some vertex x^i_l meaning "KAI-pebble i lies on node l." The vertices ∘ are used to handle the case when Spoiler does not play in the intended way, that is, Spoiler has a winning strategy if and only if he has a winning strategy on the • vertices. In order to name positions that include ∘ vertices, we define for positions $\mathfrak{p}$ and sets $T \subseteq [\mathsf{k}]$ the mapping $(\mathfrak{p}, T)$ as

$$(\mathfrak{p}, T)(i) = \begin{cases} 0, & i \in T, \\ \mathfrak{p}(i), & \text{otherwise}, \end{cases}$$

and write $\mathfrak{p}$ for $(\mathfrak{p}, \emptyset)$ and $\mathbf{0}$ for $(\mathfrak{p}, [\mathsf{k}])$. The KAI-game positions $\mathfrak{p}$ and the mappings $(\mathfrak{p}, T)$ play the same role as the valid and invalid configurations introduced in Section 3.3. Indeed, for every position $\mathfrak{p}$, "$\mathfrak{p}$ on x" denotes the same pebble position as "$\mathfrak{q}$ on x", where

$\mathfrak{q} = (\mathfrak{a}^{\mathrm{id}}, \mathfrak{p}, \emptyset)$ is a valid configuration and $\mathfrak{a}^{\mathrm{id}} = \{(i,i) \mid i \in [\mathsf{k}]\}$. Moreover, the mapping $(\mathfrak{p}, T)$ corresponds to $(\mathfrak{a}^{\mathrm{id}}, \mathfrak{p}, T)$ and we will use them interchangeably.

Now we have to introduce gadgets to ensure that Spoiler can simulate a play of the KAI-game. That is, if Player 1 can reach a position $\mathfrak{p}$ in the KAI-game, then Spoiler can reach the encoded position on the x- or y-vertices. The following list of properties ensures this.

- For the start position $\mathfrak{s}$, Spoiler can reach $\{(\mathsf{x}^i, \mathsf{x}^i_{\mathfrak{s}(i)}) \mid i \in [\mathsf{k}]\}$ from $\emptyset$.
- For a position $\mathfrak{p}$ and every rule $r \in \mathrm{appl}(\mathfrak{p})$, Spoiler can reach $\{(\mathsf{y}^i, \mathsf{y}^i_{r(\mathfrak{p})(i)}) \mid i \in [\mathsf{k}]\}$ from $\{(\mathsf{x}^i, \mathsf{x}^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$.
- Spoiler can reach $\{(\mathsf{x}^i, \mathsf{x}^i_{r(\mathfrak{p})(i)}) \mid i \in [\mathsf{k}]\}$ from $\{(\mathsf{y}^i, \mathsf{y}^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ for a rule $r \in \mathrm{appl}(\mathfrak{p})$ of Duplicator's choice.
- $\{(\mathsf{y}^i, \mathsf{y}^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ is not a partial homomorphism if $\mathfrak{p}(i) = \gamma$ for some $i \in [\mathsf{k}]$.

It follows from these properties that if Player 1 has a winning strategy in the KAI-game, then Spoiler wins the existential pebble game by simulating Player 1's winning strategy. The difficult task is to prove that this is the only way for Spoiler to win. We give a brief description of the construction and argue how Spoiler is intended to play on it. Duplicator's graph is illustrated in Figure 4.2. The gadgets are glued together at their boundary vertices and the vertex blocks that are glued together inherit their colors. In order to make sure that the colors partition the graphs we define a new color for every combination of colors occurring at one vertex in the graph. In Spoiler's graph we proceed the same way with Spoiler's side of the gadgets.

To implement the last condition, we simply delete the color c_{y^i} from y^i_γ for all $i \in [\mathsf{k}]$. Since y^i and all y^i_l, $l \in [n] \setminus \{\gamma\}$, are still colored c_{y^i}, it follows that the mapping $\mathsf{y}^i \mapsto \mathsf{y}^i_l$ that encodes "KAI-pebble i lies on node l" is a partial homomorphism if and only if l is not the goal node γ. It follows that Spoiler wins the game if he can reach the

position $\{(\mathsf{y}^i, \mathsf{y}^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ where $\mathfrak{p}$ is a winning position for Player 1 in the KAI-game.

To make sure that Spoiler can reach the start position, we use the initialization gadget $\mathrm{INIT}^{\mathfrak{q}}$ (defined in Section 3.4), whose boundary $y^1, \ldots, y^{\mathsf{k}}$ in Spoiler's graph and $y^1_0, \ldots, y^{\mathsf{k}}_n$ in Duplicator's graph is identified with vertices $\mathsf{x}^1, \ldots, \mathsf{x}^{\mathsf{k}}$ in Spoiler's graph and $\mathsf{x}^1_0, \ldots, \mathsf{x}^{\mathsf{k}}_n$ in Duplicator's graph. Recall that the initialization gadget depends on the parameters $n_{\text{old}}, m_{\text{old}}$ and a certain start configuration $\mathfrak{q}$. Here we need the gadget with $n_{\text{old}} = 1$, $m_{\text{old}} = n$ and $\mathfrak{q} = (\mathfrak{a}^{\text{id}}, \mathfrak{s}, \emptyset)$, where $\mathfrak{a}^{\text{id}}$ is the identity $\{(i,i) \mid i \in [\mathsf{k}]\}$ and $\mathfrak{s}$ the start position in the KAI-game.

The boundary vertices of the other gadgets have a similar form and can be divided into input vertices x (with certain indices) and output vertices y that are colored in the same way as the x- and y-vertices. As above, a position $\mathfrak{p}$ in the KAI-game is encoded as $\{(x^i, x^i_{\mathfrak{p}}) \mid i \in [\mathsf{k}]\}$ on these vertex blocks and we call it $\mathfrak{p}$ *on* x. The direction of the gadgets is indicated in Figure 4.2 by arrows. Thus, the players are intended to move clockwise in the graph.

For each rule r we define different rule gadgets $RS(r)$ and $RD(r)$ in which Spoiler can reach the position $r(\mathfrak{p})$ on the output y from $\mathfrak{p}$ on the input x if r is applicable to $\mathfrak{p}$. Hence, from a position $\mathfrak{p}$ on x Spoiler can choose an applicable rule r and reach $r(\mathfrak{p})$ on the output y of some rule gadget $RS(r)$. The rule gadgets play a similar role as the increment gadgets in Chapter 3. The choice gadget C^m enables Duplicator to choose one of the m rules she wants to apply. That is, Duplicator can choose a rule r such that from $\mathfrak{p}$ on y Spoiler can reach $\mathfrak{p}$ on the input of $RD(r)$ and then $r(\mathfrak{p})$ on the output of $RD(r)$. We put one copy of the switch (see Section 3.4, again with parameters $n_{\text{old}} = 1$ and $m_{\text{old}} = n$) at the output vertices of every rule gadget. Spoiler's strategy on this gadget is the same as before: he can pebble a position through the switch, that is, he can reach $\mathfrak{p}$ on the output from $\mathfrak{p}$ on the input. This concludes the description of how the gadgets can be used by Spoiler to ensure the four properties above.

Duplicator's strategy is to force Spoiler to play exactly this way. Especially, if the KAI-game does not stop, then Duplicator can play

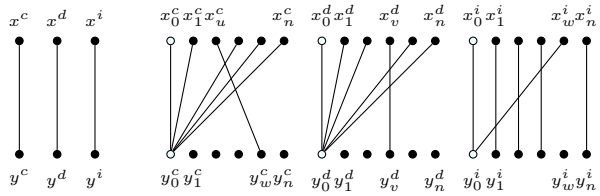


Figure 4.4.: Rule gadget $RS(u, v, w, c, d)$. The range of i is $[\mathsf{k}] \setminus \{c, d\}$.

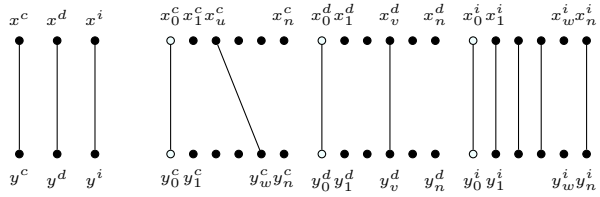


Figure 4.5.: Rule gadget $RD(u, v, w, c, d)$. The range of i is $[\mathsf{k}] \setminus \{c, d\}$.

the existential pebble game forever by forcing Spoiler to simulate this infinite play. The main tool for Duplicator is to answer with $\circ$ vertices whenever Spoiler plays incorrectly: if Spoiler pebbles a vertex x^i he is not supposed to pebble now, then Duplicator answers with x_0^i. The strategies on the gadgets ensure that such positions extend to partial homomorphisms and thus Spoiler does not benefit from them.

Rule Gadgets

The rule gadgets $RS(r)$ and $RD(r)$ consist of input vertices $x^1, \ldots, x^{\mathsf{k}}$ in Spoiler's graph and $x_0^1, \ldots, x_n^{\mathsf{k}}$ in Duplicator's graph, and output vertices $y^1, \ldots, y^{\mathsf{k}}$ and $y_0^1, \ldots, y_k^n$. For each rule $r = (u, v, w, c, d)$ we connect the vertices in the gadgets $RS(r)$ and $RD(r)$ as shown in Figure 4.4 and 4.5.

If r is applicable to $\mathfrak{p}$, then Spoiler can reach the pebble position $\{(y^i, y^i_{r(\mathfrak{p})(i)}) \mid i \in [\mathsf{k}]\}$ from $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ in both gadgets. To do so, he picks up the remaining pebble and puts it on y^1. Then he picks up the pebble from x^1 and puts it on y^2 and so on. This fact is stated in Lemma 4.5(i) and 4.6(i). Assume that r is not applicable

to $\mathfrak{p}$ (then $T_r(\mathfrak{p}) \neq \emptyset$) and the current position is $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$. On $RS(r)$ Duplicator can pebble some $\circ$ vertex y^i_0 when Spoiler asks for y^i ($i \in T_r(\mathfrak{p})$), and thus can avoid valid positions on the y-vertices. Therefore, Spoiler is penalized when he chooses a rule r not applicable to $\mathfrak{p}$ and plays on $RS(r)$. This strategy is stated in Lemma 4.5(ii) for $T = \emptyset$. If Duplicator chooses a rule r not applicable to $\mathfrak{p}$ and plays on $RD(r)$, then she will be penalized, because Spoiler wins immediately from position $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ (Lemma 4.6(iii)) by pebbling on y^i for some $i \in T_r(\mathfrak{p})$. Furthermore, Lemma 4.5(ii) and 4.6(ii) state that for invalid positions on the x-vertices (i.e. $T \neq \emptyset$), Duplicator can avoid valid positions on the y-vertices.

Lemma 4.5. *For every rule $r = (u, v, w, c, d)$ and position $\mathfrak{p}\colon [\mathsf{k}] \to [n]$ the following holds in the existential $(\mathsf{k}+1)$-pebble game on $RS(r)$:*

(i) If $r \in \mathrm{appl}(\mathfrak{p})$, then Spoiler can reach $\{(y^i, y^i_{r(\mathfrak{p})(i)}) \mid i \in [\mathsf{k}]\}$ from $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$.

(ii) Duplicator has a winning strategy $\mathcal{R}_{(\mathfrak{p},T)}$ with boundary function $\{(x^i, x^i_{(\mathfrak{p},T)(i)}) \mid i \in [\mathsf{k}]\} \cup \{(y^i, y^i_{(r(\mathfrak{p}),T\cup T_r(\mathfrak{p}))(i)}) \mid i \in [\mathsf{k}]\}$, for all $T \subseteq [\mathsf{k}]$.

Proof. For $i = 1, \ldots, \mathsf{k}$ Spoiler takes the remaining pebble and puts it on y^i. Since there is an edge $\{x^i, y^i\}$, Duplicator has to answer with $y^i_{r(\mathfrak{p})(i)}$ because this is the only vertex adjacent to $x^i_{\mathfrak{p}(i)}$. In the next step Spoiler picks up the pebble pair from $x^i, x^i_{\mathfrak{p}}$ and proceeds with $i + 1$.

The boundary function β defined in (ii) preserves the vertex colors and maps edges $\{x^i, y^i\}$ to edges $\{x^i_{(\mathfrak{p},T)(i)}, y^i_{(r(\mathfrak{p}),T\cup T_r(\mathfrak{p}))(i)}\}$, hence defines a total homomorphism on $RS(r)$. It follows that $\mathcal{R}_{(\mathfrak{p},T)} := \{h \mid h \subseteq \beta\}$ is a winning strategy. □

Lemma 4.6. *For every rule $r = (u, v, w, c, d)$ and position $\mathfrak{p}\colon [\mathsf{k}] \to [n]$ the following holds in the existential $(\mathsf{k}+1)$-pebble game on $RD(r)$:*

(i) If $r \in \mathrm{appl}(\mathfrak{p})$, then Spoiler can reach $\{(y^i, y^i_{r(\mathfrak{p})(i)}) \mid i \in [\mathsf{k}]\}$ from $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$.

(ii) *If* $r \in \mathrm{appl}(\mathfrak{p})$*, then Duplicator has a winning strategy* $\mathcal{R}_{(\mathfrak{p},T)}$ *with boundary function* $\{(x^i, x^i_{(\mathfrak{p},T)(i)}) \mid i \in [\mathsf{k}]\} \cup \{(y^i, y^i_{(r(\mathfrak{p}),T)(i)}) \mid i \in [\mathsf{k}]\}$*, for all* $T \subseteq [\mathsf{k}]$.

(iii) *If* $r \notin \mathrm{appl}(\mathfrak{p})$*, then Spoiler wins from* $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$.

Proof. Statement (i) is similar to Lemma 4.5(i). The boundary function β stated in (ii) defines a total homomorphism on $RD(r)$. Thus, $\mathcal{R}_{(\mathfrak{p},T)} := \{h \mid h \subseteq \beta\}$ is a winning strategy.

In order to win from $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ for a rule r not applicable to $\mathfrak{p}$ (Statement (iii)), Spoiler chooses some KAI-pebble $i \in T_r(\mathfrak{p})$ (whose position $\mathfrak{p}(i)$ witnesses that r is not applicable to $\mathfrak{p}$) and puts the remaining pebble on y^i. Since Duplicator has to answer with a vertex y^i_j of the same color and none of them is adjacent to $x^i_{\mathfrak{p}(i)}$, she loses immediately. □

The Choice Gadget

The boundary of the choice gadget consists of input vertices $x^1, \ldots, x^{\mathsf{k}}$ in Spoiler's graph and $x^1_0, \ldots, x^{\mathsf{k}}_n$ in Duplicator's graph. These vertices are identified with y-vertices in the final graph. The output vertices are of the form $(y_q)^1, \ldots, (y_q)^{\mathsf{k}}$ and $(y_q)^1_0, \ldots, (y_q)^{\mathsf{k}}_n$ for all $q \in [m]$ and are connected to the rule gadgets $RD(r_q)$ (recall that m is the number of rule gadgets). This gadget enables Spoiler to reach positions $((y_q)^i, (y_q)^i_{\mathfrak{p}(i)})$ from $(x^i, x^i_{\mathfrak{p}(i)})$ but Duplicator can choose the desired $q \in [m]$. This choice will later coincide with the rule $r_q \in R$ Player 2 chooses in the KAI-game when position $\mathfrak{p}$ is pebbled. The *choice gadget* C^m is defined as follows:

$$
\begin{aligned}
V(C^m_S) &= \{x^i, a^i \mid i \in [\mathsf{k}]\} \cup \{(y_q)^i \mid i \in [\mathsf{k}], q \in [m]\},\\
E(C^m_S) &= \big\{\{x^i, a^i\} \mid i \in [\mathsf{k}]\big\} \cup\\
&\qquad \big\{\{a^i, (y_q)^i\} \mid i \in [\mathsf{k}], q \in [m]\big\}\\
&\qquad \cup \big\{\{a^i, a^j\} \mid i, j \in [\mathsf{k}];\, i \neq j\big\},\\
V(C^m_D) &= \{x^i_l \mid i \in [\mathsf{k}], 0 \leq l \leq n\} \cup \{a^i_0 \mid i \in [\mathsf{k}]\}\\
&\qquad \{a^i_{l,q} \mid i \in [\mathsf{k}], l \in [n], q \in [m]\}
\end{aligned}
$$

$$
\begin{aligned}
&\cup \{(y_q)^i_l \mid i \in [\mathsf{k}], q \in [m], 0 \leq l \leq n\},\\
E(C^m_D) = &\big\{\{x^i_0, a^i_0\} \mid i \in [\mathsf{k}]\big\}\\
&\cup \big\{\{x^i_l, a^i_{l,q}\} \mid i \in [\mathsf{k}], l \in [n], q \in [m]\big\}\\
&\cup \big\{\{a^i_0, (y_q)^i_0\} \mid i \in [\mathsf{k}], q \in [m]\big\}\\
&\cup \big\{\{a^i_{l,q}, (y_q)^i_l\} \mid i \in [\mathsf{k}]; l \in [n]; q \in [m]\big\}\\
&\cup \big\{\{a^i_{l,q}, (y_{q'})^i_0\} \mid i \in [\mathsf{k}]; l \in [n]; q, q' \in [m]; q \neq q'\big\}\\
&\cup \big\{\{a^i_{l,q}, a^j_{l',q}\} \mid i, j \in [\mathsf{k}]; i \neq j; l, l' \in [n]; q \in [m]\big\}.
\end{aligned}
$$

Furthermore, for all $i \in [\mathsf{k}]$, all vertices a^i and $a^i_{l,q}$ are colored with the unique color c_{a^i}, and for all $i \in [\mathsf{k}]$, $q \in [m]$ the vertices $(y_q)^i$ and $(y_q)^i_l$ are colored with the unique color $c_{(y_q)^i}$. One partition of C^m is shown in Figure 4.6. Recall that Spoiler can reach one of the

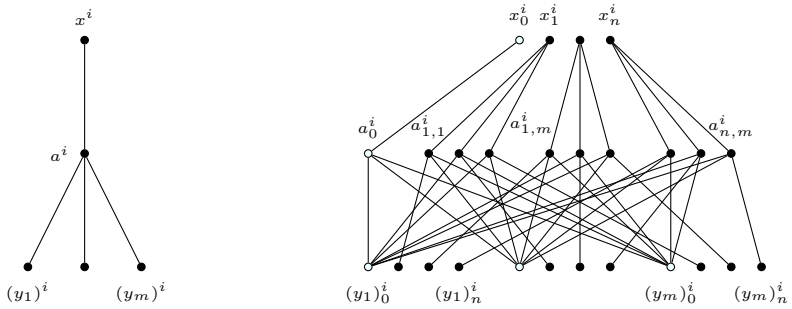


Figure 4.6.: The i-th partition of the gadget C^m.

positions $p_1, \ldots, p_m$ from p_0 if starting from position p_0 either Spoiler wins the game or one of the positions $p_1, \ldots, p_m$ occurs after a finite number of rounds.

Lemma 4.7. *In the existential* $(\mathsf{k}+1)$*-pebble game on* C^m*,*

(i) for every $\mathfrak{p}\colon [\mathsf{k}] \to [n]$ *Spoiler can reach one of the positions* $\{((y_l)^i, (y_l)^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ *for* $l \in [m]$ *from* $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$*, and*

(ii) *for every* $l \in [m]$, $\mathfrak{p}\colon [\mathsf{k}] \to [n]$, $T \subseteq [\mathsf{k}]$, *Duplicator has a winning strategy* $\mathcal{C}^l_{(\mathfrak{p},T)}$ *with boundary function* $\{(x^i, x^i_{(\mathfrak{p},T)(i)}) \mid i \in [\mathsf{k}]\} \cup \{((y_q)^i, (y_q)^i_0) \mid i \in [\mathsf{k}], q \in [m] \setminus \{l\}\} \cup \{((y_l)^i, (y_l)^i_{(\mathfrak{p},T)(i)}) \mid i \in [\mathsf{k}]\}$.

Proof. We first present Spoiler's strategy (i). Starting from position $\{(x^i, x^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ Spoiler puts the $(\mathsf{k}+1)$st pebble on a^1 and Duplicator has to respond with $a^1_{\mathfrak{p}(1),l}$ for one $l \in [m]$ she can choose. Then Spoiler picks up the pebble from x^1 and puts it on a^2. Duplicator has to respond with $a^2_{\mathfrak{p}(2),l}$, since this vertex is the only one adjacent to $x^2_{\mathfrak{p}(2)}$ and $a^1_{\mathfrak{p}(1),l}$. Next, Spoiler picks up the pebble from x^2 and puts it on a^3 and so on. Thus, Spoiler reaches $\{(a^i, a^i_{\mathfrak{p}(i),l}) \mid i \in [\mathsf{k}]\}$. In the next step he places the $(\mathsf{k}+1)$st pebble on $(y_l)^1$, and Duplicator has to answer with $(y_l)^1_{\mathfrak{p}(1)}$, since this vertex is the only one colored in the same color as $(y_l)^1$ and adjacent to $a^1_{\mathfrak{p}(1),l}$. Now Spoiler picks up the pebble from a^1 and puts it on $(y_l)^2$, Duplicator has to answer with $(y_l)^2_{\mathfrak{p}(2)}$. Following that strategy Spoiler can reach $\{((y_l)^i, (y_l)^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$. Duplicator's strategy (ii) is simply defined as $\mathcal{C}^l_{(\mathfrak{p},T)} = \wp(h^l_{(\mathfrak{p},T)})$, where $h^l_{(\mathfrak{p},T)}$ is the following total homomorphism on C^m:

$$h^l_{(\mathfrak{p},T)}(z) := \begin{cases} x^i_{(\mathfrak{p},T)(i)}, & \text{if } z = x^i, \\ a^i_{\mathfrak{p}(i),l}, & \text{if } z = a^i, i \notin T, \\ a^i_0, & \text{if } z = a^i, i \in T, \\ (y_l)^i_{(\mathfrak{p},T)(i)}, & \text{if } z = (y_l)^i, \\ (y_q)^i_0, & \text{if } z = (y_q)^i,\ q \in [m] \setminus \{l\}. \end{cases}$$

Hence, if some (not necessarily valid) position $(\mathfrak{p}, T)$ is on the input Duplicator can force Spoiler to bring this position to the output block (y_l) using the strategy $\mathcal{C}^l_{(\mathfrak{p},T)}$. □

The strategies $\mathcal{C}^1_{\mathbf{0}} = \cdots = \mathcal{C}^m_{\mathbf{0}}$ have the $\mathbf{0}$ position at every vertex block and will be denoted by $\mathcal{C}_{\mathbf{0}}$.

4.4. Correctness of the Reduction

We finish the reduction by proving that our construction satisfies the conditions of Lemma 4.4.

Proof of Lemma 4.4. It remains to construct winning strategies for Spoiler and Duplicator on the colored graphs G_S and G_D. First, we develop the winning strategy for Spoiler. Hence assume that Player 1 has a winning strategy in the k-pebble KAI-game. By Lemma 3.18, Spoiler can reach position $\{(\mathsf{x}^i, \mathsf{x}^i_{\mathfrak{s}(i)}) \mid i \in [\mathsf{k}]\}$. Let r be the first rule Player 1 chooses in the KAI-game. By Lemma 4.5 Spoiler can reach $\{(y^i, y^i_{r(\mathfrak{s})(i)}) \mid i \in [\mathsf{k}]\}$, where the y vertices are the output-boundary of the rule gadget $RS(r)$. By Lemma 3.16 he can pebble through the switch and reach $\{(\mathsf{y}^i, \mathsf{y}^i_{r(\mathfrak{s})(i)}) \mid i \in [\mathsf{k}]\}$. Let $\mathfrak{p} := r(\mathfrak{s})$. If $\mathfrak{p}$ maps some pebble to γ, then Spoiler wins the game since y^i is colored c_{y^i} whereas y^i_γ is not. Otherwise (by Lemma 4.7) Spoiler can reach position $\{((y_q)^i, (y_q)^i_{\mathfrak{p}(i)}) \mid i \in [\mathsf{k}]\}$ for one $q \in [m]$ of Duplicator's choice at the output of the choice gadget and the input of $\mathrm{RD}(r_q)$. If $r_q \notin \mathrm{appl}(\mathfrak{p})$, then Spoiler wins immediately by Lemma 4.6 (especially Spoiler wins if $\mathrm{appl}(\mathfrak{p}) = \emptyset$ and Player 2 cannot move). If $r_q \in \mathrm{appl}(\mathfrak{p})$, then Spoiler can reach $\{(\mathsf{x}^i, \mathsf{x}^i_{r_q(\mathfrak{p})(i)}) \mid i \in [\mathsf{k}]\}$. Spoiler chooses the next rule according to Player 1's winning strategy and so on. Since Player 1 eventually puts a pebble c on node γ, Spoiler can reach position $(\mathsf{y}^c, \mathsf{y}^c_\gamma)$ and thus he wins the game.

