University of Bremen

Bachelor of Science

Thesis

Minimal Depth Sorting Networks

Author:
Tobias Haslop

Supervisors:
Prof. Dr. Thomas Schneider
Prof. Dr. Rüdiger Ehlers

November 20, 2020

Acknowledgements

I am deeply grateful to my supervisor Thomas Schneider for taking on an unfamiliar topic and investing much time into our meetings to support me.

I would also like to thank my prior supervisor Rüdiger Ehlers for starting me off on finding optimal sorting networks in our bachelor's project TAPPS (TAckling Practical Problems with Solvers). His early guidance helped me to identify a feasible topic for my thesis. Further I would like to extend my gratitude to the other project members, who contributed to the joyful atmosphere. That definitely helped to keep my interest in sorting networks even beyond the project.

Abstract

Sorting networks are oblivious comparison based sorting algorithms for a fixed number of elements n. They have been a topic of research for 65 years. In this thesis we consider the problem of depth optimality, which asks for the minimum number of parallelized steps in a sorting network. In recent years Bundala and Závodný achieved a leap by settling optimality for $n \leq 16$, which was previously only known up to $n \leq 10$. They reduced the problem to Boolean SAT where they fixed the start of the network for different SAT instances. For that they showed that it suffices to consider a sufficiently small number of prefixes of a network. I propose to add constraints to the Boolean SAT formula to break symmetries between whole networks instead of breaking symmetries between prefixes of networks outside of the SAT problem. For this I provide an analysis of a novel symmetry that twists inputs of comparators. Specifically I break the symmetries between networks whose underlying graphs are isomorphic as defined by Choi and Moon. In order to show this, I rectify isomorphism results by Choi and Moon.

Contents

1 Introduction

Sorting is a fundamental task in computer science. It enforces an additional invariant on sets that allows for more efficient algorithms, i.e. binary search in logarithmic time [1, p. 409-416].

Sorting may also be used to improve caching or branch locality. In ray tracing in computer graphics for example we sort rays into groups of coherent rays [2]. Coherent rays are reflected into the same direction. Ray coherence improves the locality of their subsequent computation paths as well as the spatial locality of their memory accesses on those paths.

Notably succeeding algorithms are not the only use case where sorted data improves efficiency. Humans profit similarly from data being sorted, i.e. a sorted phone book is more usable. In this usability sense sorting is not the means to an end but rather the end itself.

There are many sorting algorithms, such as Bubblesort and Quicksort [1, p. 106-110, 113-122]. Most sorting algorithms, including these two, assume only a comparison operator $\leq$ that is used to sort. There are other sorting algorithms that make different assumptions, e.g. they additionally restrict the domain of the values that are sorted. Such sorts are not considered comparison sorts.

1.1 Sorting Networks

This thesis considers a special type of comparison sort that is called a sorting network. Sorting networks only make the additional assumption that a sorting network sorts a fixed number of elements n. Sorting algorithms, including comparison sorts, typically sort any number of elements.

Sorting networks represent all comparison sorts that are *data-oblivious* [1, p. 219-221]. Data-oblivious algorithms do not branch based on the input data. In other words they perform compare and swap operations on the same positions independent of previous comparisons. For example Quicksort is not data-oblivious because it recursively compares and swaps within a subsequence and the partitioning into subsequences depends

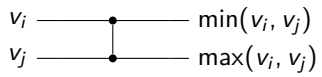

Figure 1.1: *Two channels with channel numbers i and j carrying the values v_i and v_j are sorted by a standard gate.*

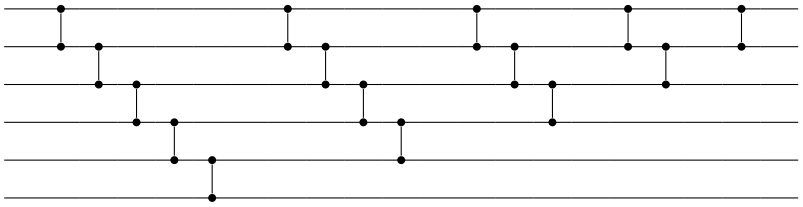

Figure 1.2: *Bubblesort represented as a sorting network for $n = 6$. Usually each iteration bubbles the largest remaining value up to the top position. However Knuth diagrams sort the maximum value to the lower channel. Hence the maximum bubbles down to the bottom instead.*

on the data, in particular the pivot element.

Sorting networks are commonly depicted as Knuth diagrams [1, p. 221]. Figure 1.2 shows the Knuth diagram of the sorting network representing bubblesort. The input values are propagated on channels (horizontal lines) from left to right through the network. Comparator gates (vertical lines) compare and swap the values on two channels such that the minimum value is always output on the upper channel and the maximum value on the lower channel (see Figure 1.1). The connection is made explicit by black filled circles depicting the gate's pins. Sorting networks are data-oblivious because comparators always compare the same channels independent of previous comparisons.

Sorting networks work *in-place*. In-place algorithms get by with the space allocated for the input and neglectible constant overhead, e.g. an auxiliary variable to perform a swap operations. Each channel represents one memory location such that at any stage exactly n memory locations are used.

Since sorting networks are data-oblivious, the number of comparisons in the best and worst case coincides with the average case. This leads to predictable runtime behavior and prevents data leaks via runtime information. Due to their oblivious nature sorting networks can be reified as a hardware circuit or alternatively in software program code.

1.2 Applications of Sorting Networks

There are various applications of sorting networks. They are suited to act as switching networks for example [3].

Further sorting networks outperform other sorting algorithms in certain situations. E.g. the standard sort in libraries is often Quicksort with insertion sort as the base case. In this case sorting networks beat insertion sort for small, fixed n [4]. This is because correct implementations of sorting networks can avoid negative interactions with branch prediction, which is employed in every modern processor to improve pipelining of conditional instructions.

Sorting networks are actually used to encode at-most-k constraints in Boolean SAT because they can even be simulated in propositional logic [5]. Such constraints ensure that at most k from a set of variables are set in a satisfying assigment.

Research on sorting networks dates back to 1954. Later sorting networks became an important research topic in parallel complexity theory [6]. Being oblivious sorting networks are a good fit for parallelized sorting on the GPU because the runtime on modern GPU architectures is particularly susceptible to branching issues. The reason aside from branch prediction possibly failing is that modern GPUs run groups of threads in lockstep. Threads in lockstep are synchronized and wait for each other if they enter different branches. An example for such a parallel setting are rank order filters in image processing such as the median filter [7]. In this case we have a network for each pixel that sorts the intensity values of the surrounding pixels.

1.3 Optimal Size and Depth of a Sorting Network

From the very beginning researchers were looking for networks with a minimal number of comparators [1, p. 225-228]. The number of comparators is called the *size* of a network.

Notably consecutive comparators that work on different channels can be applied in parallel. For an example see Figure 1.3. The number of parallelized execution steps is called the *depth* of a network. A single parallelized step is called a *layer*.

There are two optimization problems related to the size or the depth of a sorting network respectively. Both problems assume a fixed number of channels n.

1. What is the minimal size of a sorting network with n channels? Or equivalently what is the necessary number of data-oblivious comparisons to sort n elements?

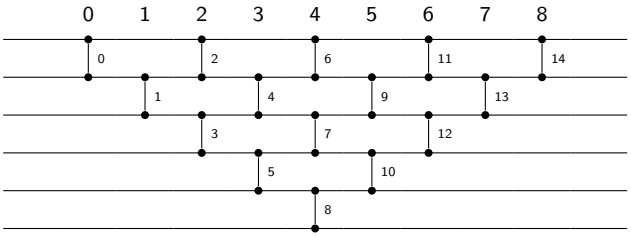

Figure 1.3: *A parallelized version of the bubblesort network from Figure 1.2 with depth 9, that is with 9 layers (parallelized steps), and size 15.*

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_n	1	3	5	9	12	16	19	25	29	35 33	39 37	45 41	51 45	56 49	60 53	71 58	78 63	86 68	92 73
d_n	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	10	11 10	11 10	11 10

Table 1.1: *Table of the minimum size (s_n) and minimum depth (d_n) of sortings networks with n channels. For unknown values the cell contains the known lower and upper bound instead. Notably there are networks that are both size and depth optimal for $n \leq 9$, but this is not the case for $n = 10$ and $n = 12$ [9].*

2. What is the minimal depth of a sorting network with n channels? Or equivalently what is the necessary number of parallelized steps to sort n elements where each step is made up of any number of independent data-oblivious comparisons?

In the theory of parallel algorithms size and depth are fundamental measures of an algorithm [8]. In this context the size is referred to as the work of the algorithm. The energy consumption of an algorithm depends on its work. This thesis focuses on Problem 2 (depth). However both problems are closely related.

Previous work by several authors over several decades succeeded in settling depth optimality for only $n \leq 17$. Table 1.1 shows the optimal size and depth for a small number of elements, i.e. $n \leq 20$. We give a detailed overview of the results in this table in Section 4.2. While mainly of theoretical interest, such results have a small practical impact too. There is work that uses sorting networks explicitely for small n [4, 7]. Although great improvements for a single sorting network with few channels are not possible, we commonly execute many networks such that the improvement scales to a higher order of magnitude. While runtime improvements may or may not be noticeable in a massively parallel setting, the savings in energy increase indefinitely over time.

1.4 Symmetries in Computer Assisted Optimality Proofs

The results for $9 \leq n \leq 17$ were obtained via computer assisted proofs [10, 11, 12]. The state of the art approach reduces the related decision problem of each optimatization problem to Boolean SAT. The reduced problem is then solved by a SAT solver [11, 12].

All previous computer assisted proofs for optimal networks extensively exploited symmetries between pairs of networks where one network sorts iff the other network sorts. Instead of checking whether each network in a symmetry class sorts, it suffices to consider only a single representative network from that class. The state of the art approach considers various symmetries on the first two layers of a network and solves SAT instances with the first two layers fixed for each representative prefix network.