Assume that $\mathcal{K} = (\mathcal{K}_1, \mathcal{K}_2, \kappa)$ is a winning strategy for Player 2 in the k-pebble KAI-game. Recall that Player 2 can play in such a way that every position occurring in the KAI-game is either contained in $\mathcal{K}_1$ or $\mathcal{K}_2$ where $\mathcal{K}_i$ is the set of position when it is Player i's turn. We define a global critical strategy $\mathcal{S}_\mathfrak{p}$ ($\mathcal{D}_\mathfrak{p}$) for every position $\mathfrak{p}$ in $\mathcal{K}_1$ ($\mathcal{K}_2$). Duplicator can now simulate Player 2's winning strategy in the KAI-game by playing according to the critical strategy $\mathcal{S}_\mathfrak{p}$ ($\mathcal{D}_\mathfrak{p}$) if the position $\mathfrak{p}$ is the current position in the KAI-game and it is Player 1's (Player 2's) turn. If Spoiler pebbles output critical positions in these strategies, Duplicator switches the strategies in the same way as the positions in the KAI-game change. If Spoiler plays incorrectly in the sense that he pebbles a restart critical position at the switches, then

Duplicator moves to a corresponding restart strategy.

Now we construct these critical strategies for the whole graph out of smaller critical strategies $\mathcal{F}$ defined on gadgets Q (denoted $\mathcal{F}\langle Q\rangle$) using the $\uplus$-operator and Lemma 3.7. The global strategy $\mathcal{S}^{\text{init}}$ means "the KAI-game has just started, position $\mathfrak{s}$ is on the board and it is Player 1's turn." The strategy $\mathcal{S}_\mathfrak{p}$ ($\mathcal{D}_\mathfrak{p}$) denotes "position $\mathfrak{p}$ is on the board and it is Player 1's (Player 2's) turn."

$$\begin{aligned}\mathcal{S}^{\text{init}} = \mathcal{I}^{\text{init}} \uplus \mathcal{C}_\mathbf{0} \uplus \biguplus_{l\in\text{appl}(\mathfrak{s})} \left(\mathcal{R}_\mathfrak{s}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{in}}_{r_l(\mathfrak{s})}\langle MS(r_l)\rangle\right) \uplus \\ \biguplus_{l\in[m]\setminus\text{appl}(\mathfrak{s})} \left(\mathcal{R}_\mathfrak{s}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{restart}}_{(r_l(\mathfrak{s}),T_{r_l})}\langle MS(r_l)\rangle\right) \uplus \\ \biguplus_{l\in[m]} \left(\mathcal{R}_\mathbf{0}\langle RD(r_l)\rangle \uplus \mathcal{H}^{\text{out}}_\mathfrak{s}\langle MD(r_l)\rangle\right).\end{aligned}$$

We define the global critical strategies $\mathcal{S}_\mathfrak{p}$ and $\mathcal{S}^{\text{restart}}_{(\mathfrak{p},T)}$ for all $\mathfrak{p}\in\mathcal{K}_1$ and $T\neq\emptyset$. In the strategy $\mathcal{S}_\mathfrak{p}$ the position $\mathfrak{p}$ is at the $\mathtt{x}$-vertices encoding that $\mathfrak{p}$ is the current position in the KAI-game and it is Player 1's turn. The strategies $\mathcal{S}^{\text{restart}}_{(\mathfrak{p},T)}$ contain the restart critical positions of $\mathcal{S}_\mathfrak{p}$.

$$\begin{aligned}\mathcal{S}_\mathfrak{p} = \mathcal{I}^{\text{init}}_\mathfrak{p} \uplus \mathcal{C}_\mathbf{0} \uplus \biguplus_{l\in\text{appl}(\mathfrak{p})} \left(\mathcal{R}_\mathfrak{p}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{in}}_{r_l(\mathfrak{p})}\langle MS(r_l)\rangle\right) \uplus \\ \biguplus_{l\in[m]\setminus\text{appl}(\mathfrak{p})} \left(\mathcal{R}_\mathfrak{p}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{restart}}_{(r_l(\mathfrak{p}),T_{r_l})}\langle MS(r_l)\rangle\right) \uplus \\ \biguplus_{l\in[m]} \left(\mathcal{R}_\mathbf{0}\langle RD(r_l)\rangle \uplus \mathcal{H}^{\text{out}}_\mathfrak{p}\langle MD(r_l)\rangle\right),\end{aligned}$$

$$\begin{aligned}\mathcal{S}^{\text{restart}}_{(\mathfrak{p},T)} = \mathcal{I}^{\text{init}}_{(\mathfrak{p},T)} \uplus \mathcal{C}_\mathbf{0} \uplus \\ \biguplus_{l\in[m]} \left(\mathcal{R}_{(\mathfrak{p},T)}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{restart}}_{(r_l(\mathfrak{p}),T\cup T_{r_l})}\langle MS(r_l)\rangle \uplus \right. \\ \left. \mathcal{R}_0\langle RD(r_l)\rangle \uplus \mathcal{H}^{\text{out}}_{(\mathfrak{p},T)}\langle MD(r_l)\rangle\right).\end{aligned}$$

Furthermore, for all $\mathfrak{p}\in\mathcal{K}_2$ and $T\neq\emptyset$ let $\mathcal{D}_\mathfrak{p}$ and $\mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)}$ be the

following global critical strategies. Similar as in the strategies above, $\mathcal{D}_{\mathfrak{p}}$ puts the position $\mathfrak{p}$ at the y-vertices encoding that $\mathfrak{p}$ is the current position in the KAI-game and it is Player 2's turn. Again, the strategies $\mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)}$ contain the restart critical positions of $\mathcal{D}_{\mathfrak{p}}$.

$$\begin{aligned}
\mathcal{D}_{\mathfrak{p}} = \mathcal{I}^{\text{init}}_{\mathbf{0}} \uplus \mathcal{C}^{\kappa(\mathfrak{p})}_{\mathfrak{p}} \uplus \biguplus_{l\in[m]} \big(\mathcal{R}_{\mathbf{0}}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{out}}_{\mathfrak{p}}\langle MS(r_l)\rangle\big)\uplus \\
\biguplus_{l\in[m]\setminus\{\kappa(\mathfrak{p})\}} \big(\mathcal{R}_{\mathbf{0}}\langle RD(r_l)\rangle \uplus \mathcal{H}^{\text{restart}}_{\mathbf{0}}\langle MD(r_l)\rangle\big)\uplus \\
\mathcal{R}_{\mathfrak{p}}\langle RD(r_{\kappa(\mathfrak{p})})\rangle \uplus \mathcal{H}^{\text{in}}_{r_{\kappa(\mathfrak{p})}(\mathfrak{p})}\langle MD(r_{\kappa(\mathfrak{p})})\rangle, \\
\mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)} = \mathcal{I}^{\text{init}}_{\mathbf{0}} \uplus \mathcal{C}^{\kappa(\mathfrak{p})}_{(\mathfrak{p},T)} \uplus \biguplus_{l\in[m]} \big(\mathcal{R}_{\mathbf{0}}\langle RS(r_l)\rangle \uplus \mathcal{H}^{\text{out}}_{(\mathfrak{p},T)}\langle MS(r_l)\rangle\big)\uplus \\
\biguplus_{l\in[m]\setminus\{\kappa(\mathfrak{p})\}} \big(\mathcal{R}_{\mathbf{0}}\langle RD(r_l)\rangle \uplus \mathcal{H}^{\text{restart}}_{\mathbf{0}}\langle MD(r_l)\rangle\big)\uplus \\
\mathcal{R}_{(\mathfrak{p},T)}\langle RD(r_{\kappa(\mathfrak{p})})\rangle \uplus \mathcal{H}^{\text{restart}}_{(r_{\kappa(\mathfrak{p})}(\mathfrak{p}),T)}\langle MD(r_{\kappa(\mathfrak{p})})\rangle.
\end{aligned}$$

Before we formally state Duplicator's winning strategy, we briefly describe these critical strategies. First, the only critical positions of $\mathcal{S}^{\text{restart}}_{(\mathfrak{p},T)}$ and $\mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)}$ are inside the initialization gadget and contained in $\mathcal{S}^{\text{init}}$. Thus, this is a good situation for Duplicator, since Spoiler has to restart the game by playing on the initialization gadget. At the beginning of the game Duplicator plays according to the strategy $\mathcal{S}^{\text{init}}$, where position $\mathfrak{s}$ is on the x-vertices. The only critical positions of that strategy are inside the $MS(r)$-gadgets for rules r applicable to $\mathfrak{s}$. If Spoiler pebbles some restart-critical positions there, then Duplicator can switch to $\mathcal{S}^{\text{restart}}_{(\mathfrak{s},T)}$. If Spoiler pebbles an output-critical position on $MS(r)$, then Duplicator can switch to strategy $\mathcal{D}_{r(\mathfrak{s})}$ where position $\mathfrak{p} = r(\mathfrak{s})$ is on the y-vertices. The only critical positions now are inside the switch $MD(r_{\kappa(\mathfrak{p})})$ and inside the initialization gadget. If Spoiler pebbles a restart-critical position on $MD(r_{\kappa(\mathfrak{p})})$, then Duplicator sticks to $\mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)}$. If Spoiler pebbles an output-critical position, then Duplicator chooses strategy $\mathcal{S}_{\mathfrak{p}'}$, where $\mathfrak{p}' = r_{\kappa(\mathfrak{p})}(\mathfrak{p})$. The critical positions from $\mathcal{S}_{\mathfrak{p}'}$ are within the initialization gadget and the

switches $MS(r)$ for rules r applicable to $\mathfrak{p}'$. Thus, combining all these critical strategies allows Duplicator to play forever. Now we define the winning strategy for Duplicator:

$$\mathcal{H} = \mathcal{S}^{\text{init}} \cup \bigcup_{\mathfrak{p}\in\mathcal{K}_1, T\subseteq[\mathsf{k}], T\neq\emptyset} \left(\mathcal{S}_{\mathfrak{p}} \cup \mathcal{S}^{\text{restart}}_{(\mathfrak{p},T)}\right) \cup \bigcup_{\mathfrak{p}\in\mathcal{K}_2, T\subseteq[\mathsf{k}], T\neq\emptyset} \left(\mathcal{D}_{\mathfrak{p}} \cup \mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)}\right).$$

Since $\mathcal{H}$ is a union of critical strategies, it suffices by Lemma 3.5 to show that for each critical strategy $\mathcal{G}$ and each partial homomorphism $h \in \text{crit}(\mathcal{G})$ there is a critical strategy $\mathcal{F}$ such that $h \in \widehat{\mathcal{F}}$. From the definition of the global critical strategies and the properties of the partial critical strategies they contain, it follows that

$$\begin{aligned}
\text{crit}(\mathcal{S}^{\text{restart}}_{(\mathfrak{p},T)}) &\subseteq \widehat{\mathcal{S}}^{\text{init}},\\
\text{crit}(\mathcal{D}^{\text{restart}}_{(\mathfrak{p},T)}) &\subseteq \widehat{\mathcal{S}}^{\text{init}},\\
\text{crit}(\mathcal{S}_{\mathfrak{p}}) &\subseteq \widehat{\mathcal{S}}^{\text{init}} \cup \bigcup_{T\neq\emptyset} \widehat{\mathcal{S}}^{\text{restart}}_{(\mathfrak{p},T)} \cup \bigcup_{l\in\text{appl}(\mathfrak{p})} \widehat{\mathcal{D}}_{r_l(\mathfrak{p})},\\
\text{crit}(\mathcal{D}_{\mathfrak{p}}) &\subseteq \widehat{\mathcal{S}}^{\text{init}} \cup \widehat{\mathcal{S}}_{r_{\kappa(\mathfrak{p})}(\mathfrak{p})} \cup \bigcup_{T\neq\emptyset} \widehat{\mathcal{D}}^{\text{restart}}_{(\mathfrak{p},T)},\\
\text{crit}(\mathcal{S}^{\text{init}}) &\subseteq \bigcup_{l\in\text{appl}(\mathfrak{s})} \widehat{\mathcal{D}}_{r_l(\mathfrak{s})}.
\end{aligned}$$

From the definition of $\mathcal{H}$ and the properties of the KAI-game winning strategy $\mathcal{K}$, it follows, that if a global critical strategy mentioned in the left hand side of the above inclusions is a strategy in $\mathcal{H}$, then so are all strategies on the right hand side. This concludes the proof of Lemma 4.4. □

Part II.

Boolean Satisfiability

5. Resolution

The 3-SAT problem is to decide whether a given 3-CNF formula is satisfiable. In the light of Part I it can be seen as a special constraint satisfaction problem where the domain is $\{0, 1\}$, the variables are the Boolean variables of the CNF formula and the clauses are ternary constraints that restrict possible assignments. This problem was among the first ones shown to be NP-complete and plays a prominent role in computer science. There is a whole field, SAT-solving, which is dedicated to the design of efficient algorithms for 3-SAT. The theoretical backbone of SAT-solving lies in the area of proof complexity. In this branch one tries to understand the inherent complexity of SAT-solvers by analyzing proof systems that model the structure of algorithms for SAT. This approach is the same as the one we have taken in Part II, where we considered a formal derivation system to obtain lower bounds on the inherent complexity of constraint propagation algorithms.

Resolution is a well-known and intensively studied proof system to detect the unsatisfiability of a given formula in conjunctive normal form. Starting with the clauses from the CNF formula one iteratively derives new clauses using only one simple rule: The resolution rule takes two clauses $\gamma \cup \{X\}$, $\delta \cup \{\neg X\}$ and resolves to $\gamma \cup \delta$. The given CNF formula is unsatisfiable if, and only if, the empty clause can be derived. Despite its simplicity resolution has found many applications in practical SAT-solving. For example, treelike resolution can be seen as the underlying proof system for SAT-solvers relying on the DPLL-procedure. Furthermore, modern *clause-learning* SAT-solvers try to find resolution refutations, which may not necessarily be treelike.

One natural complexity measure for resolution is the *length* of a refutation. This measure is also important for resolution based satisfiability testing since the running time of such approaches is lower

bounded by the length of the underlying resolution refutation. Haken [Hak85] proved the first superpolynomial lower bound on the length of resolution refutations for the pigeon hole principle. Several improvements and length lower bounds for other combinatorial principles followed. A second complexity measure is the *width* of a resolution refutation, which is the size of the largest clause in the refutation. Ben-Sasson and Widgerson [BW01] underlined its importance by showing that every length S resolution refutation of an n-variable 3-CNF formula can be transformed to a refutation of width at most $O(\sqrt{n \log S})$. Hence, if a 3-CNF formula has a "short" (subexponential) refutation, then it has also a "narrow" refutation of sublinear width. This fact enabled them to re-derive essentially all previously known exponential length lower bounds by proving linear width lower bounds. Furthermore, they proposed a simple dynamic algorithm that searches for a refutation of smallest width. This heuristics was already known before and dates back to Galil [Gal77]. It proceeds in a very simple way:

for $i = 1, \ldots, n$ **do**
    Derive all clauses of width at most i.
    if the empty clause has been derived **then** reject.
Accept.

Since on n variables there are at most $O(n^k)$ clauses of width k, the algorithm terminates after $n^{O(w)}$ steps, where w is the smallest width of a resolution refutation of Γ. The procedure always finds a resolution refutation, if there is any, and hence is a complete SAT-solver. However, if the w is large, then this algorithm is too space consuming and because of this, such approaches are not used in practice. On the other hand, if one searches only for refutations of width up to k, one gets a polynomial time $(n^{O(k)})$ heuristic, which is correct if $w \leq k$. Note that this situation is similar to constraint propagation and k-consistency (see Chapter 2). In fact, resolution refutations of width k are the same as CSP-refutations of width k for the special CSP that encodes a 3-CNF formula. This connection is discussed in depth in the next section. Afterwards we state our main results on

the complexity of finding bounded width resolution refutations and discuss known results in this area. All results of this chapter appeared in the extended abstract [Ber12b].

5.1. Resolution and Constraint Propagation

In this section we give basic definitions and establish a connection between propagation algorithms and resolution refutations. This connection was revealed by Atserias and Dalmau [AD08]. An extension of this characterization to regular resolution was presented by Hertel [Her08].

Some basic definitions are in order. A *literal* is either a Boolean variable X or its negation $\neg X$. A *clause* γ is a disjunction of literals and the *width* of a clause is the number of literals in it. A CNF formula Γ is a conjunction of clauses and a d-CNF formula is a CNF formula that contains only clauses of width at most d. It is common to view clauses as sets of literals and formulas as sets of clauses. Resolution is a well-known calculus for proving the unsatisfiability of a given CNF formula. The *resolution rule* on X takes two clauses $\gamma \cup \{X\}$ and $\delta \cup \{\neg X\}$ and derives the *resolvent* $\gamma \cup \delta$. A *resolution derivation* of a clause γ from a CNF formula Γ is a sequence of clauses $(\gamma_1, \ldots, \gamma_n)$ such that $\gamma = \gamma_n$ and every clause γ_i is either contained in Γ or a resolvent of two preceding clauses. A *resolution refutation* is a resolution derivation of the empty clause.

The *length* of a resolution derivation is the number of clauses it contains and the *width* of a resolution derivation is the maximum width over all clauses in that derivation. A resolution derivation of γ can also be viewed as a directed acyclic graph (dag) where the nodes are labeled with the clauses from the derivation, one node of in-degree 0 is labeled with γ and all nodes of out-degree 0 are labeled with clauses from Γ. There is one arc from δ to γ_1 and one arc from δ to γ_2 if δ is the resolvent of γ_1 and γ_2. The depth of a resolution derivation of γ from Γ is the number of arcs on the longest directed path in the corresponding dag. A resolution derivation is *regular* if on every path from the root to the leafs in the associated dag no

variable has been used twice by the resolution rule. For technical purposes we also use the *weakening rule* that derives a clause γ from a clause $\delta \subset \gamma$. A resolution refutation with weakening can easily be transformed to a standard resolution refutation without increasing length, width and depth.

To encode a 3-CNF Γ into an instance for the homomorphism problem we define two structures A_Γ with $V(\mathsf{A}_\Gamma) = \mathrm{Var}(\Gamma)$ and B with $V(\mathsf{B}) = \{0, 1\}$ such that every homomorphism from A_Γ to B corresponds to a satisfying assignment for Γ. There are four types of width-3 clauses: $\{X, Y, Z\}$, $\{\neg X, Y, Z\}$, $\{\neg X, \neg Y, Z\}$ and $\{\neg X, \neg Y, \neg Z\}$. Because we also want to encode clauses of width 1 and 2, we do not require X, Y, Z to be distinct. For every type of a clause we introduce a corresponding ternary relation: $R_{(0,0,0)}$, $R_{(1,0,0)}$, $R_{(1,1,0)}$ and $R_{(1,1,1)}$. The index triple denotes the unique falsifying assignment of the corresponding clause type. We let $R_t^{\mathsf{B}} = \{0, 1\}^3 \setminus \{t\}$ be the set of satisfying assignments for clauses of type t. Furthermore, if a clause $\{(\neg)X, (\neg)Y, (\neg)Z\}$ of type t is present in Γ, we add (X, Y, Z) to R_t^{A}. This ensures that every such triple must be mapped to a satisfying assignment for the corresponding clause. Hence, every homomorphism from A_Γ to B corresponds to a satisfying assignment for Γ. Recall the definition of CSP-refutations from Section 2.2. Lines of CSP-refutations of width k are l-partial assignments ($l \leq k$) from A_Γ to B. The derivation rule (2.1) now has the following shape:

$$\frac{p_1' \cup \{x \mapsto 0\} \quad p_2' \cup \{x \mapsto 1\}}{p \supseteq p_1' \cup p_2'} \tag{5.1}$$

We identify every clause γ of width l with the unique partial assignment of domain size l that falsifies it. For example, the clause $\{X, \neg Y, Z\}$ is falsified by the partial assignment $\{X \mapsto 0, Y \mapsto 1, Z \mapsto 0\}$. With this correspondence the rule (5.1) is precisely the resolution rule followed by the weakening rule. Hence, from Lemma 2.8 we get the following characterization.

Lemma 5.1. *The following three statements are equivalent for a 3-CNF formula* Γ*:*

- Γ *has a resolution refutation of width* k *and depth* d*.*
- A_Γ *and* B *have a CSP-derivation of width* k *and depth* d*.*
- *Spoiler wins the existential* $(k+1)$*-pebble game on* A_Γ *and* B *within* d *rounds.*

It is convenient to play the existential $(k+1)$-pebble game directly on the formula, rather than on the structure encoding. That is why we define the following equivalent "width-k game". The game is played by two players, Spoiler and Duplicator, and the positions of the game are partial assignments of domain size at most $k+1$. The game starts with the empty assignment. In each round, Spoiler asks Duplicator for the assignment of a variable X and Duplicator has to answer with either $X \mapsto 0$ or $X \mapsto 1$. Spoiler can store at most $k+1$ variables and its assignments, but he can delete information at any time. After Spoiler has stored the $(k+1)$st assignment, he is forced to delete at least one assignment before doing anything else. Spoiler wins the game if he can reach an assignment that falsifies a clause from Γ and Duplicator wins the game if she has a strategy such that Spoiler can never reach such a position. For illustration we also view a partial assignment p of domain size l as a set of l pebbles marked with 0 or 1 and lying on the variables $\mathrm{Dom}(p)$. Note that, formulated that way, the game is quite similar to Pudlak's Prover-Delayer game for resolution [Pud00] if one bounds the size of the so-called record. A slight modification of the width-k game yields an appropriate game to characterize regular resolution refutations of width at most k [Her08]. The *regular width-k game* proceeds as the width-k game with the restriction that Spoiler is not allowed to ask for a variable twice. The next lemma gives a direct translation between (regular) width-k resolution and the (regular) width-k game. The proof is a straightforward extension of the proof given in [Her08].

Lemma 5.2. *Spoiler wins the (regular) width-k game on* Γ *within* d *rounds if, and only if,* Γ *has a (regular) resolution refutation of width at most* k *and depth at most* d*.*

Proof. We first show how a width-k resolution refutation leads to a winning strategy for Spoiler. Spoiler plays along the arcs in the res-

olution dag from the empty clause to some clause in Γ and always stores the assignment that falsifies the current clause (hence this assignment has domain size at most k). First, the game starts with the empty assignment that corresponds to the empty clause in the derivation. If the current clause is derived from $\gamma_1 \cup \{X\}$ and $\gamma_2 \cup \{\neg X\}$ via resolving on X, then Spoiler asks for X. Depending on Duplicators choice, he walks to either of the two parents and deletes assignments that are not related to the new clause. Finally, he reaches an assignment that falsifies a clause from Γ and thus he wins. Since he follows a path from the root to the leafs in the dag, the number of rounds is bounded by the depth of the refutation. Furthermore, if the refutation is regular, then Spoiler does not query a variable twice.

In a similar way one can develop a resolution refutation of width at most k from a winning strategy for Spoiler in the width-k game. In order to do this we first construct a resolution refutation that also uses the weakening rule. As mentioned before, a resolution refutation with weakening can easily be transformed to a standard resolution refutation without increasing length, width and depth or affecting regularity. The refutation we construct uses the clauses that are falsified by the current assignment, if the domain size is less than $k+1$. For every partial assignment of domain size $k+1$ occurring in the strategy, we consider the clause that relates to the corresponding partial assignment *after* Spoiler was forced to delete one variable. Deleting assignments in Spoilers strategy corresponds to weakening. If Spoiler asks for X, this essentially corresponds to resolving on X, but we have to be a little bit more precise here. Let γ be the clause that relates to the current assignment (that falsifies it) and X be the variable Spoiler asks for. If $|\gamma| < k$, then γ is obtained from $\gamma \cup \{X\}$ and $\gamma \cup \{\neg X\}$ via resolving on X. If $|\gamma| = k$, let $\gamma_1 \subset \gamma \cup \{X\}$ and $\gamma_2 \subset \gamma \cup \{\neg X\}$ be the clauses obtained after Spoiler was forced to delete at least one assignment. Now it holds that (1) γ is (a weakening of) γ_1 or (2) γ is (a weakening of) γ_2 or (3) $X \in \gamma_1$ and $\neg X \in \gamma_2$ and γ is (a weakening of) the resolvent of γ_1 and γ_2. Since every play of the game relates to a path from the empty clause to some clause in Γ in the resolution-dag we get a width-k resolution refutation of depth at most d (after getting rid of the weakening). Furthermore, every

directed path in the proof dag corresponds to a play of the game. Hence, if Spoiler does not query a variable twice, then every variable is used only once by the resolution rule on every directed path in the dag. □

6. Complexity of Bounded Width Resolution

Searching for width-k resolution refutations is a simple polynomial time heuristic for 3-SAT. As shown in the previous section, this heuristic is the same as applying k-consistency to 3-SAT. We already investigated the complexity of k-consistency on binary constraint satisfaction instances in Part I. Unfortunately, these results cannot be translated to 3-SAT and settling the computational complexity of the k-consistency test in this very special situation is the goal of this chapter. Indeed, this was the initial motivation to investigate width-k resolution refutations.

On the other hand, bounded width resolution has wide applications in the area of proof complexity. The first one is a connection to treelike resolution refutations and DPLL-solvers. Ben-Sasson and Widgerson [BW01] showed that if there is a treelike resolution refutation of length S, then there is also a resolution refutation of width $\log S$. Let S_Γ be the length of shortest treelike resolution refutation of an unsatisfiable 3-CNF formula Γ. By searching for a refutation of minimum width, we find a refutation of width at most $\log S_\Gamma$ in time $n^{O(\log S_\Gamma)}$. Since the length of the shortest treelike resolution refutation lower bounds the running time of DPLL-solvers, it follows that the running time of the minimum width heuristic is quasi-polynomial bounded in the running time of an optimal DPLL-solver.

Atserias, Fichte and Thurley [AFT11] studied the connection between bounded width resolution and clause-learning algorithms. They showed that if there is a resolution refutation of width k, then clause-learning algorithms succeed in time $n^{O(k)}$. Thus, "having a width-k refutation" is a structural property of 3-CNF formulas which causes such algorithms to be fast. Because of this, it would be nice to know

in advance whether a formula has a width-k refutation or not. To investigate this question, we consider the following decision problem.

Resolution width-k problem	
Input:	A 3-CNF formula Γ.
Question:	Does Γ have a resolution refutation of width at most k?

By exhaustively deriving all possible clauses of width at most k, we can solve the resolution width-k problem in time $O(\|\Gamma\|^{k+1})$.[1] The main result of this chapter is a lower bound on the time complexity for this problem. It states that trying to derive all possible clauses of width k is essentially the best we can do to decide the resolution width problem.

Theorem 3. *For every integer $k \geq 15$ and $\varepsilon > 0$, the resolution width-k problem can not be decided in time $O(\|\Gamma\|^{\frac{k-2}{12}-\varepsilon})$ for a given 3-CNF formula Γ on multi-tape Turing machines.*

In the same way as for Theorem 2, the proof of Theorem 3 also settles the parameterized complexity of the resolution width problem.

Corollary 6.1. *Parameterized by the width k, the resolution width problem is complete for XP.*

We also investigate the complexity of this problem when k is part of the input for regular and dag-like resolution. Note that a restriction to treelike resolution wouldn't affect the complexity since there is a (regular) dag-like resolution refutation of width k if and only if there is a (regular) treelike resolution refutation of width k.

(Regular) resolution width problem	
Input:	A 3-CNF formula Γ and an integer k.
Question:	Does Γ have a (regular) resolution refutation of width at most k?

[1] This explicit upper bound follows from Lemma 5.1 and Theorem 2.3.

Theorem 4. *The resolution width problem is* EXPTIME*-complete.*

Theorem 5. *The regular resolution width problem is* PSPACE*-complete.*

Some historical remarks are in order. Motivated by an EXPTIME-completeness result for the k-consistency heuristics for general CSP [KP03], Vardi raised the question for the complexity of the resolution width problem and conjectured that it is EXPTIME-complete. In 2006, Hertel and Urquhart [HU06] claimed to have solved the problem, but later retracted their claim [HU09]. They also considered the regular resolution width problem and showed that regularity allows to solve the problem in polynomial space. Some more remarks on their research around the (regular) resolution width problem can be found in Chapter 7 of Hertel's dissertation [Her08].

To get a complete picture it would be interesting to investigate the complexity of finding regular resolution refutations of constant width k. Note that an unconditional lower bound as in Theorem 3 would imply PTIME $\neq$ PSPACE, a conjecture that is seemingly hard to prove. The right framework for investigating this question is parametrized complexity theory. Because the problem can be solved in $n^{O(k)}$, it is contained in XP. It is not hard to give an fpt-reduction that encodes the k-Clique Problem into a 3-CNF formula such that the formula is satisfiable iff the graph contains a k-clique (such encodings are discussed, for example, in [BIS07]). Furthermore, if this formula is unsatisfiable, then it has a regular resolution refutation of width $k+2$. It follows that the regular resolution width-k problem is co-W[1]-hard and not solvable in time $n^{o(k)}$ assuming the exponential time hypothesis [Che+06]. We believe that the regular resolution width-k problem is neither complete for co-W[1] nor complete for XP and leave it for future work to determine the corresponding parametrized complexity class for this problem.

Another related line of research investigates the length of bounded width refutations. There are $O(n^k)$ clauses of width at most k. Hence, the depth and length of every width-k resolution refutation is bounded by $O(n^k)$. The next theorem gives a near optimal lower bound.

Theorem 6.2 ([Ber12b]). *For every fixed integer $k \geq 3$ there is a family of unsatisfiable 3-CNF formulas $\{\Gamma_n^k\}_{n=1}^{\infty}$ with $O(n)$ variables, $O(n^2)$ clauses and minimal refutation width k such that every width-k resolution refutation of Γ_n^k has depth at least $\Omega(n^{k-1})$.*

As a consequence, this theorem provides a strong lower bound on the length of minimum width resolution refutations. A drawback of this result is that the family can easily be refuted in width $k+1$ and constant length (depending on k). Recently, Atserias, Lauria and Nordström [ALN14] provided a family of formulas which can be refuted in width k (hence in length n^k) and showed that every resolution refutation of arbitrary width has length $n^{k/2-o(1)}$. Hence, these formulas show that no resolution-based SAT-solver beats the minimum-width heuristic in the worst case.

6.1. Proof of the Lower Bounds by Reductions

We prove the main theorems via reductions from the k-pebble KAI-game and its acyclic variant[2] to the (regular) width-$(k+1)$ game and the regular variant. The reductions are stated in the next main two lemmas. The lemmas itself are proven at the end of Section 6.5.

Lemma 6.3 (First Main Lemma). *There is a* LOGSPACE*-reduction from the KAI-game to the width game and from the acyclic KAI-game to the regular width game.*

Proof of Theorem 4. It is easy to see that the resolution width problem is in EXPTIME by iteratively resolving all clauses of width at most k. Since determining the winner in the KAI-game is EXPTIME-hard (Theorem 4.3) it is EXPTIME-hard to determine the winner in the width game by Lemma 6.3. Hence, the resolution width problem is complete for EXPTIME. □

[2]See Section 4.1.

Proof of Theorem 5. Spoiler has a forced win in the regular width game if, and only if, he can win the game within $|\operatorname{Var}(\Gamma)|$ steps. Thus, an alternating Turing machine can decide whether Spoiler can win the game in polynomial time. By APTIME = PSPACE [CKS81] we conclude that the regular resolution width problem is in PSPACE. Since the acyclic KAI-game is PSPACE-hard (Theorem 4.3) and there is a LOGSPACE-reduction from the acyclic KAI-game to the regular width game (Lemma 6.3) it follows that the regular resolution width problem is complete for PSPACE. □

Lemma 6.4 (Second Main Lemma)**.** *There is a reduction from the k-pebble KAI-game to the width-$(k+1)$ game that computes for every instance G of size $\|G\|$ a 3-CNF formula $\Gamma(G)$ such that the following holds.*

- *Player 1 has a winning strategy in the k-pebble KAI-game on G if, and only if, Spoiler has a winning strategy in the width-$(k+1)$ game on $\Gamma(G)$.*
- *$\Gamma(G)$ contains $O(\|G\|^3)$ clauses and $O(\|G\|^2)$ variables.*
- *The reduction is computable in* $\mathrm{DTIME}(\|G\|^3)$.

Proof of Theorem 3. Let $k \geq 15$ be a fixed integer and $\varepsilon > 0$. Assume that $\mathbb{A}$ is an algorithm that determines the winner of the width-k game on Γ in time $O(\|\Gamma\|^{\frac{k-2}{12}-\varepsilon})$. Let $\mathbb{B}$ be the algorithm that first applies the reduction from Lemma 6.4 to a given instance G of the $(k-1)$-pebble KAI-game and then executes $\mathbb{A}$. Since $\|\Gamma(G)\| = O(\|G\|^3)$, $\mathbb{B}$ has running time $O(\|G\|^3 + \|G\|^{3(\frac{k-2}{12}-\varepsilon)})$ and thus solves the k'-pebble KAI-game in time $O(\|G\|^{\frac{k'-1}{4}-\varepsilon'})$ for a $k' \geq 14$ and $\varepsilon' > 0$. This contradicts Theorem 4.2. □

The Reduction

We devise one reduction that proves both statements in Lemma 6.3 and a weaker form of Lemma 6.4 (with $\|\Gamma(G)\| = O(\|G\|^4)$) at once. With a slight modification of that reduction we obtain the bounds from Lemma 6.4. For the rest of this chapter let $G = ([n], R, \mathfrak{s}, \gamma)$ with $R = \{r_1, \ldots, r_m\}$, $\mathfrak{s}\colon [k] \to [n]$ and $\gamma \in [n]$ be an instance of the

k-pebble KAI-game. We construct a 3-CNF formula $\Gamma(G)$ such that the following holds.

- Player 1 has a winning strategy in the k-pebble KAI-game on G if, and only if, Spoiler has a winning strategy in the width-$(k+1)$ game on $\Gamma(G)$.

- If G is acyclic and Player 1 has a winning strategy in the k-pebble KAI-game on G, then Spoiler has a winning strategy in the regular width-$(k+1)$ game on $\Gamma(G)$.

Since the width-$(k+1)$ game is just a restricted form of the existential pebble game, the overall strategy in the reduction from the k-pebble KAI-game to the width-$(k+1)$ game parallels the one we have taken in Part I. One obvious difference is that we have to build a 3-CNF formula instead of a pair of graphs. In many cases we can translate our intuition behind the graph-construction to the 3-CNF setting. However, there is one important structural difference that we want to emphasize at this point. In Chapter 4 we reduced the k-pebble KAI-game to the existential $(k+1)$-pebble game. We encoded the position of k KAI-game pebbles with k pebble pairs in the existential pebble game and there was one pebble pair left to "walk" along the edges in the graph. Now the arity of our structure shifts from binary to ternary.[3] Again, we use k pebbles to encode a position of the KAI-game. However, we need *two* more pebbles to walk along ternary clauses. Because of this, we reduce the k-pebble KAI-game to the existential $(k+2)$-pebble game (that we call width-$(k+1)$ game). This minor change makes life much harder for us, because having more pebbles makes Spoiler more powerful and as in the previous Chapter, arguing against every possible move of Spoiler is the most involved part of the proof.

[3]Note that this shift is essential, because every resolution refutation of 2-CNF formulas has width 2.

6.2. Toolbox for the Width Game

We start with adapting the tools for the existential $(k+2)$-pebble game on graphs (as developed in Section 3.2) to the existential $(k+2)$-pebble game (= width-$(k+1)$ game) on 3-CNFs. In our reduction we construct the clause set $\Gamma(G)$ out of smaller clause sets, called gadgets. The gadgets are defined on pairwise disjoint variable sets and there are additional clauses to connect these gadgets. In order to establish a winning strategy for one player, we need to combine strategies on the gadgets to a strategy on Γ. Again, the easier part is to do that for Spoiler. We say that Spoiler *can (regularly) reach* position p_2 from position p_1 on Γ if he has a strategy in the (regular) width-$(k+1)$ game such that starting from position p_1 he either wins the game or position p_2 occurs in the game after some finite number of rounds. We can combine such strategies to show that Spoiler can reach some position p from $\emptyset$; if p falsifies a clause from $\Gamma(G)$ this gives us a winning strategy for Spoiler and hence a resolution refutation. Note that, as on graphs, the "can reach" relation is transitive. However, this does not hold for the regular version. To combine regular strategies on the gadgets to a regular strategy on the entire construction, we additionally make sure that Spoiler uses every gadget only once. As indicated in the proof of Lemma 5.2 there is a tight connection between strategies for Spoiler and resolution derivations. If $|\operatorname{Dom}(p_1)|, |\operatorname{Dom}(p_2)| \leq k+1$, then the notion of reaching positions can also be stated in terms of resolution: Spoiler can (regularly) reach p_2 from p_1 on Γ if, and only if, there is a (regular) width-$(k+1)$ resolution derivation of γ_{p_1} from $\Gamma \cup \{\gamma_{p_2}\}$, where γ_p is the maximal clause falsified by p. The next definition applies the notion of critical strategies for Duplicator in the existential $(k+2)$-pebble game (see Definition 3.4) to the width-$(k+1)$ game.

Definition 6.5. A *critical strategy* for Duplicator in the width-$(k+1)$ game on Γ is a nonempty family $\mathcal{H}$ of partial assignments that do not falsify any clause from Γ and a set of *critical positions* $\operatorname{crit}(\mathcal{H}) \subseteq \mathcal{H}$ such that:

- $p \in \operatorname{crit}(\mathcal{H}) \Rightarrow |\operatorname{Dom}(p)| = k+1$.

- If $p \in \mathcal{H}$ and $p' \subset p$, then $p' \in \mathcal{H}$.
- For every $p \in \mathcal{H} \setminus \mathrm{crit}(\mathcal{H})$, $|\mathrm{Dom}(p)| \leq k+1$, and every variable $Z \in \mathrm{Var}(\Gamma)$ there is a $z \in \{0,1\}$ such that $p \cup \{Z \mapsto z\} \in \mathcal{H}$.

If $\mathrm{crit}(\mathcal{H}) = \emptyset$, then $\mathcal{H}$ is a *winning strategy*.

If there is a winning strategy $\mathcal{H}$ for Duplicator, then she can always provide a correct answer z for a queried variable Z without falsifying any clause from Γ. A critical strategy is nearly a winning strategy in the sense that Duplicator wins unless the game reaches a critical position. Duplicator may not have an appropriate answer in that situation, but she knows that Spoiler has stored a critical position (and nothing else, since $|\mathrm{Dom}(p)| = k+1$) and can use this information to flip to another critical strategy $\mathcal{H}'$ with $p \in \mathcal{H}'$. The following lemma (a restatement of Lemma 3.5) enables us to construct a winning strategy out of a collection of critical strategies.

Lemma 6.6. *If $\mathcal{H}_1, \ldots, \mathcal{H}_l$ are critical strategies on Γ and for all $i \in [l]$ and all $p \in \mathrm{crit}(\mathcal{H}_i)$ there exists a $j \in [l]$ such that $p \in \mathcal{H}_j \setminus \mathrm{crit}(\mathcal{H}_j)$, then $\bigcup_{i \in [l]} \mathcal{H}_i$ is a winning strategy on Γ.* □

Every gadget $Q \subseteq \Gamma(G)$ we construct has a *boundary* $\mathrm{bd}(Q) \subseteq \mathrm{Var}(Q)$, that is the set of the variables on which the gadget is connected to other gadgets. Furthermore, two gadgets Q and Q' are only connected by the clauses $\{X, \neg Y\}$ and $\{\neg X, Y\}$ (denoted $X \leftrightarrow Y$) for variables $X \in \mathrm{bd}(Q)$ and $Y \in \mathrm{bd}(Q')$. A *boundary function* of a strategy $\mathcal{H}$ on a gadget Q is a function $\beta\colon \mathrm{bd}(Q) \to \{0,1\}$ such that $p(X) = \beta(X)$ for all $p \in \mathcal{H}$ and $X \in \mathrm{bd}(Q) \cap \mathrm{Dom}(p)$. We say that two strategies $\mathcal{G}$ and $\mathcal{H}$ on gadgets $Q^{\mathcal{G}}$ and $Q^{\mathcal{H}}$ are *connectable*, if they have boundary functions $\beta_{\mathcal{G}}$ and $\beta_{\mathcal{H}}$ and it holds that $\beta_{\mathcal{G}}(X) = \beta_{\mathcal{H}}(Y)$ for all $(X \leftrightarrow Y) \in \Gamma(G)$, $X \in \mathrm{bd}(Q^{\mathcal{G}})$, $Y \in \mathrm{bd}(Q^{\mathcal{H}})$.

Lemma 6.7. *Let $\mathcal{G}$ and $\mathcal{H}$ be two connectable critical strategies on gadgets $Q^{\mathcal{G}}$ and $Q^{\mathcal{H}}$. The composition $\mathcal{G} \uplus \mathcal{H} := \{g \cup h \mid g \in \mathcal{G}, h \in \mathcal{H}\}$ is a critical strategy on $Q^{\mathcal{G}} \cup Q^{\mathcal{H}}$ and their connecting clauses. Furthermore, $\mathcal{G} \uplus \mathcal{H}$ has critical positions $\mathrm{crit}(\mathcal{G}) \cup \mathrm{crit}(\mathcal{H})$ and the boundary function $\beta_{\mathcal{G}} \cup \beta_{\mathcal{H}}$.* □

We use the operator $\uplus$ to construct a critical strategy for $\Gamma(G)$ out of critical strategies on the gadgets. Then we show that the union of those global critical strategies is by Lemma 6.6 a winning strategy for Duplicator.

6.3. Overview of the Construction

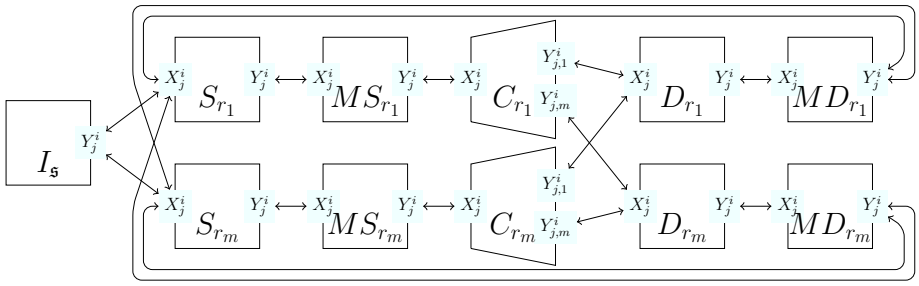


Figure 6.1.: The 3-CNF formula $\Gamma(G)$.

In this paragraph we give an overview on the construction and the gadgets we use. Detailed descriptions of the gadgets and the strategies are given in the next section. We construct $\Gamma(G)$ as illustrated in Figure 6.1. The gadgets and their boundary variables are depicted as boxes and the arrows indicate the connection of the boundary variables. To encode the positions of the KAI-game we introduce Boolean variables X^i_j for $i \in [k]$ and $j \in [n]$, which state "pebble i is on node j". Every position $\mathfrak{p}$ is encoded by the partial assignment $\{X^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$, which will be denoted "position $\mathfrak{p}$ on X". A partial assignment of the variables X^i_j is *invalid*, if there is at least on partition l such that no variable X^l_j is mapped to 1. The boundary of every gadget we construct consists of these variable blocks and we connect two blocks of variables X^i_j and Y^i_j by introducing clauses $X^i_j \leftrightarrow Y^i_j$ (for $i \in [k], j \in [n]$). If two blocks are connected in such a way, then Spoiler can regularly reach $\mathfrak{p}$ on X from $\mathfrak{p}$ on Y and vice versa. In order to do that, Spoiler stores $\mathfrak{p}$ on X and then asks for $Y^1_{\mathfrak{p}(1)}$. Duplicator has to answer with 1 since otherwise this would

falsify the clause $\{\neg X^1_{\mathfrak{p}(1)}, Y^1_{\mathfrak{p}(1)}\}$. Next, Spoiler deletes the assignment $X^1_{\mathfrak{p}(1)} \mapsto 1$ and asks for $Y^2_{\mathfrak{p}(2)}$. Once again, Duplicator has to answer with 1. Following this strategy Spoiler can regularly reach $\mathfrak{p}$ on Y from $\mathfrak{p}$ on X. We want the players to move positions from left to right through the gadgets, that is, they first store a position on the *input* boundary X on the left side, then they play on the gadget and finally they reach a position on the *output* boundary Y on the right side.

The *Initialization Gadget* $I_\mathfrak{s}$ is used to start the game. It has boundary variables $Y(I_\mathfrak{s})^i_j$ ($i \in [k]$, $j \in [n]$) and the feature that Spoiler can regularly reach $\mathfrak{s}$ on $Y(I_\mathfrak{s})$, the assignment that encodes the start position of the KAI-game.

For every rule r there is a *Rule Gadget for Spoiler* S_r with input boundary variables $X(S_r)^i_j$ and output boundary variables $Y(S_r)^i_j$. This gadget is used to modify the current KAI-game position according to rule r. If r is applicable to $\mathfrak{p}$, then Spoiler can regularly reach $r(\mathfrak{p})$ on $Y(S_r)$ from $\mathfrak{p}$ on $X(S_r)$ and he does this whenever Player 1 applies rule r to position $\mathfrak{p}$ in the KAI-game. Since Player 1 starts the KAI-game, the input $X(S_r)^i_j$ of every S_r is connected to the output $Y(I_\mathfrak{s})^i_j$ of the Initialization Gadget. Hence, Spoiler can reach the start position on the input of every S_r. If a position $\mathfrak{p}$ is on the input variables of S_r for a rule r that is not applicable to $\mathfrak{p}$, then Duplicator has a strategy to avoid valid positions at the output of S_r, i.e. there exists a partition l such that no variable $Y(S_r)^l_j$ is mapped to 1. We use this fact to force Spoiler to choose only applicable rules as it is the case for Player 1 in the KAI-game.

After every Rule Gadget S_r there is a copy MS_r of the *Switch* M with input variables $X(M)^i_j$ and output variables $Y(M)^i_j$. From a valid position $\mathfrak{p}$ on $X(M)$ Spoiler can reach $\mathfrak{p}$ on $Y(M)$, but he cannot move invalid positions through the Switch. Duplicator's *impasse strategies* ensure that from an invalid partial assignment on the input variables (where no variable $X(M)^l_j$ is mapped to 1 for at least one l) Spoiler can only reach positions that map output variables to 0. Especially, moving $\mathfrak{p}$ through S_r for a rule r not applicable to $\mathfrak{p}$ leads to an invalid position on the output of S_r and on the input of MS_r and

hence to an impasse. Another property of the Switch is that Spoiler has to reach a critical position inside the Switch in order to move a valid position from the input to the output and thus cannot store assignments outside of the Switch. Moreover, Spoiler cannot reach a position on the input from a position on the output. It follows that once Spoiler moves a position from left to right through the Switch he cannot move backwards and has no information about the variables outside of the Switch.

After every Switch there is a copy C_r of the *Choice Gadget* C that enables Duplicator to chose the next rule. This choice corresponds to the choice of Player 2 in the KAI-game. The Choice Gadget has one block of input variables $X(C)^i_j$ and for every rule r_l a block of output variables $Y(C)^i_{j,l}$. First, if the current position $\mathfrak{p}$ on $X(C)$ is already a winning position for Player 1 ($\mathfrak{p}(i) = \gamma$ for some $i \in [k]$), then Spoiler wins immediately. To ensure that we introduce clauses $\{\neg X(C)^i_\gamma\}$ for every $i \in [k]$ in C. Second, Spoiler can regularly reach $\{Y(C)^i_{\mathfrak{p}(i),q} \mapsto 1 \mid i \in [k]\}$ from $\mathfrak{p}$ on $X(C)$ for some $q \in [m]$ of Duplicator's choice. Duplicator has for every rule r_q a strategy to answer with $\mathfrak{p}$ on the input variables and on the q-th block of the output variables, and with 0 on all other output variables.

The q-th block of output variables of every Choice Gadget C is connected to input variables of the corresponding *Rule Gadget for Duplicator* D_{r_q}. Analog to S_r, these gadgets have input variables $X(D_r)^i_j$ and output variables $Y(D_r)^i_j$, and Spoiler can regularly reach position $r(\mathfrak{p})$ on $Y(D_r)$ from $\mathfrak{p}$ on $X(D_r)$. If Duplicator has chosen a rule r not applicable to the current position $\mathfrak{p}$, then Spoiler wins immediately from $\mathfrak{p}$ on $X(D_r)$. There are Switches also after the D_r gadgets and the output variables of that Switches are connected to the input variables of the S_r gadgets. Hence, Spoiler can move to the rule gadget S_r that corresponds to Player 1's next choice. By playing the way described above, Spoiler can simulate a play of the KAI-game. If this play ends up with a winning position for Player 1, then Spoiler wins the game by falsifying some clause $\{\neg X(C)^i_\gamma\}$. Duplicator's strategies ensure that this is the only way for Spoiler to win the game.

6.4. The Gadgets

Rule Gadget for Spoiler

For every rule $r = (u, v, w, c, d)$ the Rule Gadget for Spoiler S_r consists of variables $X(S_r)^i_j$ and $Y(S_r)^i_j$ for all $i \in [k]$ and $j \in [n]$, which are boundary variables, and the following clauses:

$$X(S_r)^c_u \to Y(S_r)^c_w, \tag{6.1}$$
$$X(S_r)^d_v \to Y(S_r)^d_v, \tag{6.2}$$
$$X(S_r)^i_j \to Y(S_r)^i_j, \qquad i \in [k] \setminus \{c, d\}, j \in [n] \setminus \{w\}. \tag{6.3}$$

Lemma 6.8 (Spoiler's strategy on S_r)**.** *Spoiler can regularly reach* $\mathfrak{p}$ *on* $Y(S_r)$ *from* $\mathfrak{p}$ *on* $X(S_r)$ *for every position* $\mathfrak{p}$ *and every rule* r *applicable to* $\mathfrak{p}$*.*

Proof. By definition, the gadget contains the clauses $X(S_r)^i_{\mathfrak{p}(i)} \to Y(S_r)^i_{r(\mathfrak{p})(i)}$ for $i \in [k]$. Thus, starting from position $\{X(S_r)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ Spoiler can ask for $Y(S_r)^1_{\mathfrak{p}(1)}$ and Duplicator has to answer with $Y(S_r)^1_{\mathfrak{p}(1)} \mapsto 1$. Now, Spoiler deletes $X(S_r)^1_{\mathfrak{p}(1)}$ and asks for $Y(S_r)^2_{\mathfrak{p}(2)}$. Once more, Duplicator has to answer with 1. Following this strategy, Spoiler can regularly reach $\{Y(S_r)^i_{r(\mathfrak{p})(i)} \mapsto 1 \mid i \in [k]\}$. □

The next lemma states that Duplicator does not lose when Spoiler moves through the gadget. Furthermore, if Spoiler has chosen a Rule Gadget S_r and the rule r is not applicable to the current position $\mathfrak{p}$ (hence $T_r(\mathfrak{p}) \neq \emptyset$), then Duplicator has a strategy that avoids valid positions at the output.