In this thesis I propose to consider these symmetries on all layers. Specifically I propose that the SAT solver exploits them in a single run. This is fundamentally different from the state of the art where the symmetries on prefixes are exploited in a prior step such that the eventual number of solver runs corresponds to the number of symmetry classes.

This thesis revolves around a certain kind of symmetry by Choi and Moon (CM) where pairs of symmetric networks have isomorphic underlying graphs [13]. CM state that their graph isomorphy enables a channel permutation followed by a reconnection of comparators that transforms one network into the other. In this thesis we formalize how permutation and untangling corresponds to an isomorphism between underlying graphs. It turns out that yet a third transformation is necessary to be able to represent any CM-isomorphism of the underlying graphs as a composition of such network transformations.

I believe the approach can be extended to include more from the previously considered symmetries on two-layer prefixes, i.e. saturation and perhaps even reflectional symmetries, but this is outside the scope of this thesis. However I am dubious whether saturation can be handled completely beyond the second layer. Further I believe it is possible to break suffixes instead of prefixes in a prior step outside the SAT problem.

This thesis is structured as follows. Section 2 introduces necessary basic knowledge. This includes SAT solvers, the concept of symmetries, how symmetries are composed and how they are exploited in SAT solving. For additional context Section 3 gives a brief overview on the theory of sorting networks. Section 4 considers the decision problems related to the size and depth optimality problems and presents previous work and results for small n. In particular we take a closer look at the state of the art approach. In Section 5 we define the elementary transformations of the CM-symmetry and show that permuting and untangling is better captured by input twists that are dual to untangling. We exploit the input twists by applying standard symmetry breaking techniques from the SAT solving domain in Section 6. Section 7 formalizes the correspondence between the graph isomorphism and network transformations.

2 Preliminaries

2.1 Formalization of Sorting Networks

In this section we give additional fundamental definitions related to sorting networks. Unlike in the introduction in Section 1.1 we also highlight the differences between generalized and standard comparator networks, sorting and π-sorting networks as well as ordered and unordered layers.

In the rest of this thesis we use $[x]$ to denote the set of all natural numbers from 0 to $x-1$. In particular we number the channels from 0 such that $[n]$ gives us the set of channel numbers.

We define a *comparator network* as a sequence of comparators. A *comparator* (i, j) in $[n]^2$ with $i \neq j$ operates on the ith and the jth channel. The n channel values at some stage in the network are represented as a vector with n components (natural numbers). Application of a comparator (i, j) to a vector $\overline{x}$ changes x_i to $\min(x_i, x_j)$ and x_j to $\max(x_i, x_j)$. Then a comparator network is applied by sequentially applying the comparators. Note that a comparator network yields an algorithm that modifies a vector in-place. An example comparator network for four values is $N = (0,1),(2,3),(0,2),(1,3),(0,1),(2,3)$.

A comparator $c = (i, j)$ is a *min-max* comparator if $i < j$, otherwise if $j > i$ it is a *max-min* comparator. A comparator network without any max-min comparators is called a *standard* comparator network. *Generalized* comparator networks may contain max-min comparators but not necessarily, whereas any standard network is necessarily a generalized network.

We denote the set of all generalized networks with $\mathcal{N}$. Similarly we denote the set of generalized networks with n channels and size s as $\mathcal{N}_n^s$. Like the channels we number comparators from 0 such that $[s]$ gives us all comparator numbers of a network in $\mathcal{N}_n^s$ respectively.

A comparator network N is called *sorting* if the output $N(\overline{x})$ is sorted for every input vector, i.e. for all $\overline{x}$ in $\mathbb{N}^n$, we have $y_i \leq y_{i+1}$ where $\overline{y} = N(\overline{x})$.

There is a weaker condition by that a network is called π-*sorting* if there exists a per-

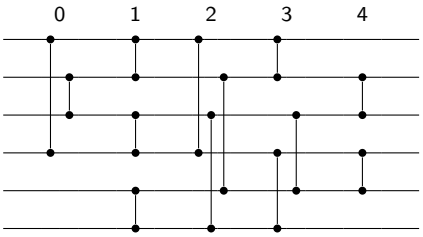

Figure 2.1: *A sorting network with six channels that admits different partitions into layers.*

mutation π on the channel numbers such that the output $N(\overline{x})$ is sorted for every input vector after applying π, i.e. for all $\overline{x}$ in $\mathbb{N}^n$, we have $z_i \leq z_{i+1}$ where $z_i = y_{\pi(i)}$ and $\overline{y} = N(\overline{x})$. Later we use an equivalent less explicit notation to apply π to a vector of values, e.g. this case is less explicitely written as $\overline{z} = \pi(N(\overline{x}))$.

It is well known that any generalized π-sorting network can be transformed into a standard sorting network by a standardization procedure called untangling that may reconnect the comparators to different channels [1, p. 238, 667-668][6]. In Section 5.1 we look at untangling in more detail.

A subsequence of immediately consecutive comparators in a network is called a *layer* if all comparators in that subsequence work on different channels. Thus comparators in a layer are independent of each other. They may be applied in parallel and their order within the layer is irrelevant. Therefore we distinguish between an *ordered* layer, that is a sequence of comparators, and an *unordered* layer, that is a set of comparators.

Sometimes it is advantageous to consider a network as a sequence of layers instead of a sequence of comparators. A network of ordered layers corresponds to exactly one sequence of comparators and exactly one sequence of unordered layers. Note that there may be different partitions of the same comparator sequence into layers. For example Figure 2.1 shows a sorting network with six channels where the first comparator on the last two channels may be partitioned into either the first or the second layer.

Each network N has several prefix and suffix networks. We call a network N' a prefix of N and a network N'' a suffix of N if there is a decomposition of N into N' and N'', i.e. $N = N' ; N''$. In particular we allow a network to have no comparators such that every network is a prefix and a suffix of itself.

2.2 Boolean SAT Solving

In this thesis we reduce the problem of finding a sorting network with certain properties to the Boolean SAT problem. The Boolean SAT problem asks whether one can assign

Boolean values to all variables in a propositional logic formula such that it is satisfied. The name SAT abbreviates SATISFIABILITY. By Cook's Theorem the Boolean SAT problem is NP complete.

An instance of Boolean SAT is a propositional logic formula in conjunctive normal form (CNF). A CNF-formula is of the form $\bigwedge_i \bigvee_j l_{i,j}$ where $\bigvee_j l_{i,j}$ are *clauses* and $l_{i,j}$ are *literals*. A literal is either *positive* or *negative*. A positive literal is a variable v. A negative literal is a negated variable $\neg v$.

An assignment is a set of variable value pairs that assigns a variable exactly one truth value. In a complete assignment every variable in the formula is assigned a truth value. Unless stated otherwise the term assignment refers to a complete assignment. A truth value is a Boolean value that is either true or false.

A positive literal is satified if its variable is assigned true and likewise a negative literal is satisfied if its variable is assigned false. A clause is satisfied if at least one of its literals is satisfied and a formula is satisfied if all of its clauses are satisfied. The Boolean SAT decision problem asks whether there is some complete assignment under that the given CNF-formula is satisfied.

We can represent a clause as a set of literals and a CNF-formula as a set of clauses such that the order of literals within a clause and the order of clauses within a formula are irrelevant.

The problem of satisfiability is more general than Boolean SAT. Other variants include constraint satisfaction problems or satisfiability modulo theories. Unless otherwise stated, we mean Boolean SAT whenever we say SAT in this thesis.

SAT solvers are computer programs that specialize in solving instances of the Boolean SAT problem efficiently. Although SAT is NP complete, SAT solvers solve instances with millions of variables in practice. Progress of SAT solvers is tracked in annual SAT competitions [14].

Most competing modern SAT solvers build on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [15]. At its core the DPLL algorithm is a backtracking algorithm that repeatedly guesses the truth assigment for a single variable. Assigning variable v to be true removes all occurrences of v as a positive literal as well as all clauses containing the literal $\neg v$. If this yields an empty set of clauses, we have satisfied the formula. If this yields a conflict, that is an unsatisfiable empty clause, we backtrack to assign the other truth value to a previously assigned variable. If we cannot backtrack, the formula is unsatisfiable.

DPLL improves the simple backtracking algorithm by taking unit clauses into account. Such clauses contain exactly one literal and can only be satisfied by satisfying that literal.

The DPLL algorithm defers guessing in the presence of unit clauses. Instead it assigns the truth value for the corresponding variable such that the literal of the unit clause is satisfied. This is called *Unit Propagation*.

Another improvement of DPLL is *Conflict Driven Clause Learning* (CDCL) [16]. The idea is that the cause of a conflict can be a much earlier guess of a variable's truth value. In this case we have to backtrack a lot of times before we invert the early erroneous guess and thus solve the conflict. If a CDCL solver encounters a conflict, it identifies variables involved in the conflict. Then it learns a new clause that prevents the conflict and rewinds its truth value guess decisions accordingly. Rewinding is called non-chronological backtracking.

Let us consider an example taken from an acessible talk on SAT solvers [17]. The problem has the clauses.

1. $x_1 \vee x_2$
2. $x_1 \vee x_3 \vee x_8$
3. $\neg x_2 \vee \neg x_3 \vee x_4$
4. $\neg x_4 \vee x_5 \vee x_7$
5. $\neg x_4 \vee x_6 \vee x_8$
6. $\neg x_5 \vee \neg x_6$
7. $x_7 \vee \neg x_8$
8. $x_7 \vee \neg x_9 \vee x_{10}$

A possible run of the CDCL algorithm may first guess x_7 is false. Then unit propagation (UP) in clause 7 infers x_8 to be false. The next guess may be that x_9 is true. Then UP in clause 8 infers x_{10} to be true. Another guess may be that x_1 is false. Then x_1 is true by UP in clause 1, x_3 is true by UP in clause 2, x_4 is true by UP in clause 3, x_5 is true by UP in clause 4. However there is a conflict, because x_6 is false by UP in clause 6 and true by UP in clause 5. Analyzing the conflict a CDCL solver may learn that setting x_4 and x_5 to true and x_8 to false leads to this conflict. Notably this holds independent of any previously guessed truth values of a variable. Alternatively setting x_4 to true and x_7 and x_8 leads to the same conflict because of clause 4. By the contrapositive being conflict free implies that $\neg x_4 \vee x_7 \vee x_8$. Hence we add this clause and rewind to before guessing x_7 (and inferring x_4 and x_5). With the learned clause we infer that x_4 is false at an early stage after guessing x_7 is false. This can prevent costly repeated backtracking in the decision tree of the truth value guesses.

Learned clauses take part in unit propagation. Consequently a lot of additional learned clauses slow down unit propagation. Thus SAT solvers employ different clause management strategies, e.g. heuristics to identify useful learned clauses and discard useless ones.

The next important performance factor is the order, in which variables take part in truth value guessing [18]. For CDCL solvers the Variable State Independent Decaying Sum branching heuristic (VSIDS) has proven to be the most successful heuristic [19]. The point here is that guessing the truth values of the variables in a static order of the variables is in principle suboptimal.

2.3 Symmetries

In this section we consider symmetries, their relation to group theory and equivalence relations as well as their role in speeding up SAT solvers. A more comprehensive introduction to symmetries in satisfiability solving is given in [20].

A *transformation* is a function f that maps a set S to itself, i.e. $f : S \to S$. Recal that $\mathcal{N}_n^s$ is the set of all comparator networks with n channels and size s. Then we call a function $f : \mathcal{N}_n^s \to \mathcal{N}_n^s$ a network transformation.

A *symmetry* is an invertible transformation of a particular set preserving a certain property. Typically a symmetry preserves some structure of the elements. As a consequence the symmetry then also preserves properties that are derived from that structure.

Continuing the first example a sorting network symmetry is an invertible network transformation preserving the property that a network sorts. In other words for all networks N in $\mathcal{N}_n$ we have that N is a sorting network if and only if $f(N)$ is a sorting network. In this thesis we consider sorting network symmetries that preserve an underlying graph structure of a comparator network. Whether a network sorts is determined by that graph structure.

2.3.1 Symmetry Group Representations

Symmetries, group theory and representation theory are closely related. A group G is a set together with a binary operation $\circ$ that satisfies the group laws

$$\forall\ f, g \in G.\ f \circ g \in G \tag{2.1}$$
$$\exists\ \mathsf{id} \in G\ \ \forall\ f \in G.\ f \circ \mathsf{id} = f = \mathsf{id} \circ f \tag{2.2}$$
$$\forall\ f \in G\ \ \exists\ f^{-1} \in G.\ f \circ f^{-1} = \mathsf{id} = f^{-1} \circ f \tag{2.3}$$
$$\forall\ f, g, h \in G.\ (f \circ g) \circ h = f \circ (g \circ h) \tag{2.4}$$

2.1 G is closed under $\circ$.
2.2 There is an identity element id.
2.3 Every element f has an inverse f^{-1} that cancels it.
2.4 The binary operation $\circ$ is associative.

We recall that symmetries are invertible and note that there is an identity transformation. Composing two transformations that preserve the same property yields a transformation that preserves that property too. Composition is associative. Hence symmetries form a group under composition.

The symmetric group $\mathsf{Sym}(S)$ of a set S contains all permutations of S, that is all invertible transformations of elements in S. Note that $\mathsf{Sym}(S)$ unlike symmetry groups preserves no properties.

The concept of a *group action* further involves the elements of the set S on that the transformations of a group work. Let G be a set of transformations that is a group under composition. Then a left action is a binary operator $\cdot : G \times S \to S$. We write $f \cdot x$ to say that f acts on x (from the left). A left group action must satisfy the group action laws

$$\forall\ f, g \in G,\quad x \in S.\ f \cdot (g \cdot x) = (f \circ g) \cdot x$$
$$\forall\ x \in S.\ \mathsf{id} \cdot x = x.$$

An analogous definition of a right action exists, but we limit ourselves to considering only left actions without loss of generality. Note that $\cdot$ is right associative. We may omit parantheses and $\cdot$, i.e. write gfx instead of $g \cdot (f \cdot x)$.

For example if the group G contains transformations of S, then a suitable action is transformation application, that is $f \cdot x \mapsto f(x)$. This is the *natural* group action and we say that G acts naturally on S.

However the group G may instead contain transformations of a set T unrelated to S. Then a suitable definition of a group action enables G to act on S anyway. For ex-

ample in Section 5 we consider an action that enables the group of all channel number permutations to act as network transformations on the set of networks.

Strictly speaking the definition of a group action does not require G to be a set of transformations that is a group under composition. Actually G may be any group, but this generalization is not relevant to this thesis.

The orbit of a point x contains all of its possible destinations points through a transformation in G. The orbit is denoted by $G \cdot x$.

$$G \cdot x = \{g \cdot x \mid g \in G\}$$

Proposition 1

1. A group G acting on a set S with some action $\cdot$ induces an equivalence relation on S where the equivalence class of an element in S is its orbit.

2. An equivalence relation R on a set S induces a group of transformations of S such that the group acts naturally on S where the orbit of an element in S is its equivalence class.

Proof.

1. The proof can be found in standard literature, e.g. [21, p. 202].

2. We define the set G_R such that it contains a transformation f of S if $x \; R \; f(x)$ for all x in S. Next we show that G_R is a group under composition. This implies that it is acting on S because we have an appropriate definition for $\cdot$ at hand.

   The identity of G_R is the identity transformation. An inverse exists because R is a symmetric relation. Composition is associative. The set of transformations G_R is closed under composition because R is a transitive relation.

   We observe that the orbit of an element in S is its equivalence class by definition of G_R.

□

This means we can alternatively define a symmetry as an equivalence relation whose equivalence classes preserve a particular property.

Note that Proposition 1.1 and Proposition 1.2 reverse each other except that 2 induces only the natural group action. Thus every group G acting on a set S with some action $\cdot$ induces another group acting naturally on S where the two orbits of each element in S are the same. We say that the group G is *represented* on S or more explicitely that G is represented as another group that acts naturally on S. Alternatively we say that G acts as another group on S. Although Proposition 1.2 is relevant because it states that every equivalence relation induces a group of transformations, it is actually not the most suitable definition for this concept of group representation by an action as the next example shows.

We consider the sets $A = \{0, 1, 2\}$ and the set $B = \{a_0, a_1, a_2\}$. The following transformations of A shift every element in a cycle and form a group G under composition, that is $G = \{\mathsf{id}, x \mapsto x + 1 \mod 3, x \mapsto x - 1 \mod 3\}$. For G there is a single orbit, that is A. We can define an action $* : G \times B \to B$ as $f * a_i = a_{f(i)}$. Then for G acting on B with the action $*$ there is similarly a single orbit, that is B. Hence the induced equivalence relation is B^2 because all elements in B are in the same equivalence class (orbit). Further G is represented as another group H on B. Note that H contains all of the $3! = 6$ transformation of B, i.e. it additionally contains all the swaps of two elements in B respectively. Next we revise the concept of representation such that the group representation H does not contain the swaps and every transformation in G corresponds to exactly one transformation in H.

There is an alternative equivalent definition of the group action $\cdot : G \times S \to S$ in curried notation with the signature $\mathsf{act} : G \to (S \to S)$ [21]. With the earlier assumption that G contains transformations of a set T this yields the signature $\mathsf{act} : (T \to T) \to (S \to S)$. This alternative definition illustrates that a transformation of T from a group G acting on S induces a transformation of S. We say a transformation of T acts as a transformation of S or that it is represented as a transformation of S. Be aware that act is only total if G contains all transformations of T. If we recall that G is represented on S as another group H acting naturally on S, then we can define the signature more precisely as $\mathsf{act} : G \to H$. With this signature act is not only total but also surjective.

We call a representation *faithful* if act is injective too. Then there is an inverse group action $\mathsf{act}^{-1} : H \to G$. That is if a group G with transformations of T is represented as a group H on S by an injective action $\mathsf{act} : G \to H$, then H likewise is represented as the group G on T by the action $\mathsf{act}^{-1} : H \to G$. For an example we continue the previous remark that in Section 5 the group of all permutations of channel numbers is faithfully represented as a group of network transformations on the set of networks. That group of network transformations is likewise represented as that permutation group on the set of channel numbers. The latter representation is also called a permutation representation because a group is represented as a permutation group.

2.3.2 Symmetry Breaking in SAT

Previous work on optimal sorting networks encodes a comparator network as an instance of the Boolean SAT problem. SAT solving can be sped up by pruning the search space based on symmetries. This is also called symmetry breaking.

SAT Symmetries

In the context of SAT a symmetry preserves satisfiability. It is a permutation π of pairs of variables and truth values. We observe that a literal l corresponds to a pair of a variable x and a truth value v, i.e. we can define a bijection f that converts between them.

$$f(l) = \begin{cases} f(x, \text{true}), & l = x \\ f(x, \text{false}), & l = \neg x \end{cases}$$

$$f^{-1}(x, v) = \begin{cases} x, & v = \text{true} \\ \neg x, & v = \text{false} \end{cases}$$

Hence we can faithfully represent a permutation π of a variable value pair as a permutation of a literal l with the action

$$\pi \cdot l = (f^{-1} \circ \pi \circ f)(l).$$

Specifically a SAT symmetry has to satisfy a condition that is commonly called Boolean consistency. Boolean consistency states that for any variables x and x' and truth values v and v' with $\pi(x, v) = (x', v')$ we have that $\pi(x, \neg v) = (x', \neg v')$, i.e. negation is preserved by π. A symmetry group in SAT either acts on constraints or on solutions. Conversely a symmetry group containing transformations of constraints or solutions respectively is represented as a permutation group of variable-value pairs.