Lemma 6.9 (Duplicator's strategies on S_r)**.** *For every position* $\mathfrak{p}$ *Duplicator has a winning strategy* $\mathcal{R}_\mathfrak{p}$ *with boundary function* $\beta_\mathfrak{p}$ *satisfying* $\beta_\mathfrak{p}(X(S_r)^i_j) = 1$ *iff* $j = \mathfrak{p}(i)$*; and* $\beta_\mathfrak{p}(Y(S_r)^i_j) = 1$ *iff* $i \notin T_r(\mathfrak{p})$ *and* $j = r(\mathfrak{p})(i)$*. Furthermore, she has a winning strategy* $\mathcal{R}_0$ *with boundary function* $\beta_0(X^i_j) = \beta_0(Y^i_j) = 0$ *for all* $i \in [k]$ *and* $j \in [n]$*.*

Proof. Let $\mathcal{R}_\mathfrak{p} := \wp(\beta_\mathfrak{p})$ and $\mathcal{R}_0 := \wp(\beta_0)$, where $\beta_\mathfrak{p}$ and β_0 are the boundary functions defined in the above lemma. Since $\beta_\mathfrak{p}$ and β_0

define total assignments that satisfy all clauses from the gadget, $\mathcal{R}_{\mathfrak{p}}$ and $\mathcal{R}_0$ are winning strategies on S_r. □

Rule Gadget for Duplicator

For every rule $r = (u, v, w, c, d)$ the Rule Gadget for Duplicator D_r consists of boundary variables $X(D_r)^i_j$ and $Y(D_r)^i_j$ for all $i \in [k]$ and $j \in [n]$ and the following clauses:

$$X(D_r)^c_u \to Y(D_r)^c_w, \tag{6.4}$$
$$X(D_r)^d_v \to Y(D_r)^d_v, \tag{6.5}$$
$$X(D_r)^i_j \to Y(D_r)^i_j, \qquad i \in [k] \setminus \{c, d\}; j \in [n] \setminus \{w\}, \tag{6.6}$$
$$\neg X(D_r)^c_j, \qquad j \neq u, \tag{6.7}$$
$$\neg X(D_r)^d_j, \qquad j \neq v, \tag{6.8}$$
$$\neg X(D_r)^i_w, \qquad i \in [k] \setminus \{c, d\}. \tag{6.9}$$

As for the S_r gadget, Spoiler can move a valid position through the gadget while applying the rule. If Duplicator has chosen a Rule Gadget for a rule r not applicable to $\mathfrak{p}$, then she is penalized by losing immediately.

Lemma 6.10 (Spoiler's strategy on D_r)**.** *Spoiler can regularly reach $\mathfrak{p}$ on $Y(D_r)$ from $\mathfrak{p}$ on $X(D_r)$ for every position $\mathfrak{p}$ and every rule r applicable to $\mathfrak{p}$. Furthermore, if r is not applicable to $\mathfrak{p}$, then Spoiler wins from position $\mathfrak{p}$ on $X(D_r)$.*

Proof. If r is applicable to $\mathfrak{p}$, then there are clauses $X(S_r)^i_{\mathfrak{p}(i)} \to Y(S_r)^i_{r(\mathfrak{p})(i)}$ for $i \in [k]$. Spoiler can regularly reach $\{Y(D_r)^i_{r(\mathfrak{p})(i)} \mapsto 1 \mid i \in [k]\}$ from $\{X(D_r)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ analog to Lemma 6.8. If r is not applicable to $\mathfrak{p}$, then $\{X(D_r)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ falsifies some clause from (6.7)-(6.9) by definition. □

The next lemma states that Duplicator does not lose the game if the rule is applicable to the current position or if all variables are mapped to 0.

Lemma 6.11 (Duplicator's strategies on D_r)**.** *If r is applicable to $\mathfrak{p}$, then Duplicator has a winning strategy $\mathcal{R}_\mathfrak{p}$ on D_r with boundary function $\beta_\mathfrak{p}(X(D_r)^i_j) = 1$ iff $j = \mathfrak{p}(i)$; and $\beta_\mathfrak{p}(Y(D_r)^i_j) = 1$ iff $j = r(\mathfrak{p})(i)$. Furthermore, she has a winning strategy $\mathcal{R}_0$ with boundary function $\beta_0(X(D_r)^i_j) = \beta_0(Y(D_r)^i_j) = 0$ for all $i \in [k]$ and $j \in [n]$.*

Proof. Analog to Lemma 6.9, let $\mathcal{R}_\mathfrak{p} := \wp(\beta_\mathfrak{p})$ and $\mathcal{R}_0 := \wp(\beta_0)$, where $\beta_\mathfrak{p}$ and β_0 are the boundary functions defined in the above lemma. □

The Switch

The Switch M contains input variables $X(M)^i_j$, output variables $Y(M)^i_j$ and additional variables inside. The clauses of the Switch are given below for all $i, i', l \in [k]$, $j, j' \in [n]$ and $c, c' \in \{1, 2, 3, 4\}$.

$$X(M)^i_j \to A0^i_j \vee A1^i_j \tag{6.10}$$
$$A0^i_j \to A^{i,1}_j \vee A^{i,2}_j \tag{6.11}$$
$$A1^i_j \to A^{i,3}_j \vee A^{i,4}_j \tag{6.12}$$
$$A^{i,c}_j \to A^{i,c}_{j,1} \vee A^{i,c}_{j,\geq 2} \tag{6.13}$$
$$A^{i,c}_{j,\geq l} \to A^{i,c}_{j,l} \vee A^{i,c}_{j,\geq l+1} \qquad 2 \leq l \leq k-2 \tag{6.14}$$
$$A^{i,c}_{j,\geq k-1} \to A^{i,c}_{j,k-1} \vee A^{i,c}_{j,k} \tag{6.15}$$
$$\neg\big(A^{i,c}_{j,l} \wedge A^{i',c'}_{j',l}\big) \qquad i \neq i' \tag{6.16}$$
$$A^{i,c}_{j,l} \to B_l \tag{6.17}$$
$$B_1 \wedge B_{\geq 2} \to B \tag{6.18}$$
$$B_l \wedge B_{\geq l+1} \to B_{\geq l} \qquad 2 \leq l \leq k-2 \tag{6.19}$$
$$B_{k-1} \wedge B_k \to B_{\geq k-1} \tag{6.20}$$
$$A^{i,c}_{j,l} \wedge B \to Y(M)^i_j \tag{6.21}$$

The essence of the Switch can be described by a kind of pigeon hole principle. There are k holes and kn groups of four pigeons each. Every group of four pigeons corresponds to one of the kn variables $X(M)^i_j$. The four pigeons in the pigeon group $X(M)^i_j$ correspond to the four variables $A^{i,1}_j$, $A^{i,2}_j$, $A^{i,3}_j$ and $A^{i,4}_j$. The variables $A^{i,c}_j$ deter-

mine whether the corresponding pigeon is *arriving*. Variable $X(M)^i_j$ says that one pigeon $A^{i,c}_j$ of the pigeon group is arriving (stated by the clauses (6.10), (6.11) and (6.12)). Thus, a partial assignment $\{X(M)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ forces k pigeons to arrive. The variables $A^{i,c}_{j,l}$ say "pigeon $A^{i,c}_j$ sits in hole l". It is ensured by the clauses (6.13), (6.14) and (6.15) that if $A^{i,c}_j$ is arriving, then it will sit in some hole. The clauses (6.16) state that in every hole there is at most one pigeon.

The intended meaning of the variable B_l is "hole l is occupied" and it is ensured by the clauses (6.17) that this variable is true, if some pigeon actually sits in hole l. The variable B states "all holes are occupied" and it is guaranteed by the clauses (6.18), (6.19) and (6.20) that B is true, if all B_l are true. The clauses (6.21) state that if all holes are occupied and pigeon $A^{i,c}_j$ (from the pigeon group $X(M)^i_j$) sits in some hole, then $Y(M)^i_j$ has to be true. Moving a position $\mathfrak{p}$ through the Switch proceeds, roughly, in the following way. At the beginning the partial assignment $\mathfrak{p}$ is on the input $X(M)$. There sits no pigeon in any hole and Duplicator plays according to a critical *input strategy* that maps all output variables to 0. In order to reach $\mathfrak{p}$ on the output, Spoiler has to bring all pigeons into the pigeon house. He can force Duplicator to decide which pigeon from the corresponding pigeon group is arriving and then he forces Duplicator to specify a mapping from the k arriving pigeons to the k holes. Unless the k-th pigeon is arriving, Duplicator maintains $B \mapsto 0$ and thus he can maintain $Y(M)^i_j \mapsto 0$ without falsifying any clause. As soon as every pigeon is arriving the game reaches a critical position. At this point Duplicator flips to an *output strategy* with $B \mapsto 1$ and $Y(M)^i_{\mathfrak{p}(i)} \mapsto 1$. On the other hand, he flips all input variables $X(M)^i_j$ to 0 and hence prevents Spoiler from reaching any position at the input.

If there is an invalid position at the input vertices, then at most $k-1$ variables $X(M)^i_j$ are mapped to 1. Thus, Spoiler can force only $k-1$ pigeons to arrive. Since in that case at most $k-1$ holes are occupied, Duplicator uses an *impasse strategy* to maintain $B \mapsto 0$ and $Y(M)^i_j \mapsto 0$ without contradicting any clause. Therefore, Spoiler cannot move invalid positions through the Switch. The next two

lemmas state Spoiler's and Duplicator's strategies formally, the proofs are deferred to Section 6.6 as they are rather technical.

Lemma 6.12 (Spoiler's strategy on M)**.** *Spoiler can regularly reach* $\mathfrak{p}$ *on* $Y(M)$ *from* $\mathfrak{p}$ *on* $X(M)$*.*

Lemma 6.13 (Duplicator's strategies on M)**.** *For every position* $\mathfrak{p}$ *and every nonempty* $T \subseteq [k]$*, there are strategies* $\mathcal{S}^{imp}_{\mathfrak{p},T}$*,* $\mathcal{S}^{out}_{\mathfrak{p}}$ *and* $\mathcal{S}^{in}_{\mathfrak{p}}$ *for Duplicator satisfying the following conditions.*

(i) The impasse strategy $\mathcal{S}^{imp}_{\mathfrak{p},T}$ *is a winning strategy with boundary function* $\beta(X(M)^i_j) = 1$*, iff* $i \notin T$ *and* $j = \mathfrak{p}(i)$*; and* $\beta(Y(M)^i_j) = 0$*, for all* $i \in [k], j \in [n]$*.*

(ii) The output strategy $\mathcal{S}^{out}_{\mathfrak{p}}$ *is a winning strategy with boundary function* $\beta(X(M)^i_j) = 0$*, for all* $i \in [k]$*,* $j \in [n]$*; and* $\beta(Y(M)^i_j) = 1$*, iff* $j = \mathfrak{p}(i)$*.*

(iii) The input strategy $\mathcal{S}^{in}_{\mathfrak{p}}$ *is a critical strategy with* $\mathrm{crit}(\mathcal{S}^{in}_{\mathfrak{p}}) \subseteq \mathcal{S}^{out}_{\mathfrak{p}}$ *and boundary function* $\beta(X(M)^i_j) = 1$*, iff* $j = \mathfrak{p}(i)$*; and* $\beta(Y(M)^i_j) = 0$*, for all* $i \in [k], j \in [n]$*.*

The Initialization Gadget

For a start position $\mathfrak{s}$ the Initialization Gadget $I_{\mathfrak{s}}$ consists of two Switches M_1 and M_2, start variables S_1 and S_2, and boundary variables $Y(I_{\mathfrak{s}})^i_j$ for all $i \in [k]$ and $j \in [n]$. There are the following clauses in addition to the ones of M_1 and M_2:

$$S_1 \vee S_2 \tag{6.22}$$

$$S_c \to X(M_c)^i_{\mathfrak{s}(i)}, \text{ for all } i \in [k],\ c \in \{1,2\} \tag{6.23}$$

$$Y(M_c)^i_{\mathfrak{s}(i)} \to Y(I_{\mathfrak{s}})^i_{\mathfrak{s}(i)}, \text{ for all } i \in [k],\ c \in \{1,2\} \tag{6.24}$$

Lemma 6.14 (Spoiler's strategy on $I_{\mathfrak{s}}$)**.** *Spoiler can regularly reach* $\mathfrak{s}$ *on* $Y(I_{\mathfrak{s}})$*.*

Proof. First, Spoiler pebbles S_1 and S_2. Because of clause $S_1 \vee S_2$, Duplicator has to answer 1 for S_1 or S_2. Depending on Duplicator's

choice, Spoiler can either reach $\mathfrak{s}$ on $X(M_1)$ or $\mathfrak{s}$ on $X(M_2)$ owing to clauses (6.23). Applying Lemma 6.12, Spoiler can reach $\mathfrak{s}$ on $Y(M_1)$ ($\mathfrak{s}$ on $Y(M_2)$) and thus he can reach $\mathfrak{s}$ on $Y(I_\mathfrak{s})$ using clauses (6.24). □

We can combine the strategies from Lemma 6.13 on the switches M_1 and M_2 to obtain strategies for Duplicator on $I_\mathfrak{s}$. The winning strategy $\mathcal{I}^{\text{init}}$ says that Duplicator does not lose when Spoiler reaches $\mathfrak{s}$ on $Y(I_\mathfrak{s})$. Duplicator uses the critical strategies $\mathcal{I}_\mathfrak{p}^{\text{init}}$ and $\mathcal{I}_0^{\text{init}}$ if other positions than the start position occur at the output of $I_\mathfrak{s}$ during the course of the game.

Lemma 6.15 (Duplicator's strategies on $I_\mathfrak{s}$)**.** *There are strategies $\mathcal{I}^{init}$, $\mathcal{I}_\mathfrak{p}^{init}$ and $\mathcal{I}_0^{init}$ for Duplicator with the following properties.*

(i) $\mathcal{I}^{init}$ *is a winning strategy with boundary function* $\beta(Y(I_\mathfrak{s})_j^i) = 1$*, iff* $j = \mathfrak{s}(i)$*.*

(ii) $\mathcal{I}_\mathfrak{p}^{init}$ *is a critical strategy with* $\operatorname{crit}(\mathcal{I}_\mathfrak{p}^{init}) \subseteq \mathcal{I}^{init}$ *and boundary function* $\beta_\mathfrak{p}(Y(I_\mathfrak{s})_j^i) = 1$*, iff* $j = \mathfrak{p}(i)$*.*

(iii) $\mathcal{I}_0^{init}$ *is a critical strategy with* $\operatorname{crit}(\mathcal{I}_0^{init}) \subseteq \mathcal{I}^{init}$ *and boundary function* $\beta_0(Y(I_\mathfrak{s})_j^i) = 0$ *for all boundary variables* $Y(I_\mathfrak{s})_j^i$*.*

Proof. Recall the strategies $\mathcal{S}_\mathfrak{s}^{\text{out}}$ and $\mathcal{S}_\mathfrak{s}^{\text{in}}$ from Lemma 6.13.

$$\begin{aligned}
\mathcal{I}_1 := \mathcal{S}_\mathfrak{s}^{\text{in}}(M_1) \uplus \mathcal{S}_\mathfrak{s}^{\text{out}}(M_2) \uplus \wp\big(&\{S_1 \mapsto 1, S_2 \mapsto 0\} \cup \\
&\{Y^i_{\mathfrak{s}(i)} \mapsto 1 \mid i \in [k]\} \cup \\
&\{Y^i_j \mapsto 0 \mid i \in [k], j \in [n], j \neq \mathfrak{s}(i)\}\big) \\
\mathcal{I}_2 := \mathcal{S}_\mathfrak{s}^{\text{out}}(M_1) \uplus \mathcal{S}_\mathfrak{s}^{\text{in}}(M_2) \uplus \wp\big(&\{S_1 \mapsto 0, S_2 \mapsto 1\} \cup \\
&\{Y^i_{\mathfrak{s}(i)} \mapsto 1 \mid i \in [k]\} \cup \\
&\{Y^i_j \mapsto 0 \mid i \in [k], j \in [n], j \neq \mathfrak{s}(i)\}\big) \\
\mathcal{I}^{\text{init}} := \mathcal{I}_1 \cup \mathcal{I}_2 &
\end{aligned}$$

By Lemma 6.7, $\mathcal{I}_1$ and $\mathcal{I}_2$ are critical strategies with $\operatorname{crit}(\mathcal{I}_1) = \operatorname{crit}(\mathcal{S}_\mathfrak{s}^{\text{in}}(M_1))$ and $\operatorname{crit}(\mathcal{I}_2) = \operatorname{crit}(\mathcal{S}_\mathfrak{s}^{\text{in}}(M_2))$. From

$$\operatorname{crit}(\mathcal{I}_1) = \operatorname{crit}(\mathcal{S}_\mathfrak{s}^{\text{in}}(M_1)) \subseteq \mathcal{S}_\mathfrak{s}^{\text{out}}(M_2) \subseteq \mathcal{I}_2 \setminus \operatorname{crit}(\mathcal{I}_2) \text{ and}$$

$$\operatorname{crit}(\mathcal{I}_2) = \operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_2)) \subseteq \mathcal{S}^{\text{out}}_{\mathfrak{s}}(M_1) \subseteq \mathcal{I}_1 \setminus \operatorname{crit}(\mathcal{I}_1)$$

it follows that $\mathcal{I}^{\text{init}}$ is a winning strategy by Lemma 6.6. This proves (i), to establish (ii) and (iii) let

$$\begin{aligned}
\mathcal{I}^{\text{init}}_{\mathfrak{p}} := \mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_1) \uplus \mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_2) \uplus \wp\big(&\{S_1 \mapsto 1, S_2 \mapsto 1\} \cup \\
&\{Y^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\} \cup \\
&\{Y^i_j \mapsto 0 \mid i \in [k], j \in [n], j \neq \mathfrak{p}(i)\}\big) \text{ and} \\
\mathcal{I}^{\text{init}}_0 := \mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_1) \uplus \mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_2) \uplus \wp\big(&\{S_1 \mapsto 1, S_2 \mapsto 1\} \cup \\
&\{Y^i_j \mapsto 0 \mid i \in [k], j \in [n]\}\big).
\end{aligned}$$

Lemma 6.7 tells us that $\mathcal{I}^{\text{init}}_{\mathfrak{p}}$ and $\mathcal{I}^{\text{init}}_0$ are critical strategies with $\operatorname{crit}(\mathcal{I}^{\text{init}}_0) = \operatorname{crit}(\mathcal{I}^{\text{init}}_{\mathfrak{p}}) = \operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_1)) \cup \operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{s}}(M_2))$. Therefore, $\operatorname{crit}(\mathcal{I}^{\text{init}}_{\mathfrak{p}}) \subseteq \mathcal{I}^{\text{init}}$ and $\operatorname{crit}(\mathcal{I}^{\text{init}}_0) \subseteq \mathcal{I}^{\text{init}}$. □

The Choice Gadget

The Choice Gadget C contains input variables $X(C)^i_j$ for $i \in [k]$, $j \in [n]$ and output variables $Y(C)^i_{j,q}$ for all $i \in [k]$, $j \in [n]$ and $q \in [m]$ as boundary. Furthermore there are inner variables $E^i_{j,\geq q}$ for $i \in [k]$, $j \in [n]$ and $2 \leq q \leq m-1$. The clauses are given below.

$$\neg X(C)^i_\gamma \qquad i \in [k] \tag{6.25}$$

$$X(C)^i_j \to Y(C)^i_{j,1} \vee E^i_{j,\geq 2} \tag{6.26}$$

$$E^i_{j,\geq q} \to Y(C)^i_{j,q} \vee E^i_{j,\geq q+1} \qquad 2 \leq q \leq m-2 \tag{6.27}$$

$$E^i_{j,\geq m-1} \to Y(C)^i_{j,m-1} \vee Y(C)^i_{j,m} \tag{6.28}$$

$$\neg(Y(C)^i_{j,q} \wedge Y(C)^i_{j,q'}) \qquad q \neq q' \tag{6.29}$$

Lemma 6.16 (Spoiler's strategy on the Choice Gadget)**.** *Spoiler can regularly reach* $\{Y(C)^i_{\mathfrak{p}(i),q} \mapsto 1 \mid i \in [k]\}$ *from* $\{X(C)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ *for some* $q \in [m]$ *of Duplicator's choice.*[4] *Moreover, if* $\mathfrak{p}$ *is a winning position for Player 1, then Spoiler wins immediately.*

[4] This statement in terms of resolution: There is a regular width-$(k+1)$ resolution derivation of $\{\neg X(C)^i_{\mathfrak{p}(i)} \mid i \in [k]\}$ from $\Gamma \cup \{\{\neg Y(C)^i_{\mathfrak{p}(i),q} \mid i \in [k]\} \mid q \in \{m\}\}$.

Proof. Starting from $\{X(C)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ Spoiler picks up the two remaining pebbles and asks for $Y(C)^1_{\mathfrak{p}(1),1}$ and $E^1_{\mathfrak{p}(1),\geq 2}$. Owing to clause (6.26) Duplicator has to answer 1 for one of the two. If Duplicator does not answer with $Y(C)^1_{\mathfrak{p}(1),1} \mapsto 1$, then Spoiler moves the pebbles from $X(C)^1_{\mathfrak{p}(1)}$ and $Y(C)^1_{\mathfrak{p}(1),1}$ to $Y(C)^1_{\mathfrak{p}(1),2}$ and $E^1_{\mathfrak{p}(1),\geq 3}$. Because of clause (6.27) Duplicator has to answer with $Y(C)^1_{\mathfrak{p}(1),2} \mapsto 1$ or $E^1_{\mathfrak{p}(1),\geq 3} \mapsto 1$. Using that strategy Spoiler can reach $\{Y(C)^1_{\mathfrak{p}(1),q} \mapsto 1\} \cup \{X(C)^i_{\mathfrak{p}(i)} \mapsto 1 \mid 2 \leq i \leq k\}$ for some $q \in [m]$ of Duplicator's choice. In the next step Spoiler applies the same technique to the other partitions. If Duplicator chooses a $q' \neq q$ in one of the other partition, then she loses immediately owing to clause (6.29). Thus, Spoiler can reach $\{Y(C)^i_{\mathfrak{p}(i),q} \mapsto 1 \mid i \in [k]\}$. Since he has not pebbled a variable twice, this strategy is regular. In addition, if $\mathfrak{p}$ is a winning position for Spoiler, then $\{X(C)^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ clearly falsifies some clause from (6.25). □

Lemma 6.17 (Duplicator's strategies on the Choice Gadget)**.** *For every position $\mathfrak{p}\colon [k] \to [n] \setminus \{\gamma\}$ and every $q \in [m]$ there is a winning strategy $\mathcal{C}^q_{\mathfrak{p}}$ for Duplicator with boundary function $\beta^q_{\mathfrak{p}}(X(C)^i_j) = 1$, iff $j = \mathfrak{p}(i)$; and $\beta^q_{\mathfrak{p}}(Y(C)^i_{j,l}) = 1$, iff $j = \mathfrak{p}(i)$ and $l = q$. Furthermore, there is a winning strategy $\mathcal{C}_0$ with boundary function β_0 mapping all boundary variables to 0.*

Proof. Let $C^q_{\mathfrak{p}}$ be the total assignment consisting of $\beta^q_{\mathfrak{p}}$ together with

$$E^i_{\mathfrak{p}(i),\geq l} \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \text{ and } l \leq q, \\ 0, \text{ otherwise,} \end{cases}$$

and C_0 be the assignment that maps every variable in the gadget to 0. Since the assignments $C^q_{\mathfrak{p}}$ and C_0 falsify no clause, the strategies $\mathcal{C}^q_{\mathfrak{p}} := \wp(C^q_{\mathfrak{p}})$ and $\mathcal{C}_0 := \wp(C_0)$ are winning strategies with the desired boundary function. □

6.5. Correctness of the Reduction

Now we construct the global strategies for Spoiler (Lemma 6.18) and Duplicator (Lemma 6.19) using the local strategies on the gadgets. Afterwards we present the proofs of our main lemmas (Lemma 6.3 and Lemma 6.4).

Lemma 6.18 (Spoiler's global strategy)**.** *If Player 1 has a winning strategy in the (acyclic) k-pebble KAI-game on G, then Spoiler has a winning strategy in the (regular) width-$(k+1)$ game on $\Gamma(G)$.*

Proof. Assume that Player 1 has a winning strategy in the k-pebble KAI-game on G. We have to show that Spoiler can reach a position that falsifies a clause. First, Spoiler can reach $\mathfrak{s}$ on $Y(I_{\mathfrak{s}})$ via the Initialization Gadget. Let r be the rule applicable to $\mathfrak{s}$ Player 1 chooses first in his strategy and $\mathfrak{p}_1 := r(\mathfrak{s})$. Spoiler can reach $\mathfrak{s}$ on $X(S_r)$ by the connection of the boundary. He can move through the Rule Gadget to $\mathfrak{p}_1$ on $Y(S_r)$ and hence to $\mathfrak{p}_1$ on $X(MS_r)$ since the boundary variables are connected. In the next step he moves through the Switch and reaches $\mathfrak{p}_1$ on $Y(MS_r)$ and then $\mathfrak{p}_1$ on $X(C_r)$. If position $\mathfrak{p}_1$ is a winning position for Player 1 in the KAI-game (that is, one pebble is on node γ), then Spoiler wins immediately. Thus, assume that $\mathfrak{p}_1$ is no winning position and Player 2 chooses a rule r in the KAI-game. At this point Spoiler forces Duplicator to choose a $q \in [m]$ such that he can reach $\{Y(C_r)^i_{\mathfrak{p}_1(i),q} \mapsto 1 \mid i \in [k]\}$ and hence $\mathfrak{p}_1$ on $X(D_{r_q})$. If Duplicator has chosen a $q \in [m]$ such that r_q is not applicable to $\mathfrak{p}_1$, then Spoiler wins immediately, especially he wins if there is no rule applicable to $\mathfrak{p}_1$ and Player 2 is unable to move. Thus, let r_q be applicable to $\mathfrak{p}_1$ and $\mathfrak{p}_2 := r_q(\mathfrak{p}_1)$. Spoiler moves through the Rule Gadget, reaches $\mathfrak{p}_2$ on $Y(D_{r_q})$ and then $\mathfrak{p}_2$ on $X(MD_{r_q})$. Now he moves through the Switch to $\mathfrak{p}_2$ on $Y(MD_{r_q})$. Via the connection of the output variables of the Switch MD_{r_q} to the input variables of the Rule Gadgets S_r, Spoiler chooses a rule r that is applicable to $\mathfrak{p}_2$ and moves to $\mathfrak{p}_2$ on $X(S_r)$. The choice of the rule corresponds to the choice of Player 1 in his winning strategy. In the sequel, Spoiler applies that rule by moving through the Rule Gadget and so on. Simulating the strategy of Player 1 in this way, Spoiler

can reach a position on the input variables of some Choice Gadget that encodes a winning position for Player 1 and thus falsifies a clause $\{\neg X(C)^i_\gamma\}$.

If G is acyclic, then no rule can be applied twice. Thus, following the strategy above Spoiler does not play twice on one gadget. Since all partial strategies on the gadgets are regular this gives rise to a global regular strategy for Spoiler. □

Lemma 6.19 (Duplicator's global strategy)**.** *If Player 2 has a winning strategy in the k-pebble KAI-game on G, then Duplicator has a winning strategy in the width-$(k+1)$ game on $\Gamma(G)$.*

Proof. Let $\mathcal{K} = (\mathcal{K}_1, \mathcal{K}_2, \kappa)$ be a winning strategy for Player 2 in the k-pebble KAI-game on G. We construct a winning strategy $\mathcal{H}$ for Duplicator in the width-$(k+1)$ game on $\Gamma(G)$. First, we define auxiliary critical strategies $\mathcal{H}^1_\mathfrak{p}$, $\mathcal{H}^2_\mathfrak{p}$ and $\mathcal{H}^{\text{init}}$. In order to do this we combine Duplicator's strategies on the gadgets using the $\uplus$-operator and write $\mathcal{A}\langle B\rangle$ to pinpoint strategy $\mathcal{A}$ on gadget B. For all $\mathfrak{p} \in \mathcal{K}_1$ let

$$\begin{aligned}\mathcal{H}^1_\mathfrak{p} := \mathcal{I}^{\text{init}}_\mathfrak{p} \uplus \biguplus_r \big(\mathcal{C}_0\langle C_r\rangle \uplus \mathcal{R}_0\langle D_r\rangle \uplus \mathcal{S}^{\text{out}}_\mathfrak{p}\langle MD_r\rangle\big) \uplus \\ \biguplus_{r\in \text{appl}(\mathfrak{p})} \big(\mathcal{R}_\mathfrak{p}\langle S_r\rangle \uplus \mathcal{S}^{\text{in}}_{r(\mathfrak{p})}\langle MS_r\rangle\big) \uplus \\ \biguplus_{r\notin \text{appl}(\mathfrak{p})} \big(\mathcal{R}_\mathfrak{p}\langle S_r\rangle \uplus \mathcal{S}^{\text{imp}}_{r(\mathfrak{p}),T_r(\mathfrak{p})}\langle MS_r\rangle\big).\end{aligned}$$

The initialization strategy $\mathcal{H}^{\text{init}}$ differs from $\mathcal{H}^1_\mathfrak{s}$ only in the choice of the strategy on the Initialization Gadget: It contains the winning strategy $\mathcal{I}^{\text{init}}$ instead of the critical strategy $\mathcal{I}^{\text{init}}_\mathfrak{s}$. For all $\mathfrak{p} \in \mathcal{K}_2$ let

$$\begin{aligned}\mathcal{H}^2_\mathfrak{p} := \mathcal{I}^{\text{init}}_0 \uplus \biguplus_r \big(\mathcal{C}^{\kappa(\mathfrak{p})}_\mathfrak{p}\langle C_r\rangle \uplus \mathcal{R}_0\langle S_r\rangle \uplus \mathcal{S}^{\text{out}}_\mathfrak{p}\langle MS_r\rangle\big) \uplus \\ \biguplus_{r\neq r_{\kappa(\mathfrak{p})}} \big(\mathcal{R}_0\langle D_r\rangle \uplus \mathcal{S}^{\text{imp}}_{\mathfrak{p},[k]}\langle MD_r\rangle\big) \uplus \\ \big(\mathcal{R}_\mathfrak{p}\langle D_{r_{\kappa(\mathfrak{p})}}\rangle \uplus \mathcal{S}^{\text{in}}_{r_{\kappa(\mathfrak{p})}(\mathfrak{p})}\langle MD_{r_{\kappa(\mathfrak{p})}}\rangle\big).\end{aligned}$$

Note that all these strategies are by Lemma 6.7 global critical strategies. The strategies above enable Duplicator to simulate the moves of the KAI-game. Playing within strategy $\mathcal{H}^{\text{init}}$ means that the KAI-game has just started, position $\mathfrak{s}$ is on the board and it is Player 1's turn. Duplicator uses the strategy $\mathcal{H}^1_{\mathfrak{p}}$ ($\mathcal{H}^2_{\mathfrak{p}}$) to ensure that the current position in the KAI-game is $\mathfrak{p}$ and it is Player 1's (Player 2's) turn. If Spoiler reaches a critical position on the switches, then Duplicator flips the strategies in the same way as the positions in the KAI-game change. Let us now define the winning strategy $\mathcal{H}$ formally: $\mathcal{H} := \mathcal{H}^{\text{init}} \cup \bigcup_{\mathfrak{p}\in\mathcal{K}_1} \mathcal{H}^1_{\mathfrak{p}} \cup \bigcup_{\mathfrak{p}\in\mathcal{K}_2} \mathcal{H}^2_{\mathfrak{p}}$.

To verify that $\mathcal{H}$ is indeed a winning strategy it remains to show, by Lemma 6.6, that every critical position in one auxiliary strategy is contained as non-critical position in some other auxiliary strategy. For a strategy $\mathcal{A}$ let $\widehat{\mathcal{A}} := \mathcal{A} \setminus \text{crit}(\mathcal{A})$. We get the inclusions below by Lemma 6.13 and Lemma 6.15 for all $\mathfrak{p}_1 \in \mathcal{K}_1$ and $\mathfrak{p}_2 \in \mathcal{K}_2$.

$$
\begin{aligned}
\text{crit}(\mathcal{H}^{\text{init}}) &= \bigcup_{r\in\text{appl}(\mathfrak{s})} \text{crit}(\mathcal{S}^{\text{in}}_{r(\mathfrak{s})}\langle MS_r\rangle) \\
&\subseteq \bigcup\nolimits_{r\in\text{appl}(\mathfrak{s})} \widehat{\mathcal{H}}^2_{r(\mathfrak{s})} \\
\text{crit}(\mathcal{H}^1_{\mathfrak{p}_1}) &= \text{crit}(\mathcal{I}^{\text{init}}_{\mathfrak{p}_1}) \cup \bigcup_{r\in\text{appl}(\mathfrak{p}_1)} \text{crit}(\mathcal{S}^{\text{in}}_{r(\mathfrak{p}_1)}\langle MS_r\rangle) \\
&\subseteq \widehat{\mathcal{H}}^{\text{init}} \cup \bigcup_{r\in\text{appl}(\mathfrak{p}_1)} \widehat{\mathcal{H}}^2_{r(\mathfrak{p}_1)} \\
\text{crit}(\mathcal{H}^2_{\mathfrak{p}_2}) &= \text{crit}(\mathcal{I}^{\text{init}}_{\mathfrak{p}_2}) \cup \text{crit}(\mathcal{S}^{\text{in}}_{r_{\kappa(\mathfrak{p}_2)}(\mathfrak{p}_2)}\langle MD_{r_{\kappa(\mathfrak{p}_2)}}\rangle) \\
&\subseteq \widehat{\mathcal{H}}^{\text{init}} \cup \widehat{\mathcal{H}}^1_{r_{\kappa(\mathfrak{p}_2)}(\mathfrak{p}_2)}
\end{aligned}
$$

Since $\mathfrak{s} \in \mathcal{K}_1$, it follows that $r(\mathfrak{s}) \in \mathcal{K}_2$ and hence $\mathcal{H}^2_{r(\mathfrak{s})} \subseteq \mathcal{H}$ for all r applicable to $\mathfrak{s}$. Because $\mathfrak{p}_1 \in \mathcal{K}_1$, it holds that $r(\mathfrak{p}_1) \in \mathcal{K}_2$ and thus $\mathcal{H}^2_{r(\mathfrak{p}_1)} \subseteq \mathcal{H}$ for all $r \in \text{appl}(\mathfrak{p}_1)$. Since $\mathfrak{p}_2 \in \mathcal{K}_2$, it follows that $r_{\kappa(\mathfrak{p}_2)}(\mathfrak{p}_2) \in \mathcal{K}_1$ and thus $\mathcal{H}^1_{r_{\kappa(\mathfrak{p}_2)}(\mathfrak{p}_2)} \subseteq \mathcal{H}$. Consequently, all strategies mentioned in the above inclusion are contained in $\mathcal{H}$.

□

Proof of the First Main Lemma (Lemma 6.3). It is easy to verify that the reduction can be performed in LOGSPACE. Lemma 6.3 then follows from Lemma 6.18 and Lemma 6.19. □

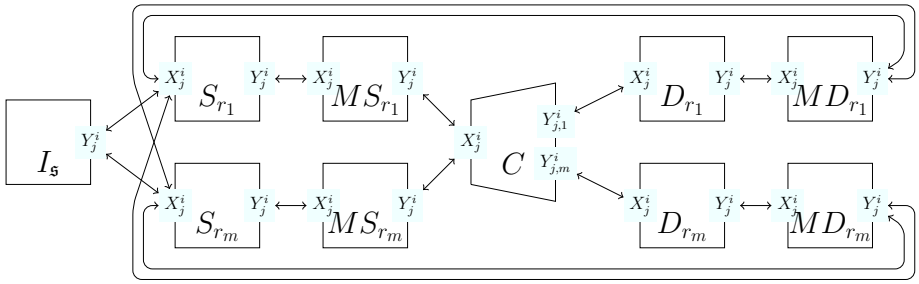


Figure 6.2.: The 3-CNF formula $\Gamma'(G)$.

Proof of the Second Main Lemma (Lemma 6.4). The overall number of clauses used by all gadgets in $\Gamma(G)$ is bounded by $O(\|G\|^4)$. The most wasteful part is the set of $O(m)$ Choice Gadgets of size $O(knm^2)$ each. However, since we do not argue about regular refutations here, it suffices to use one Choice Gadget whose input variables are connected to the output variables of all MS_r gadgets. The modified construction $\Gamma'(G)$ is illustrated in Figure 6.2. The proof of Lemma 6.18 and Lemma 6.19 goes through with that modification (except for regularity). With this clause set $\Gamma'(G)$ we get $\|\Gamma'(G)\| = O(\|G\|^3)$ and $|\operatorname{Var}(\Gamma'(G))| = O(\|G\|^2)$ as desired. □

6.6. Strategies on the Switch

Recall from Section 6.4 the definition of and intuition behind the Switch. For the readers convenience we restate the clauses and enumerate the variables explicitly. Unless stated otherwise we assume $i, i', l \in [k]$, $j, j' \in [n]$ and $c, c' \in \{1, 2, 3, 4\}$.

$$X(M)^i_j \to A0^i_j \vee A1^i_j \tag{6.10}$$

$$A0^i_j \to A^{i,1}_j \vee A^{i,2}_j \tag{6.11}$$
$$A1^i_j \to A^{i,3}_j \vee A^{i,4}_j \tag{6.12}$$
$$A^{i,c}_j \to A^{i,c}_{j,1} \vee A^{i,c}_{j,\geq 2} \tag{6.13}$$
$$A^{i,c}_{j,\geq l} \to A^{i,c}_{j,l} \vee A^{i,c}_{j,\geq l+1} \qquad 2 \leq l \leq k-2 \tag{6.14}$$
$$A^{i,c}_{j,\geq k-1} \to A^{i,c}_{j,k-1} \vee A^{i,c}_{j,k} \tag{6.15}$$
$$\neg\big(A^{i,c}_{j,l} \wedge A^{i',c'}_{j',l}\big) \qquad i \neq i' \tag{6.16}$$
$$A^{i,c}_{j,l} \to B_l \tag{6.17}$$
$$B_1 \wedge B_{\geq 2} \to B \tag{6.18}$$
$$B_l \wedge B_{\geq l+1} \to B_{\geq l} \qquad 2 \leq l \leq k-2 \tag{6.19}$$
$$B_{k-1} \wedge B_k \to B_{\geq k-1} \tag{6.20}$$
$$A^{i,c}_{j,l} \wedge B \to Y(M)^i_j \tag{6.21}$$

Variables:

$$X(M)^i_j, \tag{6.30}$$
$$A0^i_j, A1^i_j, A^{i,1}_j, A^{i,2}_j, A^{i,3}_j, A^{i,4}_j, \tag{6.31}$$
$$A^{i,c}_{j,l}, \tag{6.32}$$
$$A^{i,c}_{j,\geq l}, \qquad 2 \leq l \leq k-1 \tag{6.33}$$
$$B_l, \tag{6.34}$$
$$B_{\geq l}, \qquad 2 \leq l \leq k-1 \tag{6.35}$$
$$B, \tag{6.36}$$
$$Y(M)^i_j. \tag{6.37}$$

Let $\boldsymbol{A}^i$ be the set of variables from lines (6.31) – (6.33) with upper index i and $\boldsymbol{A} := \bigcup_{i\in[k]} \boldsymbol{A}^i$. By $\boldsymbol{B}$ we denote the set of variables from lines (6.34) – (6.36).

Proof of Lemma 6.12

Reminder of Lemma 6.12 (Spoiler's strategy on M)**.** Spoiler can regularly reach $\mathfrak{p}$ on $Y(M)$ from $\mathfrak{p}$ on $X(M)$.

Proof of Lemma 6.12. Starting from $\{X^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ Spoiler asks for $A0^1_{\mathfrak{p}(1)}$ and $A1^1_{\mathfrak{p}(1)}$ with the two remaining pebbles. Because of (6.10), Duplicator has to answer 1 for $A0^1_{\mathfrak{p}(1)}$ or $A1^1_{\mathfrak{p}(1)}$. Assume that Spoiler reaches $A0^1_{\mathfrak{p}(1)} \mapsto 1$. Then he picks up the pebbles from $A1^1_{\mathfrak{p}(1)}$ and $X^1_{\mathfrak{p}(1)}$ and places them on $A^{1,1}_{\mathfrak{p}(1)}$ and $A^{1,2}_{\mathfrak{p}(1)}$. Owing to (6.11), Duplicator has to answer 1 for $A^{1,1}_{\mathfrak{p}(1)}$ or $A^{1,2}_{\mathfrak{p}(1)}$. If Duplicator has answered with $A1^1_{\mathfrak{p}(1)} \mapsto 1$ above, then Spoiler could reach $A^{1,3}_{\mathfrak{p}(1)} \mapsto 1$ or $A^{1,4}_{\mathfrak{p}(1)} \mapsto 1$ in an analog way. Thus, there is a $c \in [4]$ of Duplicator's choice such that Spoiler can reach $\{A^{1,c}_{\mathfrak{p}(1)} \mapsto 1\} \cup \{X^i_{\mathfrak{p}(i)} \mapsto 1 \mid 2 \leq i \leq k\}$. Spoiler now takes the remaining two pebbles and applies the same strategy to the other partitions. Therefore, he reaches the position $\{A^{i,b(i)}_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ for some mapping $b : [k] \to [4]$. As before, Spoiler maintains one pebble on every partition to store information and uses the two remaining pebbles to walk within one partition. Using (6.13), (6.14) and (6.15) Spoiler can reach position $A^{1,b(1)}_{\mathfrak{p}(1),l_1} \mapsto 1$, for some $l_1 \in [k]$ of Duplicator's choice, without grabbing the pebbles from the other partitions. Using the same technique Spoiler can reach $A^{2,b(2)}_{\mathfrak{p}(2),l_2} \mapsto 1$ in partition 2. If Duplicator chooses $l_2 = l_1$, then she loses immediately because of clause (6.16). Thus, she has to choose some $l_2 \in [k] \setminus \{l_1\}$. Following this strategy, Spoiler can successively reach $A^{i,b(i)}_{\mathfrak{p}(i),l_i} \mapsto 1$ for some $l_i \in [k] \setminus \{l_1, \ldots, l_{i-1}\}$ and eventually $\{A^{i,b(i)}_{\mathfrak{p}(i),\sigma(i)} \mapsto 1 \mid i \in [k]\}$ for some permutation $\sigma\colon [k] \to [k]$.

At this point Spoiler asks for B. Assume first that Duplicator answers with 1. Because of clause (6.21) Spoiler can force Duplicator to answer 1 when he asks for $Y^1_{\mathfrak{p}(1)}$ with the remaining pebble. Then Spoiler picks up the pebble from $A^{1,b(1)}_{\mathfrak{p}(1),\sigma(1)}$ and puts it on $Y^2_{\mathfrak{p}(2)}$. Once more, clause (6.21) forces Duplicator to answer with 1. Playing that strategy also on the other partitions, Spoiler can reach $\{Y^i_{\mathfrak{p}(i)} \mapsto 1 \mid i \in [k]\}$ and is done.

So assume that Duplicator answers 0 when he is asked for B. The current position is $\{A^{i,b(i)}_{\mathfrak{p}(i),\sigma(i)} \mapsto 1 \mid i \in [k]\} \cup \{B \mapsto 0\}$. Using the clauses (6.17) and the $(k{+}2)$-th pebble, Spoiler can reach $\{B_l \mapsto 1 \mid l \in [k]\} \cup \{B \mapsto 0\}$. In the next step Spoiler asks for $B_{\geq k-1}$ and

Duplicator has to answer with 1 owing to clause (6.20). Then Spoiler picks up the pebble from B_k and asks for $B_{\geq k-2}$. Duplicator has to answer with 1 according to clause (6.19). Following this strategy Spoiler can reach positions $\{B_l \mapsto 1 \mid 1 \leq l < i\} \cup \{B_{\geq i} \mapsto 1, B \mapsto 0\}$ for $i = (k-1) \dots 2$. Since $\{B_1 \mapsto 1, B_{\geq 2} \mapsto 1, B \mapsto 0\}$ falsifies clause (6.18), Spoiler wins the game. □

Proof of Lemma 6.13

Reminder of Lemma 6.13 (Duplicator's strategies on M)**.** For every position $\mathfrak{p}$ and every nonempty $T \subseteq [k]$, there are strategies $\mathcal{S}^{\text{imp}}_{\mathfrak{p},T}$, $\mathcal{S}^{\text{out}}_{\mathfrak{p}}$ and $\mathcal{S}^{\text{in}}_{\mathfrak{p}}$ for Duplicator satisfying the following conditions.

(i) The impasse strategy $\mathcal{S}^{\text{imp}}_{\mathfrak{p},T}$ is a winning strategy with boundary function $\beta(X(M)^i_j) = 1$, iff $i \notin T$ and $j = \mathfrak{p}(i)$; and $\beta(Y(M)^i_j) = 0$, for all $i \in [k], j \in [n]$.

(ii) The output strategy $\mathcal{S}^{\text{out}}_{\mathfrak{p}}$ is a winning strategy with boundary function $\beta(X(M)^i_j) = 0$, for all $i \in [k]$, $j \in [n]$; and $\beta(Y(M)^i_j) = 1$, iff $j = \mathfrak{p}(i)$.

(iii) The input strategy $\mathcal{S}^{\text{in}}_{\mathfrak{p}}$ is a critical strategy with $\text{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{p}}) \subseteq \mathcal{S}^{\text{out}}_{\mathfrak{p}}$ and boundary function $\beta(X(M)^i_j) = 1$, iff $j = \mathfrak{p}(i)$; and $\beta(Y(M)^i_j) = 0$, for all $i \in [k], j \in [n]$.

Proof of Lemma 6.13. Let $T \subseteq [k]$ be a nonempty set and $\mathfrak{p} : [k] \to [n]$ a position. Since T is nonempty we can fix some $t^* \in T$, say the minimal one. First, we present a total assignment $S^{\text{imp}}_{\mathfrak{p},T}$ that falsifies no clause from the Switch and defines Duplicator's impasse strategy (i). Translated to our pigeonhole metaphor (see page 120), the mapping states that $|[k] \setminus T| < k$ pigeons are arriving, where Duplicator always takes the first pigeon $A^{i,1}_{\mathfrak{p}(i)}$ from the pigeon group $X^i_{\mathfrak{p}(i)}$ and assigns it to hole i. Hence, hole t^* is unoccupied and therefore the pigeon house is not full ($B \mapsto 0$).

$$X^i_j \mapsto \begin{cases} 1, \text{ if } i \notin T \text{ and } j = \mathfrak{p}(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.38}$$

$$A0^i_j \mapsto \begin{cases} 1, \text{ if } i \notin T \text{ and } j = \mathfrak{p}(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.39}$$

$$A1^i_j \mapsto 0 \tag{6.40}$$

$$A^{i,c}_j \mapsto \begin{cases} 1, \text{ if } i \notin T,\ j = \mathfrak{p}(i) \text{ and } c = 1 \\ 0, \text{ otherwise} \end{cases} \tag{6.41}$$

$$A^{i,c}_{j,l} \mapsto \begin{cases} 1, \text{ if } i \notin T,\ j = \mathfrak{p}(i),\ c = 1 \text{ and } l = i \\ 0, \text{ otherwise} \end{cases} \tag{6.42}$$

$$A^{i,c}_{j,\geq l} \mapsto \begin{cases} 1, \text{ if } i \notin T,\ j = \mathfrak{p}(i),\ c = 1 \text{ and } l \leq i \\ 0, \text{ otherwise} \end{cases} \tag{6.43}$$

$$B_l \mapsto \begin{cases} 1, \text{ if } l \notin T \\ 0, \text{ otherwise} \end{cases} \tag{6.44}$$

$$B_{\geq l} \mapsto \begin{cases} 1, \text{ if } l > t^* \\ 0, \text{ otherwise} \end{cases} \tag{6.45}$$

$$B \mapsto 0 \tag{6.46}$$

$$Y^i_j \mapsto 0 \tag{6.47}$$

Now we define impasse strategy of Duplicator $\mathcal{S}^{\text{imp}}_{\mathfrak{p},T}$ to be the powerset of $S^{\text{imp}}_{\mathfrak{p},T}$. In a similar way we define total satisfying assignments which provide the output strategy. For the rest of the proof the assignment $b : [k] \to [4]$ specifies Duplicator's choice of the arriving pigeon $A^{i,b(i)}_{\mathfrak{p}(i)}$ from the pigeon group $X^i_{\mathfrak{p}(i)}$. Furthermore, we denote by $\sigma : [k] \to [k]$ the permutation that assigns k arriving pigeons to k holes. For all mappings $b : [k] \to [4]$ and a permutations $\sigma : [k] \to [k]$ the satisfying assignment $S^{\text{out}}_{\mathfrak{p},b,\sigma}$ states that the pigeon house is full, Duplicator chooses pigeons out of pigeon groups according to b and assigns pigeons to holes according to σ. Formally, $S^{\text{out}}_{\mathfrak{p},b,\sigma}$ is defined as follows.

$$X^i_j \mapsto 0 \tag{6.48}$$

$$A0^i_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \text{ and } b(i) \in \{1,2\} \\ 0, \text{ otherwise} \end{cases} \tag{6.49}$$

$$A1^i_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \text{ and } b(i) \in \{3,4\} \\ 0, \text{ otherwise} \end{cases} \tag{6.50}$$

$$A^{i,c}_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \text{ and } b(i) = c \\ 0, \text{ otherwise} \end{cases} \tag{6.51}$$

$$A^{i,c}_{j,l} \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i),\ b(i) = c \text{ and } l = \sigma(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.52}$$

$$A^{i,c}_{j,\geq l} \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i),\ b(i) = c \text{ and } l \leq \sigma(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.53}$$

$$B_l \mapsto 1 \tag{6.54}$$

$$B_{\geq l} \mapsto 1 \tag{6.55}$$

$$B \mapsto 1 \tag{6.56}$$

$$Y^i_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \\ 0, \text{ otherwise.} \end{cases} \tag{6.57}$$