A *constraint transformation* is a function on the set of constraints, i.e. a formula φ. Since a formula is a set of clauses and a clause is a set of literals, we can faithfully represent a permutation of literals π as a constraint transformation with the action that applies π pointwise

$$\pi \cdot \varphi = \{\{\pi(l) \mid l \in c\} \mid c \in \varphi\}.$$

Equivalently we can say

$$\pi \cdot \bigwedge_i \bigvee_j l_{i,j} = \bigwedge_i \bigvee_j \pi(l_{i,j}).$$

A *solution transformation* is a function on the set of assignments satisfying a given formula. We can represent a permutation π of a variable value pair (x, v) as a solution transformation, that is a transformation of an assignment S, with the action

$$\pi \cdot S = \{\pi(x, v) \mid (x, v) \in S\}$$

We consider common notions of symmetry in satisfiability solving [20].

A *constraint symmetry* is a constraint transformation that maps the formula such that its representation as a set of clauses stays the same, i.e. it preserves the problem's clause set representation. Clearly this preserves satisfiability as the set of clauses is the same. For example the permutation that only swaps literal x_1 with $\neg x_3$ (and x_3 with $\neg x_1$) in the formula $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge x_1 \wedge \neg x_3$ preserves the set of clauses

$$\{\{x_1, \neg x_2\}, \{\neg x_2, \neg x_3\}, \{x_1\}, \{\neg x_3\}\} = \{\{\neg x_3, \neg x_2\}, \{\neg x_2, x_1\}, \{\neg x_3\}, \{x_1\}\}.$$

A *solution symmetry* is a solution transformation that maps every satisfying assignment to the same assignment, i.e. it is preserves the solution. For an example we again consider the formula $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge x_1 \wedge \neg x_3$. The set of satisfying assignments is

$$\{\{(x_1, \text{true}), (x_2, \text{false}), (x_3, \text{false})\}, \{(x_1, \text{true}), (x_2, \text{true}), (x_3, \text{false})\}\}.$$

The permutation that only swaps (x_2, true) with (x_2, false) is a solution symmetry because it preserves this set of solutions. We already know that swapping (x_1, true) with (x_3, false) is a constraint symmetry for this formula. It is a solution symmetry too.

A more general SAT symmetry is a transformation that simply preserves satisfiability instead of preserving the problem or the solution.

Notably the same formula has the same set of solutions, while the same set of solutions can be described by different formulas. Hence every constraint symmetry is a solution symmetry but not the other way around. For example the formulas $x_1 \wedge x_2$ and $x_1 \wedge x_2 \wedge (x_1 \vee \neg x_2)$ have the same set of solutions and thus admit the same solution symmetries, but they do not admit the same constraint symmetries. As we see in the example minor changes like adding redundant clauses can prohibit constraint symmetries. For this reason constraint symmetries are also called syntactic symmetries, while solution symmetries are called semantic symmetries.

We may consider restricted notions of a symmetry group in SAT, e.g. a subgroup that acts as a permutation group on the variables only or the truth values only. This yields

the notion of a variable symmetry and a value symmetry respectively. In particular for Boolean SAT value symmetries are sometimes called phase-shift symmetries. For example the permutation that swaps (x_1, false) with (x_1, true) acts only on the truth values. The permutation that swaps (x_1, false) with (x_2, false) as well as (x_1, true) with (x_2, true) acts only on the variables.

Detecting Symmetries

There are many attempts in the literature where authors try to automatically detect symmetries with the goal to break them later. The standard approach to detect symmetries is to represent formulas as a colored graph such that symmetries correspond to automorphisms on this graph and then use a dedicated tool to detect graph automorphisms [22, 23, 24]. This yields constraint symmetries.

In general determining whether a solution transformation is a solution symmetry is coNP-complete, that is at least as hard as determining whether a formula is unsatisfiable [25]. Determining whether a whole group of problem transformations is a group of problem symmetries does not make the problem easier.

Besides domain oblivious automated symmetry detection the human knowledge of the domain may include knowledge of symmetries. As an example the k-coloring problem can be solved with a SAT-solver that utilizes a Theorem about the chromatic number that relates symmetric cases [26].

Symmetry Breaking

Solving a SAT problem with many symmetries leads to a redundant exploration of search space. If we know about pairs of symmetric assignments beforehand, then we may consider only one assignment representative for each equivalence class of symmetric assignments. Since symmetry preserves satisfiability, we know that if the representative does not satisfy the formula, then none of the symmetric assignments satisfies the formula either.

The most common approach to prune the search space based on symmetries is to add additional symmetry breaking constraints in an initial preprocessing step. A symmetry breaking constraint must preserve satisfiability and should ideally reduce the search space to only the representative assignment.

A symmetry breaking constraint φ is *correct* if for every equivalence class of assignments under that symmetry there is at least one assignment that satisfies φ.

A symmetry breaking constraint φ is *complete* if for every equivalence class of assignments under that symmetry there is at most one assignment that satisfies φ.

A straightforward, correct and complete symmetry breaking constraint is the lex-leader constraint [22]. We assume an order of the variables $x_1 < x_2 < \cdots < x_k$ of the input formula with k variables. With this order we can consider an assignment as a binary number. This yields a lexicographic order $<$ of the assignments. Let π be the symmetry that transforms every assignment X into the assignment $\pi \cdot X$. The lex-leader constraint chooses the smallest assignment by some lexicographic ordering as the representative. Specifically it adds constraints that disallow $\pi \cdot X$ for every X with $X \leq \pi(X)$.

$$\bigwedge_{i=1}^{k} \left(\bigwedge_{j=1}^{i} x_j \leftrightarrow \pi(x_j) \right) \rightarrow (\neg x_i \vee \pi(x_i))$$

The constraint ensures that x_i is always smaller than $\pi(x_i)$ if all preceding literals x_j and $\pi(x_j)$ evaluate to the same truth value.

Then for every such pair of symmetric assignments X and $\pi \cdot X$ it is still satisfied by the smaller assignment X. And the lex-leader constraint is correct because for every class of symmetric assignments the lexicographically minimal assignment satisfies the lex-leader constraint. Further the lexicographically minimal assignment is the only satisfying assignment in its class because every symmetric and lexicographically greater assignment is disallowed by the lex-leader constraint. Thus the lex-leader constraint is a correct and complete symmetry breaking constraint for any symmetry π.

Such a lex-leader constraint is added for any symmetry π in the symmetry group and any assignment X where $X < \pi(X)$. It turns out that such a symmetry breaking constraint adds too many large clauses that are rarely utilized. Rather they slow down unit propagation. Efficient approaches only use a small subset of the constraints with small clauses [24, 27]. This does not break the symmetry completely but avoids the great overhead such that a speedup can be achieved. This incomplete breaking of symmetries is also called partial symmetry breaking. Another optimization is to introduce auxiliary variables to achieve a more compact encoding of the lex-leader constraint [28].

Adding symmetry breaking constraints in a preprocessing step is called *static* symmetry breaking. In contrast to that *dynamic* symmetry breaking reduces the clause overhead by adding symmetry breaking constraints during the search when appropriate [26, 29]. This requires modification of the SAT-solver and has to integrate with other SAT-solving techniques such as backjumping. Hence it is difficult to profit from progress in SAT competitions. Fixing the order in which variable assignments are guessed can reduce the size of dynamically added symmetry breaking constraints. However static variable orders are inferior to the VSIDS heuristic.

2.4 Relational Composition

In this thesis we consider various symmetries as equivalence relations between comparator networks. For an equivalence relation R we may write $x \equiv_R y$ instead of $x\ R\ y$ or $(x, y) \in R$. On several occasions we combine different kinds of equivalences to obtain a generalized one. For a straightforward formalization we draw on the concept of a relation algebra [30], which provides us with a composition operator ; for two relations.

Definition 2 (Composition of relations)
The composition of two relations $R \subseteq X \times Y$ and $S \subseteq Y \times Z$ is defined as

$$R\,;S = \{(x, z) \in X \times Z \mid \exists\, y \in Y \text{ such that } x\ R\ y \text{ and } y\ S\ z\}.$$

Proposition 3
The composition of two reflexive relations $R \subseteq X^2$ and $S \subseteq X^2$ is a superset of R and S respectively.

$$R \subseteq R\,;S$$
$$S \subseteq R\,;S$$

Proof. Given either xRz or xSz we show that $x\ (R\,;S)\ z$. In the former case we have xRz and zSz by reflexivity of S such that there is an y in X with xRy and ySz, in particular $y = z$. This proves $x\ (R\,;S)\ z$. The latter case can be argued analogously. □

Proposition 4
Relational composition is idempotent if and only if it works on a transitive relation, i.e. for any relation $R \subseteq X^2$

$$R\,;R = R \iff R \text{ is transitive.}$$

Proof.

$\Longrightarrow$ Let $R\,;R = R$. We prove that R is transitive, i.e. we show that for any x,y and z from X, if xRy and yRz, then xRz. By Definition 2 we have $x\ (R\,;R)\ z$, that is xRz by the premiss $R\,;R = R$.

$\Longleftarrow$ Let R be transitive. We prove that $R\,;R = R$. By Proposition 3 we know that $R;R \supseteq R$. For the other inclusion $R;R \subseteq R$ we show that $x\ (R\,;R)\ z$ implies xRz. By Definition 2 there is an y such that xRy and yRz, that is xRz by transitivity of R.

□

Note that the composition of two equivalence relations may not yield an equivalence relation. Consider the equivalence relations $R = \{(x,x),(y,y),(z,z),(x,y),(y,x)\}$ and $S = \{(x,x),(y,y),(z,z),(y,z),(z,y)\}$. We have that (x,z) is in the composition $R\,;S$, but (z,x) is not. Thus $R\,;S$ is not symmetric and hence cannot be an equivalence relation. Symmetrically we have that (z,x) is in the composition $S\,;R$, but (x,z) is not.

Proposition 5
For all equivalence relations $R, S \subseteq X^2$ the following are equivalent:

1. $R\,;S$ is a symmetric relation.
2. Composition of R and S is commutative, i.e. $R\,;S = S\,;R$.
3. $R\,;S$ is an equivalence relation.

Proof.

$1 \implies 2$ Let $R\,;S$ be a symmetric relation. We show that $R\,;S = S\,;R$. In detail we show that $x\ (R\,;S)\ y$ if and only if $x\ (S\,;R)\ y$ for all x and y in X. First we utilize symmetry of $R\,;S$ and finally symmetry of R and S. Recall that by the premiss $R\,;S$ is an equivalence relation and thus symmetric.

$$\begin{aligned} x\ (R\,;S)\ y &\iff y\ (R\,;S)\ x \\ &\iff \exists\, z \in X \text{ s.t. } yRz \text{ and } zSx \\ &\iff \exists\, z \in X \text{ s.t. } zSx \text{ and } yRz \\ &\iff \exists\, z \in X \text{ s.t. } xSz \text{ and } zRy \\ &\iff x\ (S\,;R)\ y \end{aligned}$$

$2 \implies 3$ Let the composition of R and S be commutative such that $R\,;S = S\,;R$. We prove that $R\,;S$ is an equivalence relation, i.e. that $R\,;S$ is reflexive, symmetric and transitive.

To show reflexivity of $R\,;S$, it must hold that $x\ (R\,;S)\ x$ for all x in X. This holds because we have xSx and xRx by reflexivity of S and R such that there is an element y with xSy and yRx, in particular $y = x$.

To show symmetry of $R\,;S$, it must hold for all x and y in X that $x\ (R\,;S)\ y$ iff $y\ (R\,;S)\ x$. This can be shown by first utilizing symmetry of R and S and finally

the premiss $R\,;S = S\,;R$.

$$\begin{aligned} x\ (R\,;S)\ y &\iff \exists\, z \in X \text{ such that } xRz \text{ and } zSy \\ &\iff \exists\, z \in X \text{ such that } zRx \text{ and } ySz \\ &\iff \exists\, z \in X \text{ such that } ySz \text{ and } zRx \\ &\iff y\ (S\,;R)\ x \\ &\iff y\ (R\,;S)\ x \end{aligned}$$

To show transitivity of $R\,;S$, it must hold for all x, y, z in X that $x\ (R\,;S)\ y$ and $y\ (R\,;S)\ z$ imply $x(R\,;S)z$. This can be shown by first utilizing the premiss $R\,;S = S\,;R$ and finally the transitivity of R and S.

$$\begin{aligned} x\ (R\,;S)\ y \wedge y\ (R\,;S)\ z &\iff \exists\, v, w \in X \text{ s.t. } xRv \text{ and } vSy \text{ and } yRw \text{ and } wSz \\ &\iff x\ (R\,;S\,;R\,;S)\ z \\ &\iff x\ (R\,;(S\,;R)\,;S)\ z \\ &\iff x\ (R\,;(R\,;S)\,;S)\ z \\ &\iff x\ (R\,;R\,;S\,;S)\ z \\ &\iff \exists\, v, w \in X \text{ s.t. } xRv \text{ and } vRy \text{ and } ySw \text{ and } wSz \\ &\iff \exists\, v, w \in X \text{ s.t. } (xRv \text{ and } vRy) \text{ and } (ySw \text{ and } wSz) \\ &\implies xRy \text{ and } ySz \\ &\implies x\ (R\,;S)\ z \end{aligned}$$

$3 \implies 1$ The equivalence relation $R;S$ is symmetric by definition of an equivalence relation.

□

Let two groups G and H each act on the same set S with the actions $\cdot : G \times S \to S$ and $* : H \times S \to S$ respectively. Their induced equivalence relations $R_G \subseteq S^2$ and $R_H \subseteq S^2$ commute under composition, i.e. $R_G\,;R_H = R_H\,;R_G$, iff their representations commute under composition, i.e. for all g in G, all h in H and all x in S we have that $g{\cdot}h{*}x = h{*}g{\cdot}x$. Remember that actions are right associative, i.e. it is $g{\cdot}(h{*}x) = h{*}(g{\cdot}x)$, and the second group law states for all f_1, f_2 in some group that $f_1 \cdot (f_2 \cdot x) = (f_1 \circ f_2) \cdot x$.

3 Sorting Networks

In this section we give a brief overview on the theory of sorting networks. We already introduced sorting networks as data-oblivious in-place sorting algorithms depicted as Knuth diagrams in Section 1.1. To complete this introduction we first consider a certain subclass of sorting networks called primitive networks that induce stability. Afterwards we list various known sorting network construction schemes. This includes practical constructions, but we also identify asymptotic bounds for the size and depth of sorting networks with theoretical constructions.

3.1 Stable Sorting Networks

A *stable* sorting algorithm retains the order of equal elements. Generally sorting networks are not stable sorts. For a counterexample see Figure 3.1. However a sorting network is stable if all of its comparators work on adjacent channels. Knuth calls such networks *primitive* [1, p. 240]. For example the bubblesort network from Figure 1.2 is primitive. In primitive networks the channel number of a value can change at most by 1 in each step. Thus the order (by channel numbers) of two values can only change by a direct comparison. Therefore the order of equal values is retained.

Some authors restrict their notion of sorting networks to primitive sorting networks [31]. However this thesis is neither concerned with stability nor with primitive networks. Thus we allow comparators between any two channels.

The odd-even transposition network (Figure 3.2) is another primitive sorting network with size $\binom{n}{2}$ like the bubblesort network (Figure 1.2) but with fewer layers. Actually $\binom{n}{2}$ is the minimum number of comparators necessary in a primitive sorting network [1, p. 240, 669].

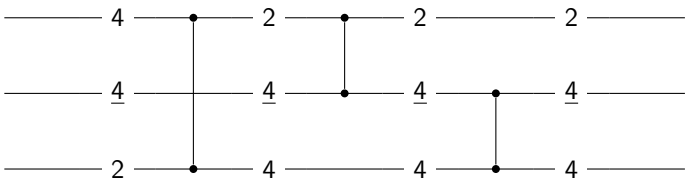

Figure 3.1: *A sorting network that is not stable. In a stable sorting network the order of equal values is retained, i.e. if* $\underline{4}$ *comes before* 4 *in the input sequence, then* $\underline{4}$ *comes before* 4 *in the output sequence too.*

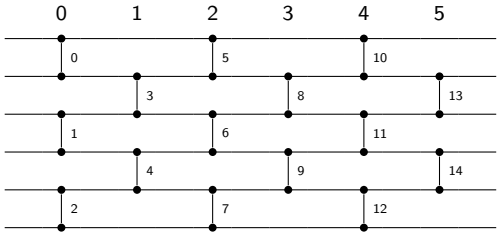

Figure 3.2: *A network representing the odd-even transposition sort for* $n = 6$ *with depth* 6 *and size* $\binom{n}{2} = 15$.

3.2 Size and Depth in Construction Schemes

It is well known that any correct comparison sort performs at least $\Omega(n \cdot \log n)$ comparisons [1, p. 182]. This gives an asymptotic lower bound for the minimum size of sorting networks.

In fact Ajtai, Komlós and Szemerédi (AKS) gave a construction scheme for sorting networks with size $O(n \cdot \log n)$ [32] such that the minimum size is asymptotically equal to $n \cdot \log n$ up to a constant factor. However AKS networks remain theoretical, due to the large constants hidden in the big O bound.

There are sorting algorithms with a lower time complexity such as radix sort, but they work under different assumptions.

A layer comprises at most $\lfloor \frac{n}{2} \rfloor$ comparisons. The minimum number of comparisons in a sorting network is $\Omega(n \cdot \log n)$. Hence the minimum depth is $\Omega(\log n)$. Similar to the size the AKS construction scheme gives a corresponding network with depth $O(\log n)$, but the construction remains impractical [32].

Instead construction schemes like Batcher's bitonic or odd-even mergesort [33] or the pairwise sorting network [34] have proven to be practical. They are designed for cases where n is a power of 2. They are only depth optimal up to $n = 2^3 = 8$ and the latter

Sorting network	Size	Depth
Bubblesort	$\binom{n}{2}$	$2n-3$
Odd-even transposition sort	$\binom{n}{2}$	n
Bitonic mergesort	$\frac{1}{2}n\binom{\log(n)+1}{2}$	$\binom{\log(n)+1}{2}$
Pairwise sorting network	$\frac{1}{2}n\binom{\log n}{2}+n-1$	$\binom{\log(n)+1}{2}$
Odd-even mergesort	$\frac{1}{2}n\binom{\log n}{2}+n-1$	$\binom{\log(n)+1}{2}$
AKS sorting network	$O(n\log n)$	$O(\log n)$

Table 3.1: *Various construction schemes of sorting networks. For the bitonic and odd-even mergesort as well as the pairwise sorting network we assume that n is a power of 2. The AKS network is asymptotically optimal.*

two schemes are only size optimal up to $n = 2^3 = 8$. The size of the bitonic mergesort scales worse with n than the latter two schemes. For details see Table 3.1.

4 Optimality Proofs for Small Networks

In this section we perform a simple analysis of the complexity of sorting network optimization. We trace previous results for small n and observe the success of SAT solver assisted proofs. In particular we take a closer look at the most successful approach for optimal depth.

4.1 Complexity of Sorting Network Optimization

We introduced the size and depth optimization problems in Section 1.3 as Problem 1 and Problem 2 respectively. The related decision problems further fix the size s or depth d respectively.

3. Is there a sorting network with n channels and size s? Or equivalently can s data-oblivious comparisons sort n elements?

4. Is there a sorting network with n channels and depth d? Or equivalently can d parallelized steps sort n elements where each step is made up of any number of independent data-oblivious comparisons?

Note that increasing n never decreases the minimal size or depth respectively. Because given a sorting network for $n + 1$ channels we can remove the last channel to obtain a comparator network with n channels and the same size and depth. To see that this comparator network sorts, we observe that any run with n inputs corresponds to a run of the original sorting networks where the same first n inputs are the same and the last input is the maximum of the first n.