The output strategy $\mathcal{S}^{\text{out}}_{\mathfrak{p}}$ is the union of the powersets of $S^{\text{out}}_{\mathfrak{p},b,\sigma}$ for all possible choices of b and σ. Designing the critical input strategy is a little more difficult. For this, let $t \in [k]$, $b : [k] \to [4]$ and $\sigma : [k] \to [k]$ be a permutation on $[k]$. We define a partial assignment $S^t_{\mathfrak{p},b,\sigma}$, which does not falsify any clause. The intended meaning of this mapping is that Duplicator chooses pigeons out of the pigeon groups according to b and assigns holes to the pigeons according to σ. The difference to the output strategy is, that in partition t the chosen pigeon $A^{t,b(t)}_{\mathfrak{p}(t)}$ has not yet arrived. And because of this, $A^{t,b(t)}_{\mathfrak{p}(t)}$ and the assignment to its hole $A^{t,b(t)}_{\mathfrak{p}(t),\sigma(t)}$ are undefined. Moreover, the hole $\sigma(t)$ is still empty ($B_{\sigma(t)} \mapsto 0$) and the house is not full ($B \mapsto 0$). Note that this partial assignment cannot be extended to a satisfying assignment, because pigeon $A^{t,b(t)}_{\mathfrak{p}(t)}$ would be forced to arrive and to sit in hole $A^{t,b(t)}_{\mathfrak{p}(t),\sigma(t)}$, which contradicts the emptiness of this hole. Now

we give a formal definition of $S^t_{\mathfrak{p},b,\sigma}$.

$$X^i_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.58}$$

$$A0^i_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \text{ and } b(i) \in \{1,2\} \\ 0, \text{ otherwise} \end{cases} \tag{6.59}$$

$$A1^i_j \mapsto \begin{cases} 1, \text{ if } j = \mathfrak{p}(i) \text{ and } b(i) \in \{3,4\} \\ 0, \text{ otherwise} \end{cases} \tag{6.60}$$

$$A^{i,c}_j \mapsto \begin{cases} \text{undefined, if } i = t,\ j = \mathfrak{p}(i) \text{ and } c = b(i) \\ 1, \text{ if } i \neq t,\ j = \mathfrak{p}(i) \text{ and } c = b(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.61}$$

$$A^{i,c}_{j,l} \mapsto \begin{cases} \text{undefined, if } i = t,\ j = \mathfrak{p}(i),\ c = b(i) \text{ and } l = \sigma(i) \\ 1, \text{ if } i \neq t,\ j = \mathfrak{p}(i),\ c = b(i) \text{ and } l = \sigma(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.62}$$

$$A^{i,c}_{j,\geq l} \mapsto \begin{cases} \text{undefined, if } i = t,\ j = \mathfrak{p}(i),\ c = b(i) \text{ and } l \leq \sigma(i) \\ 1, \text{ if } i \neq t,\ j = \mathfrak{p}(i),\ c = b(i) \text{ and } l \leq \sigma(i) \\ 0, \text{ otherwise} \end{cases} \tag{6.63}$$

$$B_l \mapsto \begin{cases} 0, \text{ if } l = \sigma(t) \\ 1, \text{ otherwise} \end{cases} \tag{6.64}$$

$$B_{\geq l} \mapsto \begin{cases} 0, \text{ if } l \leq \sigma(t) \\ 1, \text{ otherwise} \end{cases} \tag{6.65}$$

$$B \mapsto 0 \tag{6.66}$$

$$Y^i_j \mapsto 0 \tag{6.67}$$

For the impasse and output strategies we simply take the set of all partial assignments contained in the total assignments as winning

strategies. For the input strategies we have to be more careful and set $\mathcal{S}^t_{\mathfrak{p},b,\sigma} = \{p \mid p \subset S^t_{\mathfrak{p},b,\sigma}, |\operatorname{Dom}(p)| \leq k+2, |\operatorname{Dom}(p) \cap \boldsymbol{A}^t| \leq 2\}$.

$$\mathcal{S}^{\text{out}}_{\mathfrak{p}} := \{p \mid p \subseteq S^{\text{out}}_{\mathfrak{p},b,\sigma}, b\colon [k] \to [4], \sigma\colon [k] \to [k]\} \tag{6.68}$$

$$\mathcal{S}^{\text{imp}}_{\mathfrak{p},T} := \{p \mid p \subseteq S^{\text{imp}}_{\mathfrak{p},T}\} \tag{6.69}$$

$$\mathcal{S}^{\text{in}}_{\mathfrak{p}} := \bigcup_{t;\, b;\, \sigma} \mathcal{S}^t_{\mathfrak{p},b,\sigma} \tag{6.70}$$

It is clear by definition that all strategies have the desired boundary and satisfy the closure property. Furthermore, the output strategies $\mathcal{S}^{\text{out}}_{\mathfrak{p}}$ and the impasse strategies $\mathcal{S}^{\text{imp}}_{\mathfrak{p},T}$ are winning strategies since they are drawn from total assignments that falsify no clause.

The most difficult case is (iii). We define $\operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{p}})$ to be the set of all positions $p \in \mathcal{S}^{\text{in}}_{\mathfrak{p}}$ with domain size $k+1$ such that there exists a $t \in [k]$ with $|\operatorname{Dom}(p) \cap \boldsymbol{A}^t| = 2$ and $|\operatorname{Dom}(p) \cap \boldsymbol{A}^i| = 1$ for all $i \in [k] \setminus \{t\}$. For every $p \in \operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{p}})$ it holds that $\operatorname{Dom}(p) \subseteq \boldsymbol{A}$ and $p \subseteq S^t_{\mathfrak{p},b,\sigma}$ for some $b\colon [k] \to [4]$ and $\sigma\colon [k] \to [k]$. Since $S^t_{\mathfrak{p},b,\sigma}$ and $S^{\text{out}}_{\mathfrak{p},b,\sigma}$ agree on all variable from $\boldsymbol{A}$ where $S^t_{\mathfrak{p},b,\sigma}$ is defined, it follows that $p \subseteq S^{\text{out}}_{\mathfrak{p},b,\sigma}$ and hence $\operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{p}}) \subseteq \mathcal{S}^{\text{out}}_{\mathfrak{p}}$. It remains to show that all $p \in \mathcal{S}^{\text{in}}_{\mathfrak{p}} \setminus \operatorname{crit}(\mathcal{S}^{\text{in}}_{\mathfrak{p}})$ satisfy the extension property. We can fix t, b, σ such that $p \subset S^t_{\mathfrak{p},b,\sigma}$, $|\operatorname{Dom}(p)| \leq k+1$, and $|\operatorname{Dom}(p) \cap \boldsymbol{A}^t| \leq 2$. If Spoiler asks for a variable $Z \notin \boldsymbol{A}^t$, then Duplicator answers with $z \in \{0,1\}$ such that $p \cup \{Z \mapsto z\} \in \mathcal{S}^t_{\mathfrak{p},b,\sigma}$. So we can assume that $Z \in \boldsymbol{A}^t$.

Case 1: $|\operatorname{Dom}(p) \cap \boldsymbol{A}^t| \leq 1$.

If $Z \in \operatorname{Dom}(S^t_{\mathfrak{p},b,\sigma})$, then Duplicator can answer with $z \in \{0,1\}$ such that $p \cup \{Z \mapsto z\} \in \mathcal{S}^t_{\mathfrak{p},b,\sigma}$ because $|\operatorname{Dom}(p \cup \{Z \mapsto z\}) \cap \boldsymbol{A}^t| \leq 2$. Thus, we can assume that $Z \in \boldsymbol{A}^t \setminus \operatorname{Dom}(S^t_{\mathfrak{p},b,\sigma})$.

Case 1.1: $\{A0^t_{\mathfrak{p}(t)}, A1^t_{\mathfrak{p}(t)}\} \cap \operatorname{Dom}(p) = \emptyset$.

We can fix a $c \in [4]$ such that the variable in $\operatorname{Dom}(p) \cap \boldsymbol{A}^t$ is one of the variables $A^{t,c}_{\mathfrak{p}(t)}$, $A^{t,c}_{\mathfrak{p}(t),\geq l}$ or $A^{t,c}_{\mathfrak{p}(t),l}$. If $\operatorname{Dom}(p) \cap \boldsymbol{A}^t = \emptyset$ we set

$c = 1$. Now we can flip b on partition t:

$$\hat{b}(t) := \begin{cases} 1, \text{ if } b(t) \in \{3,4\} \text{ and } c \neq 1, \\ 2, \text{ if } b(t) \in \{3,4\} \text{ and } c = 1, \\ 3, \text{ if } b(t) \in \{1,2\} \text{ and } c \neq 3, \\ 4, \text{ if } b(t) \in \{1,2\} \text{ and } c = 3. \end{cases}$$

We set $\hat{b}(i) := b(i)$ for all other $i \in [k] \setminus \{t\}$. It follows $p \cup \{Z \mapsto 0\} \in \mathcal{S}^t_{\mathfrak{p},\hat{b},\sigma}$ for all $Z \in \boldsymbol{A}^t \setminus \mathrm{Dom}(S^t_{\mathfrak{p},b,\sigma})$.

Case 1.2: $\mathrm{Dom}(p) \cap \boldsymbol{A}^t \subset \{A0^i_{\mathfrak{p}(i)}, A1^i_{\mathfrak{p}(i)}\}$.

Once more we can flip b to $\hat{b}$ ensuring that $p \cup \{Z \mapsto 0\} \in \mathcal{S}^t_{\mathfrak{p},\hat{b},\sigma}$. We let $\hat{b}(i) := b(i)$ for all $i \in [k] \setminus \{t\}$ and

$$\hat{b}(t) := \begin{cases} 1, \text{ if } b(i) = 2, \\ 2, \text{ if } b(i) = 1, \\ 3, \text{ if } b(i) = 4, \\ 4, \text{ if } b(i) = 3. \end{cases}$$

Case 2: $|\,\mathrm{Dom}(p) \cap \boldsymbol{A}^t\,| = 2$.

Case 2.1: $\mathrm{Dom}(p) \cap \boldsymbol{B} \neq \emptyset$.

Since $|\,\mathrm{Dom}(p) \cap \boldsymbol{A}^t\,| = 2$, $|\,\mathrm{Dom}(p) \cap \boldsymbol{B}\,| \geq 1$ and $|\,\mathrm{Dom}(p)| \leq k+1$, there is some partition j such that $\mathrm{Dom}(p) \cap \boldsymbol{A}^j = \emptyset$. As there are no pebbles on partition j, it becomes our new t-partition, where some variables are undefined. We define new parameters $\hat{t}$, $\hat{\sigma}$, $\hat{b}$ such that p is contained in $\mathcal{S}^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$. Let $\hat{t} := j$, $\hat{\sigma}(i) := \sigma(i)$ for $i \in [k] \setminus \{j,t\}$, $\hat{\sigma}(t) := \sigma(j)$ and $\hat{\sigma}(j) := \sigma(t)$. Furthermore, we define $\hat{b}(i) := b(i)$ for $i \in [k] \setminus \{t\}$ and will define $\hat{b}(t)$ in the sequel. Since $\mathcal{S}^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$ is defined on all variables in $\boldsymbol{A}^t$ (and $Z \in \boldsymbol{A}^t$), Duplicator can always provide an answer z for Z such that $p \cup \{Z \mapsto z\} \in \mathcal{S}^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$. Thus, we only need to show that $p \in \mathcal{S}^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$. For this, note that $|\,\mathrm{Dom}(p) \cap \boldsymbol{A}^{\hat{t}}\,| = 0 \leq 2$. Hence, it remains to show that $p \subset S^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$. Another observation is that all variables but those from partition t are mapped to the same value

in $S^t_{\mathfrak{p},b,\sigma}$ as they were mapped to in $S^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$, independent of the choice of $\hat{b}(t)$. Especially, all variables in $\boldsymbol{B}$ stay the same since $\sigma(t) = \hat{\sigma}(\hat{t})$. Thus it remains to show that p restricted to $\boldsymbol{A}^t$ is a subset of $S^{\hat{t}}_{\mathfrak{p},\hat{b},\hat{\sigma}}$. We establish this fact by flipping $b(t)$ to $\hat{b}(t)$ as follows.

Case 2.1.1: $\{A0^t_{\mathfrak{p}(t)}, A1^t_{\mathfrak{p}(t)}\} \cap \mathrm{Dom}(p) = \emptyset$.

In this case $\hat{b}(t)$ is defined to be the smallest $c \in [4]$ such that there is no pebble on a variable of the form $A^{t,c}_{\mathfrak{p}(t)}$, $A^{t,c}_{\mathfrak{p}(t),l}$ or $A^{t,c}_{\mathfrak{p}(t),\geq l}$. Such a c exists since there are exactly two pebbles in $\mathrm{Dom}(p) \cap \boldsymbol{A}^t$.

Case 2.1.2: $\{A0^t_{\mathfrak{p}(t)}, A1^t_{\mathfrak{p}(t)}\} \cap \mathrm{Dom}(p) \neq \emptyset$.

One of the two pebbles from $\mathrm{Dom}(p) \cap \boldsymbol{A}^t$ is on $A0^t_{\mathfrak{p}(t)}$ or $A1^t_{\mathfrak{p}(t)}$. If the other pebble is not on some variable $A^{t,b(t)}_{\mathfrak{p}(t)}$, $A^{t,b(t)}_{\mathfrak{p}(t),l}$ or $A^{t,b(t)}_{\mathfrak{p}(t),\geq l}$ (for some $l \in [k]$) we let $\hat{b}(t) := b(t)$. Otherwise,

$$\hat{b}(t) := \begin{cases} 1, \text{ if } b(t) = 2, \\ 2, \text{ if } b(t) = 1, \\ 3, \text{ if } b(t) = 4, \\ 4, \text{ if } b(t) = 3. \end{cases}$$

Case 2.2: $\mathrm{Dom}(p) \cap \boldsymbol{B} = \emptyset$.

Case 2.2.1: There exists a $j \in [k]$ such that $\mathrm{Dom}(p) \cap \boldsymbol{A}^j = \emptyset$.

There is no pebble on $\boldsymbol{B}$ and no pebble on $\boldsymbol{A}^j$, therefore p is also contained in $\mathcal{S}^j_{\mathfrak{p},b,\sigma}$. Since $\boldsymbol{A}^t \subseteq \mathrm{Dom}(S^j_{\mathfrak{p},b,\sigma})$, Duplicator can provide an answer z for every requested $Z \in \boldsymbol{A}^t$ such that $p \cup \{Z \mapsto z\} \in \mathcal{S}^j_{\mathfrak{p},b,\sigma}$.

Case 2.2.2: For all $i \in [k]$: $|\,\mathrm{Dom}(p) \cap \boldsymbol{A}^i\,| \geq 1$.

In this case $p \in \mathrm{crit}(\mathcal{S}^{\mathrm{in}}_{\mathfrak{p}})$ and there is nothing to show. □

Part III.

Graph Isomorphism

7. Color Refinement

In the first part of this thesis we considered homomorphisms between structures of arity 2. Now we consider *isomorphisms*, which are special homomorphism, and focus on simple undirected graphs. An isomorphism between two graphs is a bijection between their vertices that maps edges to edges and non-edges to non-edges.[1] The *graph isomorphism problem* (GI) is to determine whether there is an isomorphism between two given graphs G and H. Like the graph homomorphism problem, this problem can trivially be solved in exponential time and no polynomial time algorithm is known. However, there are some differences between the complexities of these problems. Most notably, contrary to graph homomorphism, the graph isomorphism problem is not known to be NP-complete. Moreover, it is unlikely that GI is NP-complete, as in this case the polynomial time hierarchy would collapse to its second level [Sch88]. The second fundamental difference is that graph isomorphism can be decided significantly faster than by the trivial approach. Let $n = |V(\mathsf{G})| = |V(\mathsf{H})|$ be the number of vertices of G and H. The best known algorithm to decide whether there is a homomorphism from G to H runs in time $O(2^{cn \log n})$ for some constant $c < 1$ [Wil05], which is only slightly better than trying all n^n mappings from G to H. Contrary to that, graph isomorphism can be solved in time $O(2^{\sqrt{n \log n}})$ [BL83], which is a significant improvement over the trivial approach of trying all $n!$ bijections.

Despite the differences, graph isomorphism has one thing in common with graph homomorphism: the need for heuristics to prune the search space. In this chapter we focus on *color refinement*, a simple,

[1] Formally, an isomorphism between two σ-structures A and B is a bijection $f : V(\mathsf{A}) \to V(\mathsf{B})$ such that $(x_1, \dots, x_r) \in \dot{R}^{\mathsf{A}}$ if and only if $(f(x_1), \dots, f(x_r)) \in \dot{R}^{\mathsf{B}}$ for all r-ary $\dot{R} \in \sigma$ and all $(x_1, \dots, x_r) \in V(\mathsf{A})^r$.

yet very prominent heuristic. The general idea is to color the vertices of both graphs such that every isomorphism between G and H is forced to preserve the coloring. For example, vertices of degree k in G must be mapped to vertices of degree k in H. Hence, we can color all vertices of degree k in G and H with one unique color. Such a coloring restricts the search space as we only have to consider bijections that preserve colors. Furthermore, since every isomorphism has to define a bijection on every color class, if we find a coloring where one color class contains a different number of vertices in G and H, then we know that the graphs are not isomorphic. In this case we say that the coloring distinguishes both graphs. As for the CSP and constraint propagation, there is a trade-off between running time and power of the coloring heuristic.[2] On the one hand, the heuristic should be fast and on the other hand, the coloring should distinguish many non-isomorphic graphs. Again, as it is the case for arc consistency in the area of constraint propagation, the most prominent heuristic, color refinement, is rather weak but very fast.

The idea of color refinement is to iteratively refine a coloring of a graph G (which induces a *partition* of $V(\mathsf{G})$) until a fixed point is reached. If G' and H' are an instance of the graph isomorphism problem, we apply the procedure to the disjoint union $\mathsf{G} = \mathsf{G}' \cup \mathsf{H}'$. At the beginning all vertices are colored by their degree as in the example above. Now we take two color classes, for example *green* and *red*, and have a look at the red neighbors of the green vertices. An isomorphism that matches two green vertices also has to map their red neighbors to each other. Hence, we observe that green vertices with k red neighbors in G must be mapped to green vertices with k red neighbors in H. Because of this we can split the green color class in both graphs according to their number of red neighbors. That is, for every k we use a new unique color to recolor all green vertices with k red neighbors. This procedure is iterated until for every pair of colors all vertices of the first color have the same number of neighbors in the second color. The resulting partition is known as the coarsest stable partition of G. It can be shown that this partition is unique,

[2] cf. Section 2.1

see Proposition 7.3. Thus, the goal of the color refinement is to solve the following problem.

Coarsest Stable Partition Problem (CSPP)	
Input:	A graph G encoded as adjacency list.
Output:	The coarsest stable partition of G.

In Section 7.2 we show that the CSPP can be solved in time $O(m \log n)$ on graphs with n vertices and m edges.[3] This result goes back to [CC82] and relies on Hopcroft's algorithm for minimizing finite automata [Hop71]. We investigate the question whether it is possible to solve the CSPP in linear time. Our main theorem shows that, under weak assumptions on the structure of the algorithm, this is not possible: there is a family of graphs such that every *refinement algorithm*[4] takes $\Omega(m \log n)$ computation steps to find the coarsest stable partition.

7.1. Preliminaries

We start this section with some basic terminology for graphs. Recall that a (simple undirected) graph G is a $\{\dot{E}\}$-structure, where $\dot{E}$ is a binary relation symbol and $\dot{E}^{\mathsf{G}}$ is symmetric and anti-reflexive. The elements of the domain $V(\mathsf{G})$ of G are called *vertices*. An *edge* is a set of two vertices $\{v, w\}$ such that $(v, w) \in \dot{E}^{\mathsf{G}}$. We denote the set of edges of G by $E(\mathsf{G})$ (thus, $|E(\mathsf{G})| = \frac{1}{2}|\dot{E}^{\mathsf{G}}|$). We let $n := |V(\mathsf{G})|$ and $m := |E(\mathsf{G})|$ be the number of vertices and edges, respectively. If $\{v, w\}$ is an edge, then v is *adjacent to* or *a neighbor of* w. $N(v)$ is the set of neighbors of a vertex v. A vertex v is isolated if $N(v) = \emptyset$. We assume that a graph is given in adjacency list encoding: a list of vertices and for each vertex a list of its neighbors. Note that the size of the adjacency list encoding equals the size of the structure $\|\mathsf{G}\|$ as

[3] To exclude pathological examples we always assume that a graph has no isolated vertices. Hence, $n \leq 2m \leq n^2 - n$.

[4] Formally defined in Definition 8.2.

defined on page 23. From now on we only consider graphs without isolated vertices. Thus, the size of the encoding is proportional to the number of edges.

A partition π of a set V is a set $\{S_1, \dots, S_k\}$ of pairwise disjoint nonempty subsets of V, such that $\cup_{i=1}^{k} S_i = V$. The sets S_i are called *cells* of π. The *discrete* partition of V is $\big\{\{v\} \mid v \in V\big\}$ and the *unit* partition of V is $\{V\}$. A partition π of V is *discrete on* $S \subseteq V$ if $\{v\} \in \pi$ for all $v \in S$. Given a partition π of V, and two elements $u, v \in V$, we write $u \approx_\pi v$ if there exists a cell $S \in \pi$ with $u, v \in S$. Let G be a graph. A partition π of $V(\mathsf{G})$ is *stable* for G if for every pair of vertices $u, v \in V(\mathsf{G})$ with $u \approx_\pi v$ and $R \in \pi$, it holds that $|N(u) \cap R| = |N(v) \cap R|$.

A partition ρ of V *refines* a partition π of a subset S of V if for every $u, v \in S$, $u \approx_\rho v$ implies $u \approx_\pi v$. (Usually we take $S = V$.) If ρ refines π, we write $\pi \preceq \rho$. If in addition $\rho \neq \pi$, then we also write $\pi \prec \rho$. Note that $\preceq$ is a partial order on all partitions of V. The next definition formalizes the operation "split color class S according to the number of neighbors in color class R." For technical reasons we also allow R to be a union of color classes.

Definition 7.1. Let G be a graph, and let π and π' be partitions of $V(\mathsf{G})$. For a cell $S \in \pi$ and a union of cells $R = R_1 \cup \dots \cup R_\ell$, $R_i \in \pi$, we say that π' is obtained from π by a *refining operation* (R, S) if

- for every $S' \in \pi \setminus \{S\}$, it holds that $S' \in \pi'$, and
- for every $u, v \in S$: $u \approx_{\pi'} v$ if and only if $|N(u) \cap R| = |N(v) \cap R|$.

The refining operation (R, S) is *elementary*, if $R \in \pi$.

Note that if π' is obtained from π by a refining operation (R, S), then $\pi \preceq \pi'$. We say that the operation (R, S) is *effective* if $\pi \prec \pi'$. In this case, the cell $S \in \pi$ is *split*. Note that an effective refining operation exists for π if and only if π is unstable. In addition, the next proposition says that if the goal is to obtain a (coarsest) stable partition, then applying any refining operation is safe.

Proposition 7.2. *Let π' be obtained from π by a refining operation (R, S). If ρ is a stable partition with $\pi \preceq \rho$, then $\pi \preceq \pi' \preceq \rho$.*

Proof. $\pi \preceq \pi'$ follows immediately from the definitions. Now consider u, v with $u \approx_\rho v$, and thus $u \approx_\pi v$. Then for any $R' \in \pi$, $|N(u) \cap R'| = |N(v) \cap R'|$. This holds because R' is a union of sets in ρ, and for all these this property holds since ρ is stable. Therefore, $u \approx_{\pi'} v$. □

A partition π is a *coarsest* partition for a property P if π satisfies P, and there is no partition ρ with $\rho \prec \pi$ that also satisfies property P.

Proposition 7.3. *Let* G *be a graph. For every partition* π *of* $V(\mathsf{G})$*, there is a unique coarsest stable partition* ρ *that refines* π*.*

Corollary 7.4. *For every graph* G *there is a unique coarsest stable partition that refines the vertex set. We denote this partition by* $\omega(\mathsf{G})$*.*

Proof of Proposition 7.3. For any partition π, the discrete partition refines π and is stable, so there exists a stable partition that refines π. Because $\preceq$ is a partial order, there exists at least one coarsest stable partition that refines π. Now suppose there exists a partition π for which there exist at least two distinct coarsest stable partitions ρ_1 and ρ_2 that refine π. Choose such a partition π so that $|\pi|$ is maximum. Clearly, π is not stable (otherwise $\rho_1 = \pi = \rho_2$). So there exists at least one effective refining operation (R, S) that can be applied to π. For the resulting partition π', $|\pi'| > |\pi|$ holds. By Proposition 7.2, both ρ_1 and ρ_2 refine π' as well. But since $|\pi'| > |\pi|$, this contradicts the choice of π.

□

7.2. Algorithms for Color Refinement

By Proposition 7.3 and Corollary 7.4 we know that in order to find the coarsest stable partition it suffices to perform a sequence of refining operations. In fact, all known algorithms solving the CSPP use this approach. In this section we give a high level description of refinement strategies for color refinement algorithms. We start with the naive approach and end up with the strategy that uses Hopcroft's idea to obtain quasilinear running time. The first attempt to implement color

refinement is to iteratively refine all possible cells S with respect to the number of neighbors in all cells R.