Hence we can reduce the optimization problem to the related decision problem in polynomial time using binary search, i.e. we reduce Problem 1 to Problem 3 and Problem 2 to Problem 4. However this is negligible as we can solve Problem 3 only for $n \leq 10$ and Problem 4 only for $n \leq 17$ (see Table 1.1 in Section 1.3).

A standard argument to classify a problem's complexity utilizes nondeterminism. I.e. if there is a network for n channels with depth d, we choose it nondeterministically and then verify that it sorts. One might think that this puts the decision Problem 4 (or

Problem 3 respectively) into NP. However this requires verification in polynomial time. This is refuted by a result from Parberry in 1991. It states that verification of a sorting network is coNP-complete [35]. Hence we have to amend the prediction e.g. by granting access to a coNP oracle. This puts the two decision variants of the problems 3 and 4 into the second level of the polynomial hierarchy Σ_2^p [36, p. 91-100].

Problems in the second level of the polynomial hierarchy alternate the quantifiers $\exists$ and $\forall$. We ask whether there is a comparator network with fixed dimensions such that all network runs sort. The complements of Σ_2^p are in the class Π_2^p and alternate the quantifiers in the reverse order. We ask whether for each comparator network with fixed dimensions there is a network run that does not sort. A positive answer proves nonexistence of a sorting network with fixed dimensions.

4.2 Previous Optimality Proofs

In this section we trace the results in Table 1.1 and related work.

Knuth collected a list of small networks [1, p. 227, 229] by various authors up to $n \leq 16$. Their optimality was settled for $n \leq 8$ [37].

There are construction schemes to extend a sorting network to a sorting network that takes one more input. One such scheme that adds only very few comparators gives the inequality $s_{n+1} \geq s_n + \lceil \log n \rceil$ where s_n denotes the minimum size of a sorting network with n inputs [38]. This inequality derives the optimal size lower bounds for $n > 10$ in Table 1.1.

Besides that inequality the methods to bound the optimal size or depth of a sorting network are computer assisted. Evolutionary algorithms have been utilized to find sorting networks [13, 39, 40]. Finding a sorting network gives an upper bound on the optimal size and depth. However evolutionary algorithms are incapable of finding new lower bounds as this requires an exhaustive search.

In 1989 an exhaustive search was performed on a supercomputer to show the networks presented by Knuth are depth optimal for $9 \leq n \leq 10$ [10]. Decades later in 2013 the depth optimality of the remaining networks was settled for $n \leq 16$ by Bundala and Závodný (BZ) [11]. Improving that work the size optimality for $9 \leq n \leq 10$ was settled in 2014 [41] and in 2015 a depth lower bound for 17 and new depth upper bounds for 17, 19 and 20 were found [42]. The current upper bound for $n = 18$ was found in 2009 [43]. In 2016 another upper bound was found for $n = 24$ such that the upper bound for $21 \leq n \leq 24$ is 12 [44].

4.3 SAT Solver Assisted Optimality Proofs

Aside from Parberry's algorithm to prove depth optimality for $n \leq 10$ all new proofs of lower bounds for larger n utilized Boolean SAT-solvers. Whereas BZ reduced Problem 4 to Boolean SAT in 2013 to prove lower bounds for $n \leq 16$ with a SAT-solver, Gregg and Marinov proved lower bounds without using a SAT-solver in 2014 [45]. They state that their algorithm and related techniques were developed independent from BZ [46]. However they only proved depth optimality for $n \leq 12$.

I do not dare to argue whether e.g. SAT-solving might be superior to other exhaustive search techniques. There are several caveats to any such comparison attempts. Different authors used vastly different hardware. This includes clusters of 144 cores [47] or even supercomputers [10], i.e. expensive inaccessible hardware. The publications occured over the span of decades and a lack of interest may be as good of an explanation for the gap between 1989 and 2013 as a lack in technique. Some authors ran computations for more than a month [12], while others did not. Further progress in computing power or in the domain of SAT-solvers occured and continues to occur non-linearly. Additionally the search space scales exponentially with n but was pruned differently in each approach. This makes it difficult to compare results for different n because for upper bounds the time variance depends on the size of the pruned search space. Even lower bounds are affected similarly by the inherent randomness in SAT solving. Hence rather than arguing whether SAT-solving may be superior to other techniques, I want to point out that my first exposure to optimality proofs of sorting networks introduced me to the SAT-solving approach due to its success in the current decade.

In this section we describe BZ's Boolean SAT encoding. It reduces Problem 4 to Boolean SAT. The use of Boolean SAT Solvers is facilitated by the zero-one principle. It states that it suffices to consider sorting Boolean values.

Theorem 6 (Zero-One Principle)
A generalized comparator network is a generalized sorting network iff it sorts all vectors in $\mathbb{B}^n$, that is all vectors of zeros and ones.

Proof.

$\Longrightarrow$ If a generalized network sorts all vectors in $\mathbb{N}^n$, then it sorts all vectors in $\mathbb{B}^n$, because $\mathbb{B}^n$ is a subset of $\mathbb{N}^n$.

$\Longleftarrow$ We prove by contradiction that a generalized network is a sorting network if it sorts all vectors of zeros and ones. Assume that there is a generalized network N that sorts all vectors of zeros and ones, but not some vector $\overline{x}$ in $\mathbb{N}^n$. Then the output is not sorted at some position p, i.e. $y_p > y_{p+1}$ where $\overline{y} = N(\overline{x})$. We define

the function $f : \mathbb{N} \to \mathbb{N}$ as

$$f(v) = \begin{cases} 0, & v < y_p \\ 1, & v \geq y_p. \end{cases}$$

Note that $f(\overline{y})$ is not sorted at position p, since $f(y_p) = 1 > 0 = f(y_{p+1})$

We show that $N(f(\overline{x})) = f(\overline{y})$, which contradicts the assumption that the vector $f(\overline{x})$ of zeros and ones is sorted by N. Specifically we show that generalized networks commute with f under application to vectors, i.e. $N(f(\overline{x})) = f(N(\overline{x}))$.

It suffices to show that each comparator $c = (i, j)$ in N commutes with f under application, that is $c(f(\overline{x})) = f(c(\overline{x}))$. In particular we show that f commutes with computing the minimum output of c under application to any vector $\overline{z}$, i.e. $f(\min(z_i, z_j)) = \min(f(z_i), f(z_j))$. The case for the maximum output is analogous.

We assume without loss of generality that $z_i \leq z_j$ such that $\min(z_i, z_j) = z_i$. Note that f is monotone such that $f(z_i) \leq f(z_j)$ iff $z_i \leq z_j$. Thus we have $\min(f(z_i), f(z_j)) = f(z_i)$.

$$f(\min(z_i, z_j) = f(z_i) = \min(f(z_i), f(z_j)) \qquad \square$$

4.3.1 Prerequisite Constraints

First I introduce some prerequisite constraints that I later use to describe the reduction of the decision problems to Boolean SAT in the state of the art approach.

$$\begin{aligned} \text{atLeastOne}(S) &= \bigvee_{l \in S} l \\ \text{atMostOne}(S) &= \bigwedge_{\substack{l_1, l_2 \in S \\ l_1 \neq l_2}} \neg l_1 \vee \neg l_2 \\ \text{exactlyOne}(S) &= \text{atLeastOne}(S) \wedge \text{atMostOne}(S) \end{aligned}$$

To ensure that in a satisfying assignment the value of literal $l_{\min}$ is the minimum of the

values of all the literals l_0, l_1 up to l_k we define a min and analogously a max constraint.

$$\begin{aligned}
\min(l_{\min}, \{l_0, l_1, \ldots, l_k\}) &= (l_{\min} \vee \neg l_0 \vee \neg l_1 \vee \cdots \vee \neg l_k) \\
&\quad \wedge (\neg l_{\min} \vee l_0) \\
&\quad \wedge (\neg l_{\min} \vee l_1) \\
&\quad \ldots \\
&\quad \wedge (\neg l_{\min} \vee l_k) \\
\max(l_{\max}, \{l_0, l_1, \ldots, l_k\}) &= (\neg l_{\max} \vee l_0 \vee l_1 \vee \cdots \vee l_k) \\
&\quad \wedge (l_{\max} \vee \neg l_0) \\
&\quad \wedge (l_{\max} \vee \neg l_1) \\
&\quad \ldots \\
&\quad \wedge (l_{\max} \vee \neg l_k)
\end{aligned}$$

4.3.2 Encoding

The network is encoded as a sequence of unordered layers. The variable $g^k_{i,j}$ indicates that there is a comparator gate between channels i and j in layer k. Notably the encoding only considers standard comparator networks such that $i < j$. The variable u^k_i indicates that there is no comparator using channel i in layer k.

To ensure that every channel is used by at most one comparator in a layer and if necessary is marked as unused, we add the valid constraint.

$$\begin{aligned}
\text{standGate}^k_{i,j} &= \begin{cases} g^k_{i,j}, & \text{if } i < j \\ g^k_{j,i}, & \text{if } i > j \end{cases} \\
\text{valid} &= \bigwedge_{i \in [n]} \bigwedge_{k \in [d]} \text{exactlyOne}\Big(\{u^k_i, \text{standGate}^k_{i,j} \mid j \in [n], j \neq i\}\Big)
\end{aligned}$$

Further to ensure that any input vector $\overline{x}$ is sorted, we introduce a variable ${}^{\overline{x}}v^k_i$ for the value on channel i before layer k. Since layers are enumerated from zero, the network output value on channel i is modeled by ${}^{\overline{x}}v^d_i$.

We add constraints such that the values before the first layer match $\overline{x}$, values between layers are updated correctly by the comparators and the values after the last layer are

in sorted order.

$$
\begin{aligned}
{}^{\overline{x}}\text{update} = & \bigwedge_{\substack{i,j\in[n]\\ i<j}} \bigwedge_{k\in[d]} g_{i,j}^k \rightarrow \left(\min({}^{\overline{x}}v_i^{k+1}, \{{}^{\overline{x}}v_i^k, {}^{\overline{x}}v_j^k\}) \wedge \max({}^{\overline{x}}v_j^{k+1}, \{{}^{\overline{x}}v_i^k, {}^{\overline{x}}v_j^k\})\right) \\
& \wedge \bigwedge_{i\in[n]} \bigwedge_{k\in[d]} u_i^k \rightarrow \left({}^{\overline{x}}v_i^{k+1} \leftrightarrow {}^{\overline{x}}v_i^k\right) \\
{}^{\overline{x}}\text{values}_{\overline{y}}^k = & \bigwedge_{i\in[n]} \begin{cases} \neg\, {}^{\overline{x}}v_i^k, & \text{if } y_i = 0 \\ {}^{\overline{x}}v_i^k, & \text{if } y_i = 1 \end{cases} \\
{}^{\overline{x}}\text{sorts} = & \ {}^{\overline{x}}\text{values}_{\overline{x}}^0 \wedge {}^{\overline{x}}\text{update} \wedge {}^{\overline{x}}\text{values}_{\text{sort}(\overline{x})}^d
\end{aligned}
$$

A sorting network with n channels and depth d exists iff the following formula is satisfiable:

$$\text{valid} \wedge \bigwedge_{\overline{x}\in\mathbb{B}^n} {}^{\overline{x}}\text{sorts}$$

This naive SAT-encoding is impractical insofar as there are exponentially many, that is 2^n, input vectors in $\mathbb{B}^n$. Nonetheless experience has shown that a subset of inputs from $\mathbb{B}^n$ suffices to either prove unsatisfiability or come up with a sorting network for small n (≤ 17) [11, 48].

4.3.3 Counterexample Guided Inductive Synthesis

The encoding from the previous section can be reduced to a still exponential but more feasible size by considering a subset of the inputs. The encoding scheme was originally used to combine the encodings of several subnetworks. In that case we only consider inputs of the form $0^a\overline{z}1^b$ where $\overline{z}$ is an input to a subnetwork. This subnetwork optimization was used to prove nonexistence of a sorting network for given n and d [11]. We skip a closer study of this technique and instead consider an alternative approach that also reduces the number of encoded inputs and can even prove existence of a sorting network.

Assume that a SAT-Solver finds a satisfying assignment, that is a network, for the encoding with sort constraints for only a subset of the inputs. The found network may not sort some inputs from the complementary subset. To sidestep this problem, we look for such a counterexample input $\overline{x}$. If there is no counterexample the network is a sorting network. Otherwise we add the constraint ${}^{\overline{x}}\text{sort}$ to the encoding. Any network satisfying the amended encoding sorts $\overline{x}$. This approach of iteratively synthesizing networks and finding counterexamples in alternation is known as *Counterexample Guided Inductive Synthesis* [49].

Finding a counterexample input is in NP [35] and hence can be reduced to SAT. We employ a second solver to find counterexamples. The encoding comprises a specialisation of the update constraint and an additional constraint to ensure the output it not sorted. We disallow each sorted vector $0^{n-b}1^b$ with b in $[n]$ by adding its pointwise negation $1^{n-b}0^b$ as the clause $\text{values}^d_{1^{n-b}0^b}$. As the encoding does not scale to an exponential size, the time spent by the counterexample solver is neglectable with respect to that spent by the network solver. Incremental SAT solving allows us to keep learned clauses from previous iterations [50].

Minimal Counterexample Inputs

We observe that leading zeros in an input to a standard network are never swapped by a comparator. The same applies for trailing ones. Thus we do not encode them, e.g. the number of value variables ${}^{\overline{x}}v_i^k$ for an input $\overline{x}$ with a leading zeros and b trailing ones decreases from $n \cdot (d+1)$ to $(n-a-b) \cdot (d+1)$. Specifically if $\overline{x} = 0^a\overline{z}1^b$, we call $\overline{z}$ the window of $\overline{x}$ and we call $|\overline{z}|$ the window size.

Hence when solving for counterexamples, we prefer minimal counterexamples, e.g. counterexample inputs with minimal window size.

4.3.4 Two-Layer Prefix

BZ indicate that pruning the search space by symmetries is essential in Parberry's proof for $n = 9$ and proceed similarly in their own work. However BZ only utilize symmetry based pruning wrt the first two layers of the network. As a side note while BZ used SAT-solver, any other exhaustive search technique should similarly handle these symmetries efficiently.

A symmetry induces an equivalence relation between standard comparator networks where either all members or none of each equivalence class are sorting. It suffices to consider representative networks of each equivalence class when proving (non-)existence of a sorting network for a given input size n and depth bound d. Previous approaches utilize this insight to split all two layer prefixes into equivalence classes and prove (non-)existence for a representative prefix of each class [11, 12].

We recall from Section 2.1 that we can untangle any generalized π-sorting network to a standard sorting network. Furthermore by permuting the channels of any standard sorting network with some permutation π we can obtain up to $n!$ different generalized π-sorting networks. There is a standard sorting network symmetry that permutes the channels and then untangles the resulting generalized network to a standard network.

We give a formal definition of this symmetry in Section 5.1, as this thesis is concerned with breaking it in all layers.

To reduce the number of prefix representatives BZ consider the set of possible output vectors outputs(N) of a network N. In the words of Codish et al. two-layer prefix networks may *subsume* each other [12]. A network N subsumes a (prefix) network N' if outputs(N) $\subseteq$ outputs(N'). This implies that if N' can be extended to a sorting network, then so can N. Thus it suffices to consider only whether N can be extended to a sorting network with d layers. Actually the notion of subsumption is defined more generally as $\pi(\text{outputs}(N)) \subseteq \text{outputs}(N')$ for some channel number permutation π such that N subsumes N' if it can be permuted and untangled into a network N'' for that outputs(N'') $\subseteq$ outputs(N'). Subsumption yields a partial order where $N \leq N'$ if N subsumes N'. To reduce the number of two-layer prefixes only minimal representatives are considered. Unfortunately the problem of deciding whether a network subsumes another network is GI-complete [51].

BZ showed that we can require the first layer to be maximal, i.e. there must be the maximal number of comparators $\lfloor \frac{n}{2} \rfloor$ in the first layer [11]. BZ further reduce the number of representatives by banning certain *saturated* extensions to two layers. While BZ's notion of saturation is syntactic, Codish et al. adopted this term and gave it the semantic meaning whereby in simplified terms a saturated second layer neither contains redundant comparators nor can a non-redundant comparator be added [52]. Codish et al. conjecture that for different saturated two-layer prefixes N and N' it holds that outputs(N) $\not\subset$ outputs(N').

Lastly BZ consider another form of sorting network symmetry called reflection. The idea is that we map every comparator in a standard sorting network by $(i, j) \mapsto (n-i-1, n-j-1)$. This reflects every comparator across the center channel for uneven n or an imagined center channel for even n (see Figure 4.1). Since this is the same network but reflected, it sorts in the reverse order, i.e. it is a generalized reverse-sorting network where $\mathsf{reverse} = \left(\begin{smallmatrix} 0 & 1 & \dots & n-2 & n-1 \\ n-1 & n-2 & \dots & 1 & 0 \end{smallmatrix}\right)$. That is because every input vector $\overline{x}$ is sorted like the input $\mathsf{reverse}(\overline{x})$ in the reflected network. Specifically the vector $\overline{y}$ at any stage in the network is equal to $\mathsf{reverse}(\overline{y})$ in the same stage in the reflected network.

Typically we are restricted to standard networks. Thus we observe that mapping every comparator in a standard network to a max-min comparator by $(i, j) \mapsto (j, i)$ yields a generalized reverse-sorting network too. That is because every input vector $\overline{x}$ is sorted like the input $\mathsf{negate}(\overline{x})$ in the network with max-min comparators. Specifically the vector $\overline{y}$ at any stage in the network is equal to $\mathsf{negate}(\overline{y})$ in the same stage in the network with max-min comparators. In this case negate is the pointwise negation, e.g. $\mathsf{negate}(011011101) = 100100010$.

We can compose these mappings to obtain the reflection $(i, j) \mapsto (n-j-1, n-i-1)$ that maps standard networks to standard networks. Figure 4.1 shows an example.

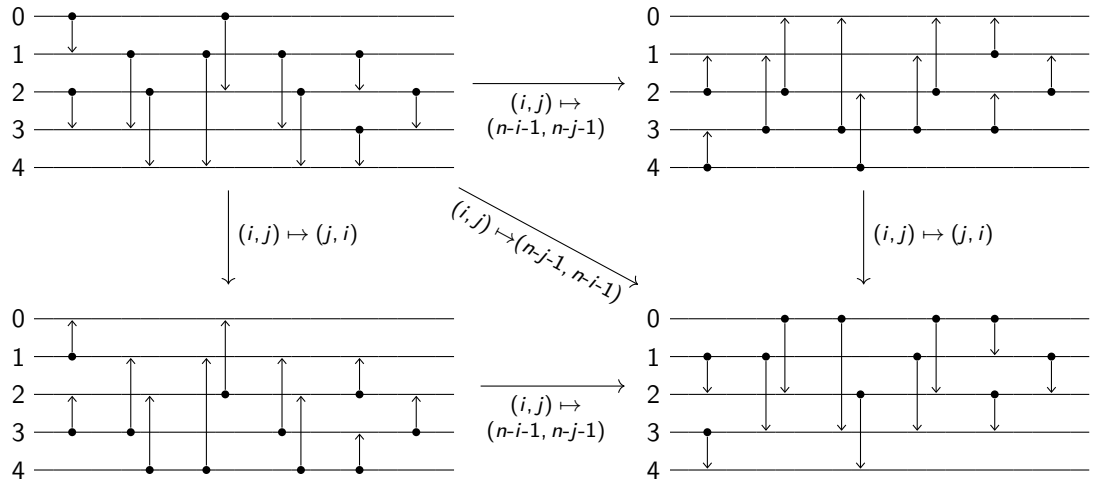

Figure 4.1: *A reflection symmetry.*

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\lvert G_n \rvert$	10^0	10^0	10^1	10^1	10^1	10^2	10^2	10^3	10^3	10^4	10^5	10^5	10^6	10^7	10^7	10^8	10^8	10^9	10^{10}
$\lvert R_n \rvert$	1	1	2	4	5	8	12	22	21	48	50	117	94	262	211	609	411	1367	894