Algorithm 1

$\pi \leftarrow \{V(\mathsf{G})\}$
repeat
  if there is an effective elementary operation (R, S) **then**
    $\pi \leftarrow \pi(R, S)$
until π is stable.

Since a partition of n vertices can be refined at most $n - 1$ times, we iterate the loop at most $O(n)$ times. Furthermore, we can check all possible refining cells R against all possible splitting cells S by computing for every vertex the number of neighbors in every color class R. Using adjacency lists, this can be implemented in time linear in the number of vertices n plus the number of edges m. This running time is bounded by $O(m)$, as there are no isolated vertices. Hence, the overall running time of this first approach is $O(nm)$. In each step we check for all pairs (R, S) whether the operation is effective. This might be wasteful because even if some pair of cells R, S does not change, it is checked multiple times whether S can be refined with R. Indeed, if we already have refined against one refining cell R we do not need to refine against R until R itself splits up. The next algorithm uses this fact for improvement. We maintain one `stack` of refining cells and iteratively refine all other cells with the refining cell popped from the `stack`. If some cell splits up, we add all the new cells to the `stack` (and possibly delete the old one).

Note that at the moment when we test whether (R, S) is effective, R may no longer be a cell of the current partition, as it could have been split via (R, R) in the same loop. Because of this phenomenon, we also allow R to be a set of cells in Definition 7.1. The running time analysis of this algorithm is again easy. We can compute for

Algorithm 2

```
π ← {V(G)}
stack ← a stack containing V(G)
while stack is not empty do
    R ← pop(stack)
    for all S ∈ π do
        if (R, S) is effective then
            Split S into S = S_1 ∪ ··· ∪ S_ℓ according to (R, S).
            π ← π(R, S).
            Delete S from the stack.
            for all S_i do
                push S_i on the stack.
```

every vertex in S the number of neighbors in a color class R in time proportional to $\sum_{v \in S} \deg(v)$ by running over the adjacency list of the vertices in S. Thus, we can split every color class S with respect to one color class R in time $O(m)$. Furthermore, we push at most n cells on the stack. Hence, the **while** loop is executed at most n times and the overall running time is $O(nm)$. Again we have quadratic running time and need a further trick to get a quasilinear algorithm.

Note that if a cell S is not on the `stack`, then every pair of vertices in every cell of G has the same number of neighbors in S. If one cell S not on the stack splits up into $S_1, \ldots, S_\ell$, we argue that it suffices to refine against all but one of the smaller cells S_i. Assume that two vertices get distinguished because they have a different number of neighbors in some S_i. Since they shared the same cell before, we know that they have the same number of neighbors in S. Hence, there must be a $j \neq i$ such that they have a different number of neighbors in color class S_j. Therefore, we can choose an arbitrary S_i that is not used for refinement. To obtain maximum speed-up in running time, we ignore the largest cell. The resulting refinement strategy is exactly the same as in Hopcroft's algortihm for minimizing finite automata [Hop71] and leads to quasilinear running time.

Algorithm 3

$\pi \leftarrow \{V(G)\}$
`stack` $\leftarrow$ a stack containing $V(G)$
while `stack` is not empty **do**
  $R \leftarrow$ pop(`stack`)
  for all $S \in \pi$ **do**
    if (R, S) is effective **then**
      Split S into $S = S_1 \cup \cdots \cup S_\ell$ according to R.
      $\pi \leftarrow \pi(R, S)$.
      if $S \in$ `stack` **then**
        replace S by $S_1, \ldots, S_\ell$ on the `stack`.
      else
        $S_{\max} \leftarrow S_j$ s.t. $|S_j| = \max_i |S_i|$
        for all $S_i \neq S_{\max}$ **do**
          push S_i on the `stack`.

We have already discussed the correctness of this algorithm and it remains to analyze the running time. As in the previous algorithm, the **for** loop takes $O(\sum_{v \in S} \deg(v))$. Let $\mathcal{R}$ be the set of cells R that are popped from stack during the execution of the algorithm. Then the overall running time of the algorithm is proportional to $\sum_{R \in \mathcal{R}} \sum_{v \in R} \deg(v)$. Let $\text{stack}(v) := |\{R \in \mathcal{R} \mid v \in R\}|$ be the number of times the vertex v occurs in a cell R popped from the stack. Assume that v is contained in a cell R_1 popped from the stack and later on contained in a cell R_2 popped from the stack. This can only happen if R_1 was split up in a refinement step and R_2 is one of the (but not the largest!) sub-cell of R_1. Hence, $2|R_2| \leq R_1$ and therefore $\text{stack}(v) \leq \log n$. Consequently, the running time is asymptotic to

$$\sum_{R \in \mathcal{R}} \sum_{v \in R} \deg(v) = \sum_{v \in V(G)} \text{stack}(v) \deg(v)$$
$$\leq \log n \sum_{v \in V(G)} \deg(v) = O(m \log n).$$

8. Lower Bounds for Refinement Algorithms

All refinement strategies start with the unit partition $\{V(\mathsf{G})\}$ and try to iteratively apply splitting operations (R, S) until they end up with the coarsest stable partition $\omega(\mathsf{G})$. Thus, every run of such a splitting procedure corresponds to a sequence of refining operations. To formalize this, we call $((R_1, S_1), \ldots, (R_\ell, S_\ell))$ a *refining sequence* if there exists a sequence of partitions $(\pi_1, \ldots, \pi_{\ell+1})$ of $V(G)$ such that

- $\pi_1 = \{V(G)\}$ is the unit partition and $\pi_{\ell+1} = \omega(\mathsf{G})$ is the coarsest stable partition,
- and for all $i \leq \ell$ it holds that $\pi_{i+1} = \pi_i(R, S)$.

Note that we do not require a refining operation (R_i, S_i) to be effective, as this sequence also contains the splitting operations an algorithm tries to apply. We define the *cost* of a refining operation (R, S) to be the number of edges between R and S in G, where edges in $R \cap S$ are counted twice. This definition is motivated by the observation that if an algorithms wants to apply (R, S), it has to compute the number of neighbors within R for every element in S. Hence, every edge between R and S has to be considered.

Definition 8.1. Let G be a graph and $\mathfrak{r}$ be a refining sequence for G. The *cost* of a refining operation $(R, S) \in \mathfrak{r}$ is defined as

$$\operatorname{cost}(R, S) = |\{(u, v) \in R \times S \mid \{u, v\} \in E(\mathsf{G})\}|$$

and the cost of the refining sequence $\operatorname{cost}(\mathfrak{r}) = \sum_{(R,S)\in\mathfrak{r}} \operatorname{cost}(R, S)$ is the sum of costs over all refining operations. By $\operatorname{cost}(\mathsf{G}) = \min_{\mathfrak{r}} \operatorname{cost}(\mathfrak{r})$

we denote the *refining cost* of G, which is the minimal cost over all refining sequences for G.

To obtain lower bounds we have to add two basic assumptions on the structure of color refinement algorithms. They are formalized in the next definition.

Definition 8.2. A *refinement algorithm* is an algorithm that solves the CSPP by performing a sequence $\mathfrak{r}$ of refining operations. Furthermore, executing a refining operation $(R, S) \in \mathfrak{r}$ requires $\Omega(\text{cost}(R, S))$ computation steps.

Note that all known color refinement algorithms fall under this definition. Moreover, an algorithm solving the CSPP which does not fit into this definition would have to implement a completely different approach. By our definitions it follows that the running time of refinement algorithms applied to a graph G is lower bounded by the refinement cost of G. However, the refinement cost of G does not serve as an upper bound. This is because we do not know how to find a cost-optimal refining sequence, which in particular contains only effective refining operations. Furthermore, it is not clear how to perform an arbitrary refining operation (R, S) in time $O(\text{cost}(R, S))$. The analysis of Algorithm 7.2 shows how to compute a refining sequence of cost $m \log n$.

Corollary 8.3. *There is a refinement algorithm that executes for every graph G a refining sequence of cost at most $m \log n$ and runs in time $O(m \log n)$.*

Hence, the refining cost of a graph G is always bounded by $m \log n$. The main theorem of this chapter states that this bound is tight. It immediately follows, that the refining sequence computed by Algorithm 7.2 has optimal cost and hence no improvement is possible.

Theorem 6. *For every integer $k \geq 2$, there is a connected graph G_k with $n \in O(2^k k)$ vertices and $m \in O(2^k k^2)$ edges, such that* $\text{cost}(\mathsf{G}) = \Omega(m \log n)$.

Corollary 8.4. *There is no refinement algorithm for the* CSPP *of running time $o(m \log n)$.*

8.1. Toolbox for Refining Sequences

We first show that there is always a cost-optimal refining sequence which only contains elementary refining operations.

Lemma 8.5. *For every graph* G *there is a refining sequence* $\mathfrak{r}$ *that contains only elementary refining operations and satisfies* $\mathrm{cost}(\mathfrak{r}) = \mathrm{cost}(\mathsf{G})$.

Proof. Let $\mathfrak{r}$ be an arbitrary sequence of effective refining operations for G that satisfies $\mathrm{cost}(\mathfrak{r}) = \mathrm{cost}(\mathsf{G})$. Hence, the length ℓ of $\mathfrak{r}$ is bounded by $n-1$. Let i be the maximal index such that the first i refining operations are elementary. We prove the lemma by induction over $i = n \dots 1$. If $i = n$ then there is nothing to show (as $\ell < n$). Assume that (R, S) is the first non-elementary refining operation, where $R = R_1 \cup \dots \cup R_p$ is a union of cells R_i of the current partition π. We replace (R, S) by the sequence $(R_1, S), \dots, (R_p, S)$. Note that by applying these refining operations one by one, the resulting partition refines $\pi(R, S)$. Thus, the new sequence is "more effective" and of the same or lower cost. Because of this modification it can happen, that for some tuple (R, S) in the new sequence the set S is a union of current cells $S_1, \dots, S_q$. To fit the definitions, we need to replace every tuple of this form by the sequence $(R, S_1), \dots, (R, S_q)$. Splitting up the refining operations in this way does not increase the cost of the refining sequence. Finally, we delete all non-effective operations from the sequence. Since the elementary operation (R_1, S) at position i is elementary, it enlarges the initial sequence of elementary operations. Therefore, the lemma follows by induction. □

Finding a cost-optimal refining sequence of elementary operations can be seen as a one-player game. The game board is the graph and a configuration of the game is a partition π of the vertices together with a cost counter. Starting with the unit partition, the player picks two (not necessarily different) cells $R, S \in \pi$, adds $\mathrm{cost}(R, S)$ to the cost counter and applies the refining operation (R, S) to π. If the current partition is the unique coarsest stable partition of G, then the game ends and the score is the final value of the cost counter.

The aim of the player is to minimize the score. By Lemma 8.5, the optimal score of the game is $\mathrm{cost}(\mathsf{G})$. To prove our lower bound (Theorem 6) we have to design a graph G such that *every* refinement strategy ends up with a score of at least $\Omega(m \log n)$. To obtain this we introduce a second player. Again, the game board is a graph G and the configurations consist of a partition of the vertices and a cost counter. In each round Player 1 picks cells R, S and adds $\mathrm{cost}(R, S)$ to the cost counter, then Player 2 chooses a partition π' such that $\pi(R, S) \preceq \pi' \preceq \omega(\mathsf{G})$. The new configuration is partition π' together with the updated cost counter. As above, the game ends when current partition is $\omega(\mathsf{G})$ and the score is the final value of the cost counter. Note that by Proposition 7.2 and Proposition 7.3 it is guaranteed that in each step the current partition is either unstable (hence an effective refining operation can be applied) or it is the coarsest stable partition $\omega(\mathsf{G})$. The goal of Player 1 is to minimize the score and the goal of Player 2 is to maximize the score. We first show that Player 1 is still able to obtain a score of at most $\mathrm{cost}(\mathsf{G})$. Hence, an optimal strategy of Player 2 is to choose $\pi' = \pi(R, S)$ in every round.

Lemma 8.6. *Player 1 is able to limit the score in the refinement game to* $\mathrm{cost}(\mathsf{G})$.

Corollary 8.7. *If Player 2 has strategy to obtain a score of s in the refinement game on G, then $s \leq \mathrm{cost}(\mathsf{G})$.*

Proof of Lemma 8.6. Let G be a graph and $\mathfrak{r}$ a refining sequence with $\mathrm{cost}(\mathfrak{r}) = \mathrm{cost}(\mathsf{G})$. Player 1's initial strategy is to apply this optimal refining strategy. Since Player 2 has the possibility to switch to finer partitions in every step, applying a refinement operation (R, S) according to the current strategy might not be possible, because R or S are unions of cells of the current partition. However, Player 1 can split (R, S) into elementary operations in the same way as in the proof of Lemma 8.5 without increasing the cost of his strategy. □

One might wonder why we care about Player 2 since her optimal strategy is obvious. The drawback of following Player 2's optimal strategy is that, in order to prove lower bounds, we have to argue

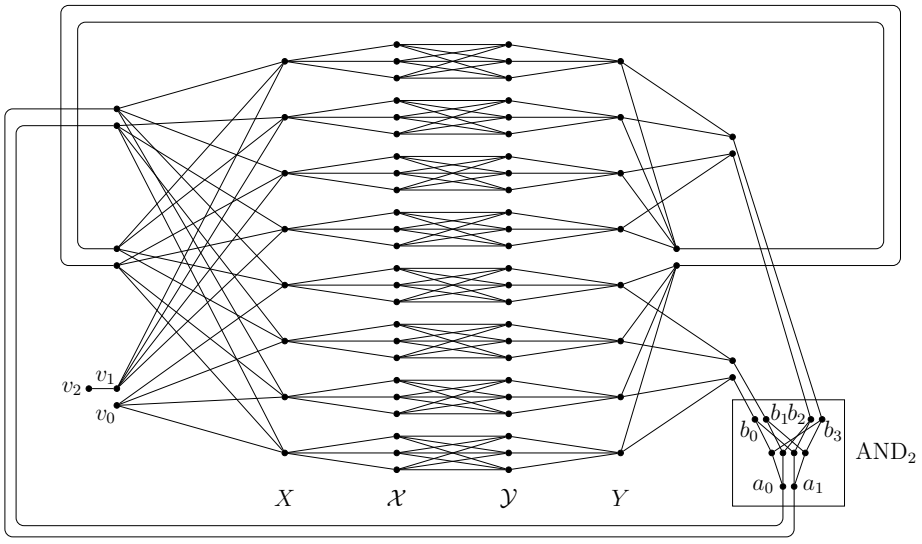


Figure 8.1.: The Graph G_3.

about the cost of all possible refining sequences Player 1 may choose. To simplify the arguments, we instead design a non-optimal strategy of Player 2 where she always chooses some π' that we know in advance. Furthermore, the partitions π' Player 2 chooses have the property that the next effective refinement operation is expensive. Since we know all π' and the cost of splitting a cell in π', we can easily derive a lower bound from this strategy using Corollary 8.7.

8.2. Construction of the Graph

To prove Theorem 6, we design for every integer $k \geq 2$ a graph G_k with $n = O(k2^k)$ vertices and $m = O(k^2 2^k)$ edges. Then we show that on G_k Player 2 is able to obtain a score of $\Omega(k^3 2^k)$. This proves the theorem by Corollary 8.7. For $k \in \mathbb{N}$, denote $\mathcal{B}_k = \{0, \ldots, 2^k - 1\}$. For $\ell \in \{0, \ldots, k\}$ and $q \in \{0, \ldots, 2^\ell - 1\}$, the subset $\mathcal{B}_q^\ell = \{q2^{k-\ell}, \ldots, (q+1)2^{k-\ell} - 1\}$ is called the *q-th binary block of level ℓ*. Analogously, for any set of vertices with indices in $\mathcal{B}_k$, we also consider binary blocks. For instance, if $X = \{x_i \mid i \in \mathcal{B}_k\}$, then $X_q^\ell = \{x_i \mid i \in \mathcal{B}_q^\ell\}$ is called a

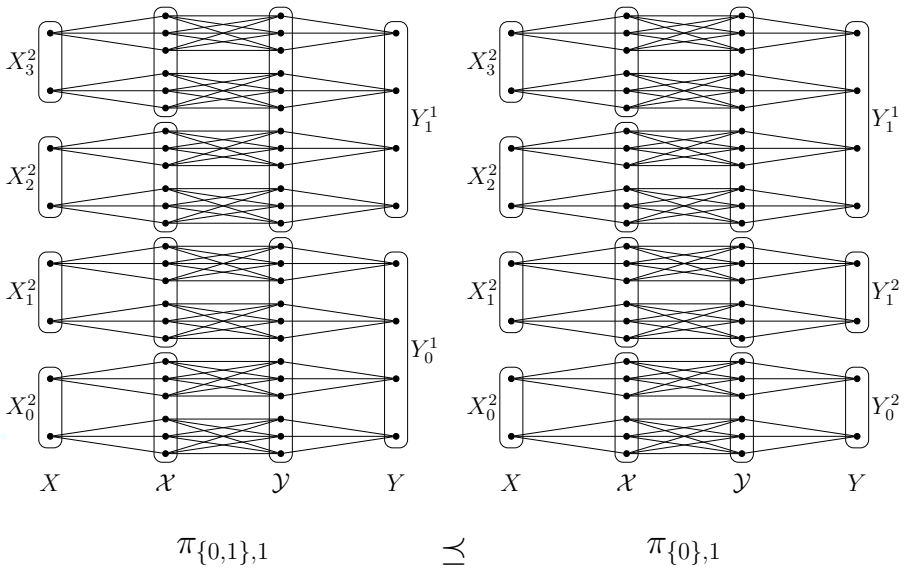


Figure 8.2.: Two subgraphs of G_k for $k = 3$ with partitions of level $\ell = 1$ (see Definition 8.10). The next effective refinement operation on these partitions is expensive, the costs are $k^2 2^{k-(\ell+1)} = 18$.

binary block of X. For such a set X, a partition π of X *into binary blocks* is a partition where every $S \in \pi$ is a binary block. A key fact for binary blocks that we will often use is that for any ℓ and q, $\mathcal{B}_q^\ell = \mathcal{B}_{2q}^{\ell+1} \cup \mathcal{B}_{2q+1}^{\ell+1}$.

In its core the graph G_k consists of the vertex sets $X = \{x_i \mid i \in \mathcal{B}_k\}$, $\mathcal{X} = \{x_i^j \mid i \in \mathcal{B}_k, j \in [k]\}$, $\mathcal{Y} = \{y_i^j \mid i \in \mathcal{B}_k, j \in [k]\}$ and $Y = \{y_i \mid i \in \mathcal{B}_k\}$. Every vertex x_i is adjacent to x_i^j for all $j \in [k]$ and every y_i is adjacent to all y_i^j. Furthermore, for all i, j_1, j_2 there is an edge between $x_i^{j_1}$ and $y_i^{j_2}$. For $\mathcal{X}$, *binary blocks* are subsets of the form $\mathcal{X}_q^\ell := \{x_i^j \mid i \in \mathcal{B}_q^\ell, j \in [k]\}$, and for $\mathcal{Y}$ the definition is analogous. For example, in the partition $\pi_{\{0,1\},1}$ of G_3 shown in Figure 8.2, X and $\mathcal{X}$ are partitioned into blocks of level 2, and Y, $\mathcal{Y}$ are partitioned into blocks of level 1.

We add gadgets to the graph to ensure that any sequence of refining operations behaves as follows. After the first step, which distinguishes vertices according to their degrees, X and Y are cells of the resulting partition. Next, X splits up into two binary blocks X_0^1 and X_1^1 of equal size. To ensure this, there are two vertices, v_0 and v_1, of different degree which are connected to all vertices from X_0^1 and X_1^1, respectively (see Figure 8.1). Splitting X_0^1 and X_1^1 in such a way causes $\mathcal{X}$ to split up accordingly into $\mathcal{X}_0^1$ and $\mathcal{X}_1^1$. One of these cells, say $\mathcal{X}_0^1$, will be used to halve $\mathcal{Y}$ in the same way. This refining operation $(\mathcal{X}_0^1, \mathcal{Y})$ is expensive because the number of edges between the refining and the splitting set are half of the edges between $\mathcal{X}$ and $\mathcal{Y}$. Next, Y can be split up into Y_0^1 and Y_1^1. Once this happens, there is a gadget AND_1 that causes the two cells X_0^1, X_1^1 to split up into the four cells X_q^2, for $q = 0, \ldots, 3$. Again, this causes cells in $\mathcal{X}, \mathcal{Y}$ and Y to split up in the same way and to achieve this, half of the edges between $\mathcal{X}$ and $\mathcal{Y}$ have to be considered (see Figure 8.2). The next gadget AND_2 ensures that if both cells of Y are split, then the four cells of X can be halved again, etc. In general, we design a gadget AND_ℓ of level ℓ that ensures that if Y is partitioned into 2^ℓ binary blocks of equal size, then X can be partitioned into $2^{\ell+1}$ binary blocks of equal size. By halving all the cells of X and Y $k = \Theta(\log n)$ times, this refinement process ends up with a discrete coloring of these vertices. Since every iteration uses half of the edges between $\mathcal{X}$ and $\mathcal{Y}$

(which are $\Theta(m)$), we get the cost lower bound of $\Omega(m \log n)$.

We now describe the use of the AND-gadgets in more detail. The gadgets themselves are provided in the next section. For every integer $\ell = 1 \dots k-1$, the gadget AND_ℓ contains a pair of *out-terminals* a_0, a_1 and a sequence of $2^{\ell-1}$ pairs of *in-terminals* b_{2q}, b_{2q+1} (for $q = 0, \dots, 2^{\ell-1}-1$). Every in-terminal pair b_{2q}, b_{2q+1} is used to detect whether the block $Y_q^{\ell-1}$ has been split into Y_{2q}^ℓ and Y_{2q+1}^ℓ. To ensure this, there are edges between b_i and all vertices from Y_i^ℓ. Hence, if $Y_q^{\ell-1}$ is a cell in the current partition, then b_{2q} and b_{2q+1} have the same number of neighbors in $Y_q^{\ell-1}$. But as soon as $Y_q^{\ell-1}$ splits into Y_{2q}^ℓ and Y_{2q+1}^ℓ, the in-terminals b_{2q} and b_{2q+1} can be distinguished. Therefore, all in-terminals of AND_ℓ can be distinguished if and only if the current partition divides Y into blocks of level ℓ. The main property of the AND-gadgets is that the out-terminal pair a_0, a_1 will be distinguished if and only if *all* in-terminal pairs are distinguished. As described above, X is divided into binary blocks of level ℓ in this situation and we want that these blocks split into binary blocks of level $\ell+1$. To achieve this, we connect a_0 to all vertices $X_{2q}^{\ell+1}$ and a_1 to all vertices $X_{2q+1}^{\ell+1}$ for every block $X_q^\ell = X_{2q}^{\ell+1} \cup X_{2q+1}^{\ell+1}$. Hence, if a_0 and a_1 get different colors, then every block X_q^ℓ can be split into $X_{2q}^{\ell+1}$ and $X_{2q+1}^{\ell+1}$. The next lemma formalizes the properties of the AND-gadgets. It states that if all input vertices are distinguished, then Player 1 can distinguish the out-terminals. Furthermore, it provides stable partitions which do not distinguish the output pair if some set S of input pairs is not distinguished. These partitions will be used in Player 2's strategy.

Lemma 8.8. *Let A be an AND_ℓ-gadget with in-terminals $b_0, \dots, b_{2^\ell-1}$ and out-terminals a_0, a_1. If π is a partition of $V(\mathsf{A})$ that is discrete on all input-vertices, then the coarsest stable partition that refines π is the discrete partition of $V(\mathsf{A})$. Furthermore, for every non-empty set $S \subseteq \mathcal{B}_{\ell-1}$ there is a stable partition $\rho_{S,\ell}$ of $V(\mathsf{A})$ such that*

- $\{b_{2q}^\ell, b_{2q+1}^\ell\} \in \rho_{S,\ell}$ *if* $q \in S$,
- $\{b_{2q}^\ell\}, \{b_{2q+1}^\ell\} \in \rho_{S,\ell}$ *if* $q \in \mathcal{B}_{\ell-1} \setminus S$,

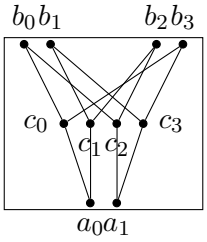


Figure 8.3.: This Cai-Fürer-Immerman gadget defines AND_2 and forms the basic building block of our construction.

- $\{a_0, a_1\} \in \rho_{S,\ell}$ *and*
- *for all* $S \supseteq S'$*:* $\rho_{S,\ell} \preceq \rho_{S',\ell}$.

8.3. The Gadgets

Now we define the AND-gadgets in detail and prove Lemma 8.8. For every integer $\ell \geq 1$, the gadget AND_ℓ is a graph A with two vertices a_0, a_1 called *out-terminals*, and an ordered sequence of $p = 2^\ell$ *in-terminals* $b_0, \ldots, b_{p-1}$. For $\ell = 1$, the graph A has $V(\mathsf{A}) = \{a_0, a_1, b_0, b_1\}$, and $E(\mathsf{A}) = \{\{a_0, b_0\}, \{a_1, b_1\}\}$. For $\ell = 2$, the graph A is identical to the construction of Cai, Fürer and Immerman [CFI92] (see Figure 8.3). The out-terminals a_0, a_1 and in-terminals $b_0, \ldots, b_3$ are indicated. For $\ell \geq 3$, AND_ℓ is obtained by taking one copy A_2 of an AND_2-gadget, and two copies A_0 and A_1 of an $\mathrm{AND}_{\ell-1}$-gadget, and adding four edges to connect the two pairs of in-terminals of A_2 with the pairs of out-terminals of A_0 and A_1, respectively (see Figure 8.4). As out-terminals of the resulting gadget we choose the out-terminals of A_2. The in-terminal sequence is obtained by concatenating the sequences of in-terminals of A_0 and A_1. For any AND_ℓ-gadget A with in-terminals $b_0, \ldots, b_{2^\ell-1}$, the *in-terminal pairs* are pairs b_{2p} and b_{2p+1}, for all $p \in \{0, \ldots, 2^{\ell-1} - 1\}$. Note that the number of vertices and the number of edges of an AND_ℓ gadget is $O(2^\ell)$.

Proof of Lemma 8.8. We prove the statement by induction over ℓ.

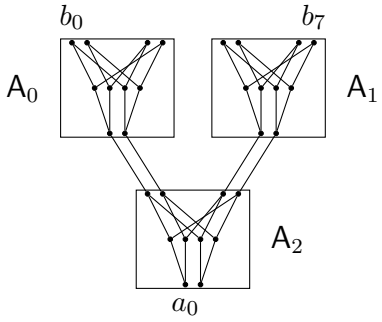


Figure 8.4.: AND_3

For $\ell = 1$, the statement is trivial. Now suppose $\ell = 2$ (see Figure 8.3). Suppose that π is a partition of $V(\mathsf{A})$ such that $\{b_0\}, \ldots, \{b_3\} \in \pi$. Because of the connection to the b_i vertices, we can partition the remaining vertices into $\{c_0\}$,$\{c_1\}$,$\{c_2\}$,$\{c_3\}$,$\{a_0, a_1\}$. Since $N(a_0) = \{c_0, c_1\}$ and $N(a_1) = \{c_2, c_3\}$, we end up with the discrete partition. Now we define the stable partitions $\rho_{S,2}$. Using Figure 8.3 it can easily be checked that the partitions are stable. Moreover, they fulfill the requirements of the lemma by definition.

$$\begin{aligned}
\rho_{\{0,1\},2} &= \big\{\{b_0, b_1\}, \{b_2, b_3\}, && \{c_0, c_1, c_2, c_3\}, && \{a_0, a_1\}\big\} \\
\rho_{\{0\},2} &= \big\{\{b_0, b_1\}, \{b_2\}, \{b_3\}, && \{c_0, c_3\}, \{c_1, c_2\}, && \{a_0, a_1\}\big\} \\
\rho_{\{1\},2} &= \big\{\{b_0\}, \{b_1\}, \{b_2, b_3\}, && \{c_0, c_2\}, \{c_1, c_3\}, && \{a_0, a_1\}\big\}
\end{aligned}$$

Now suppose $\ell \geq 3$ and let A be an AND_ℓ gadget. Recall that A consists of two $\mathrm{AND}_{\ell-1}$ gadgets, A_0 and A_1, whose out-terminals are pairwise connected to the in-terminals of an additional AND_2 gadget A_2. Now suppose that a partition π of $V(\mathsf{A})$ is discrete on all in-terminals. By induction, the coarsest stable partition that refines π is discrete on both A_0 and A_1. Moreover, it is discrete on the union $\mathsf{A}_0 \cup \mathsf{A}_1$. Since the in-terminals of the AND_2 gadget A_2 are adjacent to different vertices from $\mathsf{A}_0 \cup \mathsf{A}_1$, they get different colors. Hence, the coarsest stable partition that refines this partition is discrete on A_2 by the base case. Thus, it is the discrete partition of $V(\mathsf{A})$. To define

the stable partitions $\rho_{S,\ell}$ let $S_0 := S \cap \mathcal{B}_{\ell-1}$, $S_1 := \{i \mid i + 2^{\ell-1} \in S\}$ and $S_2 := \{i \in \{0,1\} \mid S_i = \emptyset\}$. Let $i \in \{0,1\}$. If $S_i = \emptyset$, we let π_i be the discrete partition of $V(\mathsf{A}_i)$. Otherwise we let π_i be the stable partition $\rho_{S_i,\ell-1}$ of $V(\mathsf{A}_i)$. Since S is nonempty, either S_0 or S_1 is nonempty and hence $S_2 \neq \emptyset$. Thus, there is a stable partition $\pi_2 := \rho_{S_2,2}$ of $V(\mathsf{A}_2)$. Finally, we define $\rho_{S,\ell} := \pi_0 \cup \pi_1 \cup \pi_2$. As $\rho_{S,\ell}$ splits the two out-terminals of A_0 and A_1 if and only if it splits the corresponding in-terminal pair of A_2, it follows that the partition is stable. Furthermore, $\rho_{S,\ell}$ satisfies the properties of the lemma by definition.

□

8.4. Proof of the Lower Bound

Lemma 8.9. *The coarsest stable partition $\omega(\mathsf{G}_k)$ contains the cells $\{x_i^j \mid j \in [k]\} \subseteq \mathcal{X}$ and $\{y_i^j \mid j \in [k]\} \subseteq \mathcal{Y}$ for all $i \in \mathcal{B}_k$. Furthermore, $\omega(\mathsf{G}_k)$ is discrete on $V(\mathsf{G}) \setminus (\mathcal{X} \cup \mathcal{Y})$.*

Proof. We show by induction on $i = 1 \dots k$ that starting from a partition ...

(a) ... that partitions X into binary blocks of level i, Player 1 can split $\mathcal{X}, \mathcal{Y}, Y$ into binary blocks of level i.

(b) ... that partitions Y into binary blocks of level i, Player 1 can reach the discrete partition on AND_i.

(c) ... that is discrete on AND_i, Player 1 can split X into binary blocks of level $i+1$.

Starting with $(V(\mathsf{G}), V(\mathsf{G})$ and $(\{v_0\}, X)$, Player 1 can split up X into blocks of level 1. Thus, the lemma follows from (a)–(c) by induction. Statement (a) follows by using the cells in X for refining $\mathcal{X}$, the cells in $\mathcal{X}$ for refining $\mathcal{Y}$ and finally the cells in $\mathcal{Y}$ for refining Y. Since the blocks of level i in Y are the neighborhoods of the in-terminals b_j^i of the AND_i gadget, Player 1 can reach the discrete partition on them. By Lemma 8.8 he can therefore reach the discrete partition

on the entire gadget. This proves (b). To verify (c), recall that the out-terminals a_0^i and a_1^i of AND_i are connected to X in such a way that $(\{a_0^i\},X_q^i)$ splits X_q^i into X_{2q}^{i+1} and X_{2q+1}^{i+1} for every $q \in \mathcal{B}_i$. □

Now we can define the partitions $\pi_{Q,\ell}$ for Player 2's strategy in the refinement game. In every such partition the vertex sets X and $\mathcal{X}$ consist of blocks of level $\ell+1$. The blocks of Y and $\mathcal{Y}$ with index in Q are not split (of level ℓ) and the remaining blocks of Y and $\mathcal{Y}$ are already split (of level $\ell+1$).

Definition 8.10. For any $\ell \in \{0,\dots,k-1\}$, and nonempty set $Q \subseteq \mathcal{B}_\ell$, by $\pi_{Q,\ell}$ we denote the partition of $V(\mathsf{G})$ that contains cells

- $X_q^{\ell+1}$ and $\mathcal{X}_q^{\ell+1}$ for all $q \in \mathcal{B}_{\ell+1}$,
- Y_q^ℓ and $\mathcal{Y}_q^\ell$ for all $q \in Q$, and
- $Y_{2q}^{\ell+1}, Y_{2q+1}^{\ell+1}$ and $\mathcal{Y}_{2q}^{\ell+1}, \mathcal{Y}_{2q+1}^{\ell+1}$ for all $q \in \mathcal{B}_\ell \setminus Q$.

Furthermore, $\pi_{Q,\ell}$ partitions the vertices of the AND_t gadgets according to the discrete partition if $t < \ell$; $\rho_{Q,t}$ if $t = \ell$; and $\rho_{\mathcal{B}_t,t}$ if $t > \ell$. For notational convenience we let $\pi_{\emptyset,\ell} := \pi_{\mathcal{B}_{\ell+1},\ell+1}$ for $\ell < k-1$ and $\pi_{\emptyset,k-1} := \omega(\mathsf{G})$.

Since the partitions $\pi_{Q,\ell}$ are used by Player 2 in the refinement game, we want that refining operations on them are expensive. This is ensured by the next proposition.

Proposition 8.11. *Let (R,S) be an effective elementary refining operation on $\pi_{Q,\ell}$ with $Q \neq \emptyset$. Then for some $q \in Q$, $R = \mathcal{X}_{2q}^{\ell+1}$ or $R = \mathcal{X}_{2q+1}^{\ell+1}$, and $S = \mathcal{Y}_q^\ell$. The cost of this operation is $k^2 2^{k-(\ell+1)}$.*

Proof. The partition $\pi_{Q,\ell}$ contains either $\rho_{Q,t}$, $\rho_{\mathcal{B}_t,t}$ or the discrete partition on every AND_t gadget. Hence, by Lemma 8.8, it is stable on every such gadget. Moreover, by the connection of the in-terminals to Y and the out-terminals to X it follows that $\pi_{Q,\ell}$ is a stable partition of $\mathsf{G}' = (V(\mathsf{G}), E(\mathsf{G}) \setminus \mathcal{X} \times \mathcal{Y})$. Therefore, every effective elementary refining operation involves a cell from $\mathcal{X}$ and a cell from $\mathcal{Y}$. By the structure of the partition, every such refinement operation has the desired form. □

The proposition limits the choices of Player 1 if the current partition is $\pi_{Q,\ell}$. In this situation he is forces to choose some $q \in Q$ and apply either $(\mathcal{X}_{2q}^{\ell+1}, \mathcal{Y}_q^\ell)$ or $(\mathcal{X}_{2q+1}^{\ell+1}, \mathcal{Y}_q^\ell)$. Because $\pi_{Q,\ell}(\mathcal{X}_{2q}^{\ell+1}, \mathcal{Y}_q^\ell) = \pi_{Q,\ell}(\mathcal{X}_{2q+1}^{\ell+1}, \mathcal{Y}_q^\ell)$ we can denote the resulting partition by $r_q(\pi_{Q,\ell})$. The next proposition shows that afterwards Player 2 can choose $\pi_{Q\setminus\{q\},\ell}$ to be the next partition.

Proposition 8.12. *For every $\ell \in \{0, \dots, k-1\}$, nonempty $Q \subseteq \mathcal{B}_\ell$ and $q \in Q$: $r_q(\pi_{Q,\ell}) \preceq \pi_{Q\setminus\{q\},\ell} \preceq \omega(\mathsf{G})$.*

Proof. By Lemma 8.9 the coarsest stable partition of G is nearly discrete and thus $\pi_{Q,\ell} \preceq \omega(\mathsf{G})$ by definition of $\pi_{Q,\ell}$. Furthermore, it holds that $\pi_{Q,\ell} \preceq \pi_{Q',\ell}$ if $Q \supseteq Q'$. On X, $\mathcal{X}$, $\mathcal{Y}$ and Y this can be verified by Definition 8.10 and on the AND gadgets this follows from Lemma 8.8. To finish the proof note that $\pi_{Q,\ell} \setminus \{\mathcal{Y}_q^\ell\} = r_q(\pi_{Q,\ell}) \setminus \{\mathcal{Y}_{2q}^\ell, \mathcal{Y}_{2q+1}^\ell\}$ and $\mathcal{Y}_{2q}^\ell, \mathcal{Y}_{2q+1}^\ell \in \pi_{Q\setminus\{q\},\ell}$. Therefore, $r_q(\pi_{Q,\ell}) \preceq \pi_{Q\setminus\{q\},\ell}$. □

Lemma 8.13. *Player 2 has a strategy to obtain a score of $k^3 2^{k-1}$ in the refinement game on G_k.*

Proof. The configurations of the game are the partitions $\pi_{Q,\ell}$ for nonempty Q and $0 \leq \ell \leq k-1$. The game starts with $\pi_{\mathcal{B}_0,0}$, i. e. Player 2 chooses $\pi_{\mathcal{B}_0,0}$ after the first refinement $(V(\mathsf{G}), V(\mathsf{G}))$. In each round Player 1 has to choose some $q \in Q$, apply the refining operation according to Proposition 8.11 and reach $r_q(\pi_{Q,\ell})$. Player 2 answers by choosing $\pi_{Q\setminus\{q\},\ell}$ (recall that $\pi_{\emptyset,\ell} := \pi_{\mathcal{B}_{\ell+1},\ell+1}$). This choice is valid because of Proposition 8.12. The game ends with the coarsest stable partition $\pi_{\emptyset,k-1} = \omega(\mathsf{G})$. By Proposition 8.11 we can determine the refinement cost as follows. On every level ℓ the game starts with $\pi_{\mathcal{B}_\ell,\ell}$, performs $|\mathcal{B}_\ell| = 2^\ell$ refinement operations and ends up with $\pi_{\emptyset,\ell}$. Since every refinement operation costs $k^2 2^{k-(\ell+1)}$ (Proposition 8.11), the overall costs for this level are $k^2 2^{k-1}$. As there are k levels in the game, the total score is $k^3 2^{k-1}$. □

Proof of Theorem 6. By construction, the graph G_k has $n = O(2^k k)$ vertices and $m = O(2^k k^2)$ edges. Because of Lemma 8.13 and Corollary 8.7 we have $\mathrm{cost}(\mathsf{G}_k) \geq k^3 2^{k-1} = \Omega(m \log n)$. □

Bibliography

[AIK84] Akeo Adachi, Shigeki Iwata, and Takumi Kasai. „Some combinatorial game problems require $\Omega(n^k)$ time“. In: *J. ACM* 31 (2 1984). DOI: 10.1145/62.322433.

[ABD07] Albert Atserias, Andrei A. Bulatov, and Víctor Dalmau. „On the Power of k -Consistency“. In: *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*. 2007, pp. 279–290. DOI: 10.1007/978-3-540-73420-8_26.

[AD08] Albert Atserias and Víctor Dalmau. „A combinatorial characterization of resolution width“. In: *J. Comput. Syst. Sci.* 74.3 (May 2008), pp. 323–334. DOI: 10.1016/j.jcss.2007.06.025.

[AFT11] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. „Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution“. In: *J. Artif. Intell. Res.* 40 (2011), pp. 353–373. DOI: 10.1613/jair.3152.

[AKV04] Albert Atserias, Phokion G. Kolaitis, and Moshe Y. Vardi. „Constraint Propagation as a Proof System“. In: *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*. 2004, pp. 77–91. DOI: 10.1007/978-3-540-30201-8_9.

[ALN14] Albert Atserias, Massimo Lauria, and Jakob Nordström. „Narrow Proofs May Be Maximally Long“. In: *Proceedings of the 29th IEEE Conference on Computational Complexity (CCC'14)*. 2014, pp. 286–297. DOI: 10.1109/CCC.2014.36.

[Bab81] L. Babai. „Moderately exponential bound for graph isomorphism“. In: *Proceedings of Fundamentals of Computation Theory (FCT'81)*. Springer, 1981, pp. 34–50. DOI: 10.1007/3-540-10854-8_4.

[BL83] László Babai and Eugene M. Luks. „Canonical Labeling of Graphs“. In: *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC '83)*. 1983, pp. 171–183. DOI: 10.1145/800061.808746.

[BIS07] Paul Beame, Russell Impagliazzo, and Ashish Sabharwal. „The Resolution Complexity of Independent Sets and Vertex Covers in Random Graphs“. In: *computational complexity* 16.3 (2007), pp. 245–297. DOI: 10.1007/s00037-007-0230-0.

[BW01] Eli Ben-Sasson and Avi Wigderson. „Short proofs are narrow - resolution made simple“. In: *J. ACM* 48.2 (2001), pp. 149–169. DOI: 10.1145/375827.375835.

[Ber12a] Christoph Berkholz. „Lower Bounds for Existential Pebble Games and k-Consistency Tests“. In: *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS'12)*. 2012, pp. 25–34. DOI: 10.1109/LICS.2012.14.

[Ber12b] Christoph Berkholz. „On the Complexity of Finding Narrow Proofs“. In: *Proceedings of the 53th IEEE Symposium on Foundations of Computer Science (FOCS'12)*. 2012, pp. 351–360. DOI: 10.1109/FOCS.2012.48.

[Ber13] Christoph Berkholz. „Lower Bounds for Existential Pebble Games and k-Consistency Tests“. In: *Logical Methods in Computer Science* 9.4 (2013). DOI: 10.2168/LMCS-9(4:2)2013.

[Ber14] Christoph Berkholz. „The Propagation Depth of Local Consistency“. In: *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP'14)*. 2014, pp. 158–173. DOI: 10.1007/978-3-319-10428-7_14.

[BBG13] Christoph Berkholz, Paul Bonsma, and Martin Grohe. „Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement“. In: *Proceedings of the 21st European Symposium on Algorithms (ESA'13)*. 2013, pp. 145–156. DOI: 10.1007/978-3-642-40450-4_13.

[BV13] Christoph Berkholz and Oleg Verbitsky. „On the Speed of Constraint Propagation and the Time Complexity of Arc Consistency Testing“. In: *Proceedings of the 38th International Symposium on Mathematical Foundations of Computer Science (MFCS'13)*. 2013, pp. 159–170. DOI: 10.1007/978-3-642-40313-2_16.

[Bes+05] Christian Bessière et al. „An optimal coarse-grained arc consistency algorithm“. In: *Artificial Intelligence* 165.2 (2005), pp. 165–185. DOI: doi:10.1016/j.artint.2005.02.004.

[CFI92] J. Cai, M. Fürer, and N. Immerman. „An optimal lower bound on the number of variables for graph identifications“. In: *Combinatorica* 12.4 (1992), pp. 389–410. DOI: 10.1007/BF01305232.

[CC82] A. Cardon and M. Crochemore. „Partitioning a graph in $O(|A| \log_2 |V|)$“. In: *Theoretical Computer Science* 19.1 (1982), pp. 85–98. DOI: 10.1016/0304-3975(82)90016-0.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. „Alternation“. In: *J. ACM* 28.1 (Jan. 1981), pp. 114–133. DOI: 10.1145/322234.322243.

[Che+06] Jianer Chen et al. „Strong computational lower bounds via parameterized complexity“. In: *Journal of Computer and System Sciences* 72.8 (2006), pp. 1346–1367. DOI: 10.1016/j.jcss.2006.04.007.

[Coo89] Martin C. Cooper. „An optimal k-consistency algorithm“. In: *Artificial Intelligence* 41.1 (1989), pp. 89–95. DOI: 10.1016/0004-3702(89)90080-5.

[CG70] D. G. Corneil and C. C. Gotlieb. „An Efficient Algorithm for Graph Isomorphism“. In: *J. ACM* 17.1 (1970), pp. 51–64. DOI: 10.1145/321556.321562.

[DKV02] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. „Constraint Satisfaction, Bounded Treewidth, and Finite-Variable Logics“. In: *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*. 2002, pp. 310–326. DOI: 10.1007/3-540-46135-3_21.

[DP85] Rina Dechter and Judea Pearl. *A problem simplification approach that generates heuristics for constraint-satisfaction problems*. Tech. rep. Cognitive Systems Laboratory, Computer Science Department, University of California, Los Angeles, 1985.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[FV98] Tomás Feder and Moshe Y. Vardi. „The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory“. In: *SIAM Journal on Computing* 28.1 (1998), pp. 57–104. DOI: 10.1137/S0097539794266766.

[Fre78] Eugene C. Freuder. „Synthesizing constraint expressions“. In: *Commun. ACM* 21 (11 Nov. 1978), pp. 958–966. DOI: 10.1145/359642.359654.

[Für01] Martin Fürer. „Weisfeiler-Lehman Refinement Requires at Least a Linear Number of Iterations“. In: *ICALP*. 2001, pp. 322–333. DOI: 10.1007/3-540-48224-5_27.

[Gal77] Zvi Galil. „On Resolution with Clauses of Bounded Size“. In: *SIAM Journal on Computing* 6.3 (1977), pp. 444–459. DOI: 10.1137/0206031.

[GS11] Serge Gaspers and Stefan Szeider. „The parameterized complexity of local consistency“. In: *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*. 2011, pp. 302–316. DOI: 10.1007/978-3-642-23786-7_24.

[Gro99] Martin Grohe. „Equivalence in Finite-Variable Logics is Complete for Polynomial Time“. In: *Combinatorica* 19.4 (1999), pp. 507–532. DOI: 10.1007/s004939970004.

[Gro12] Martin Grohe. „Fixed-point Definability and Polynomial Time on Graphs with Excluded Minors“. In: *J. ACM* 59.5 (2012), 27:1–27:64. DOI: 10.1145/2371656.2371662.

[Hak85] Armin Haken. „The intractability of resolution“. In: *Theoretical Computer Science* 39 (1985), pp. 297–308. DOI: 10.1016/0304-3975(85)90144-6.

[HU06] Alex Hertel and Alasdair Urquhart. *The Resolution Width Problem is EXPTIME-Complete*. Tech. rep. 133. 2006. URL: http://eccc.hpi-web.de/report/2006/133/.

[HU09] Alex Hertel and Alasdair Urquhart. *Comments on ECCC Report TR06-133: The Resolution Width Problem is EXPTIME-Complete*. Tech. rep. 003. 2009. URL: http://eccc.hpi-web.de/report/2009/003/.

[Her08] Alexander Hertel. „Applications of Games to Propositional Proof Complexity“. PhD thesis. University of Toronto, 2008. URL: http://eccc.hpi-web.de/static/books/Applications_of_Games_to_Propositional_Proof_Complexity/.

[Hop71] J.E. Hopcroft. „An n log n algorithm for minimizing states in a finite automaton“. In: *Theory of Machines and Computations*. Academic Press, 1971, pp. 189–196.

[IL90] Neil Immerman and Eric Lander. „Describing Graphs: a First-Order Approach to Graph Canonization“. In: *Complexity theory retroperspective* (1990). Ed. by A. Selman, pp. 59–81. DOI: 10.1007/978-1-4612-4478-3_5.

[KAI79] Takumi Kasai, Akeo Adachi, and Shigeki Iwata. „Classes of Pebble Games and Complete Problems“. In: *SIAM J. Comput.* 8.4 (1979), pp. 574–586. DOI: 10.1137/0208046.

[Kas90] Simon Kasif. „On the parallel complexity of discrete relaxation in constraint satisfaction networks“. In: *Artificial Intelligence* 45.3 (1990), pp. 275–286. DOI: 10.1016/0004-3702(90)90009-0.

[KP03] Phokion G. Kolaitis and Jonathan Panttaja. „On the Complexity of Existential Pebble Games“. In: *Proceedings of the 17th International Workshop CSL 2003, 12th Annual Conference of the EACSL (CSL'03)*. 2003, pp. 314–329. DOI: 10.1007/978-3-540-45220-1_26.

[KV95] Phokion G. Kolaitis and Moshe Y. Vardi. „On the Expressive Power of Datalog: Tools and a Case Study“. In: *J. Comput. Syst. Sci.* 51.1 (1995), pp. 110–134. DOI: 10.1006/jcss.1995.1055.

[KV00] Phokion G. Kolaitis and Moshe Y. Vardi. „A Game-Theoretic Approach to Constraint Satisfaction“. In: *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00)*. 2000, pp. 175–181.

[LM94] Peter B. Ladkin and Roger D. Maddux. „On Binary Constraint Problems“. In: *J. ACM* 41.3 (1994), pp. 435–469. DOI: 10.1145/176584.176585.

[Mac77] Alan K. Mackworth. „Consistency in networks of relations“. In: *Artificial Intelligence* 8.1 (1977), pp. 99–118. DOI: 10.1016/0004-3702(77)90007-8.

[Pud00] Pavel Pudlak. „Proofs as Games“. In: *The American Mathematical Monthly* 107.6 (2000), pp. 541–550.

[SH87] Ashok Samal and Tom Henderson. „Parallel consistent labeling algorithms“. In: *International Journal of Parallel Programming* 16 (5 1987), pp. 341–364. DOI: 10.1007/BF01407901.

[Sch88] Uwe Schöning. „Graph isomorphism is in the low hierarchy“. In: *Journal of Computer and System Sciences* 37.3 (1988), pp. 312–323. DOI: 10.1016/0022-0000(88)90010-4.

[Sus+91] Steven Y. Susswein et al. „Parallel path consistency“. In: *International Journal of Parallel Programming* 20.6 (1991), pp. 453–473. ISSN: 0885-7458. DOI: 10.1007/BF01547895.

[Urq12] Alasdair Urquhart. „Width and size of regular resolution proofs“. In: *Logical Methods in Computer Science* 8.2 (2012). DOI: 10.2168/LMCS-8(2:8)2012.

[Wil05] Ryan Williams. „A New Algorithm for Optimal 2-constraint Satisfaction and Its Implications“. In: *Theor. Comput. Sci.* 348.2 (2005), pp. 357–365. DOI: 10.1016/j.tcs.2005.09.023.