Table 4.1: *G_n is the set of all possible two-layer prefixes. R_n is the set of two-layer prefixes that remain after symmetry based pruning [12]. For $\lvert G_n \rvert$ the order of magnitude is listed instead.*

Reflection is a symmetry in the state of the art encoding where we permute the literals by σ such that

$$\begin{aligned}\sigma(g^k_{i,j}) &= g^k_{n-j-1,n-i-1}\\ \sigma(u^k_i) &= u^k_{n-i-1}\\ \sigma({}^{\overline{x}}v^k_i) &= \neg^{\,\mathrm{reverse}(\overline{x})}v^k_{n-i-1}\end{aligned}$$

and Boolean consistency is preserved. Reflection is a syntactic symmetry if for every encoded input $\overline{x}$ the reversed and negated input $\mathsf{negate}(\mathsf{reverse}(\overline{x})) = \mathsf{reverse}(\mathsf{negate}(\overline{x}))$ is encoded too.

Table 4.1 demonstrates the effectiveness of symmetry based pruning. It shows concrete numbers of prefixes that remain after pruning by all symmetries, that is permuting and untangling, saturation and reflection.

It was suggested to extend the fixation to the first three layers [53]. But while the computation of remaining two-layer prefixes was made performant in [52], the same approach is not applicable for three-layer prefixes. On the other hand upper bounds do not require exhaustive consideration of all two-layer prefixes. In this case typically a

single prefix is handcrafted for up to 5 layers [42, 44].

In the case of size optimality there are no layers. Instead the first k comparators are fixed [41]. While Frăsinaru and Răschip improved the computation of the remaining k-comparator prefixes in 2017 [47], it still took them more than five days for $n = 9$ to compute the remaining 14-comparator prefixes. According to them moving to $n = 10$ seems not feasible.

We can reduce the size of the encoding by picking the representative prefixes in a way that minimizes the total window size over all vectors in $\mathbb{B}^n$ after application of the prefix [12].

4.3.5 Miscellaneous Improvements

There are a number of additional improvements. This includes additional redundant clauses and variables to facilitate propagation [12]. There is also empirical evidence that in the case of proving nonexistence, it is beneficial to add enough inputs initially such that there is only one synthesis iteration [42]. In this case prefix representatives are picked in a way that minimizes the total window size over all initial input vectors instead of all vectors in $\mathbb{B}^n$.

Actually symmetry breaking constraints have been added to the state of the art encoding before. Codish et al. analyzed the last layers wrt necessary conditions in a sorting network. They added symmetry breaking constraints for the last two layers of the network, i.e. they considered two-layer suffix representatives. In particular they introduced a last layer normal form [12, 54]. In my bachelor's project we applied symmetry breaking constraints too [55]. Our symmetry constraints attempted to disallow redundant comparators by tracking with auxiliary variables whether for the values on channels i or j in layer k or l respectively it holds that $v_i^k \leq v_j^l$ for all inputs to the network. If this is the case for some layer $k = l$, then the comparator between i and j in that layer is redundant, hence disallowed. Our method did not completely disallow redundant comparators, but we conjecture it completely disallows redundant comparators in the first two layers.

5 Networks Symmetries

In this section we formalize various symmetries between generalized comparator networks. Section 4.3.4 already introduced the reflection symmetry. We consider another symmetry mentioned in Section 4.3.4 that permutes and untangles the channels. For that we formalize permutations of channels and twists of comparators in Section 5.1. Specifically we analyze how a novel twisting symmetry relates to permuting and untangling the channels in Section 5.2. Finally we consider yet another symmetry that permutes parallel comparators in Section 5.3.

5.1 Permuting and Untangling Channels

We can permute the channels of any standard sorting network to obtain a generalized π-sorting network, which we can untangle to another standard sorting network. In this section we formalize the permutation of channels as well as the the standardization transformation called untangling.

Note that any standard π-sorting network is necessarily a standard sorting network. Previous work utilized this to prove untangling correct [6]. We call a transformation of standard comparator networks a correct standard sorting network symmetry if it preserves the property of being a sorting network.

However we cannot decompose the standard network symmetry that permutes and untangles into two correct network symmetries where one permutes and the other untangles. That is because the intermediate network is a generalized network, i.e. it is possibly nonstandard. Additionally we consider a SAT encoding of a generalized comparator network in Section 6.4. That is why we have to define all of the following network symmetries on generalized instead of standard networks.

So far we have only defined correctness of symmetries on standard networks. As we extend this notion of correctness from standard to generalized networks, we discover different definitions for a generalized sorting network in previous work. One definition considers any generalized π-sorting network a generalized sorting network [11], another restricts the definition to actual generalized sorting networks, that is the case $\pi = \mathsf{id}$ [12], while yet another definition may explicitly distinguish between the two definitions

by drawing on the auxiliary definition of a π-sorting network [6].

Thus we distinguish between two notions of correctness. We call a generalized network symmetry *correct under permuted outputs* if the following holds: A network is π-sorting for some π iff the related network is π'-sorting for some π'. That is in contrast to *strong correctness* that additionally requires $\pi = \pi'$.

We also define a dual tangling transformation. In Section 5.2 we relate it to permuting and untangling. In particular we show that it relates the same standard networks as permuting and untangling. Moreover we see that it, in contrast to permuting and untangling, is strongly correct on generalized networks.

5.1.1 Permuting Channels

First we formalize the operation to permute the channels of a network. In particular we permute the channel numbers that are components of each comparator in the network. Recall that the n channels in a network are numbered from 0 to $n-1$.

A permutation of channel numbers π from $\mathsf{Sym}([n])$ acts on a comparator $c = (i, j)$ with the action $\bullet : \mathsf{Sym}([n]) \times [n]^2 \to [n]^2$, that is

$$\pi \bullet c = (\pi(i), \pi(j)).$$

Utilizing this action the same permutation π may act on a network $N = c_0, c_1, \ldots, c_{s-1}$ with the action $\diamond : \mathsf{Sym}([n]) \times \mathcal{N}_n^s \to \mathcal{N}_n^s$, that is

$$\pi \diamond N = \pi \bullet c_0, \pi \bullet c_1, \ldots, \pi \bullet c_{s-1}$$

The first transformation in Figure 5.2 shows an example application of $\diamond$ to permute the channels of a network.

Note that this definition faithfully represents $\mathsf{Sym}([n])$ as a group of network transformations and vice versa represents that very group of network transformations that permute channels as $\mathsf{Sym}([n])$.

Further this induces an equivalence relation $P \subseteq \mathcal{N}_n^{s2}$. The symmetry P relates any network N with the network N' that is obtained from N by permuting its channels. The network $\pi \diamond N$ is a $(\pi \circ \sigma)$-sorting network if N is a σ-sorting network. Thus a permutation of the channel number π acting via $\diamond$ is correct under permuted outputs. It is not strongly connect because it may be that $\pi \neq \pi \circ \sigma$.

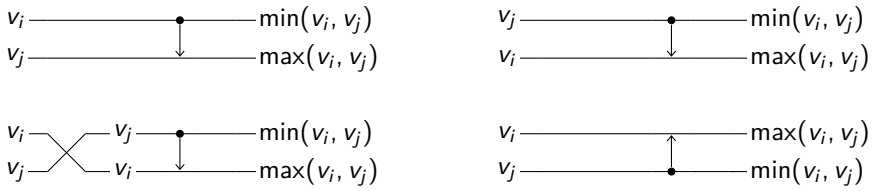

Figure 5.1: *The top-left picture shows a min-max comparator. The bottom-left picture shows a tangling of the channels before that comparator. The top-right picture shows an untangling by swapping the inputs. The bottom-right corner shows an untangling by swapping the outputs.*

5.1.2 Twists

Next we turn to the untangling operation that transforms any generalized network into a standard network. It iterates through the comparators of the network one by one in ascending order and untangles each max-min comparator to a min-max comparator such that finally there is no max-min comparator left, i.e. the network is standard. The comparators are untangled in order and untangling never affects the prefix network preceding the comparator.

The intuition behind the tangling is that we can swap the inputs of a comparator by reconnecting the input wires (see Figure 5.1) while preserving the output of the comparator. This works because the comparator does not discriminate between the inputs. However doing so entangles the channels. Such entanglements are not captured by our formal model of a comparator network. We resolve this by untangling the channels. This is possible by either swapping the channel segments before the tangling or by swapping the channel segments after the tangling. The latter variant turns a comparator from max-min to min-max or vice versa from min-max to max-min.

We refer to the procedure that transforms a single comparator from max-min to min-max (or vice versa) as an output twist or a twist of the output channels of some comparator. We call the other transformation an input twist or a twist of the input channels. This one cannot standardize comparators but is a useful dual operation nonetheless as we later see in Section 5.2.

A permutation of the channels of the kth comparator may act on a network $N = c_0, c_1, \ldots, c_{s-1}$ such that it permutes the channels of c_k in either the prefix network up to c_{k-1} or the suffix network starting with c_k.

More precisely τ from $\mathsf{Sym}([2])$ is faithfully represented as a network transformation by the action $\triangleq_k : \mathsf{Sym}([2]) \times \mathcal{N}_n^s \to \mathcal{N}_n^s$ where k is from $[s]$. I.e. the action $\triangleq_k$ is parameterized by k such that it actually gives s different actions. Let $c_k = (k_0, k_1)$ and let $\pi_k : [n] \to [n]$

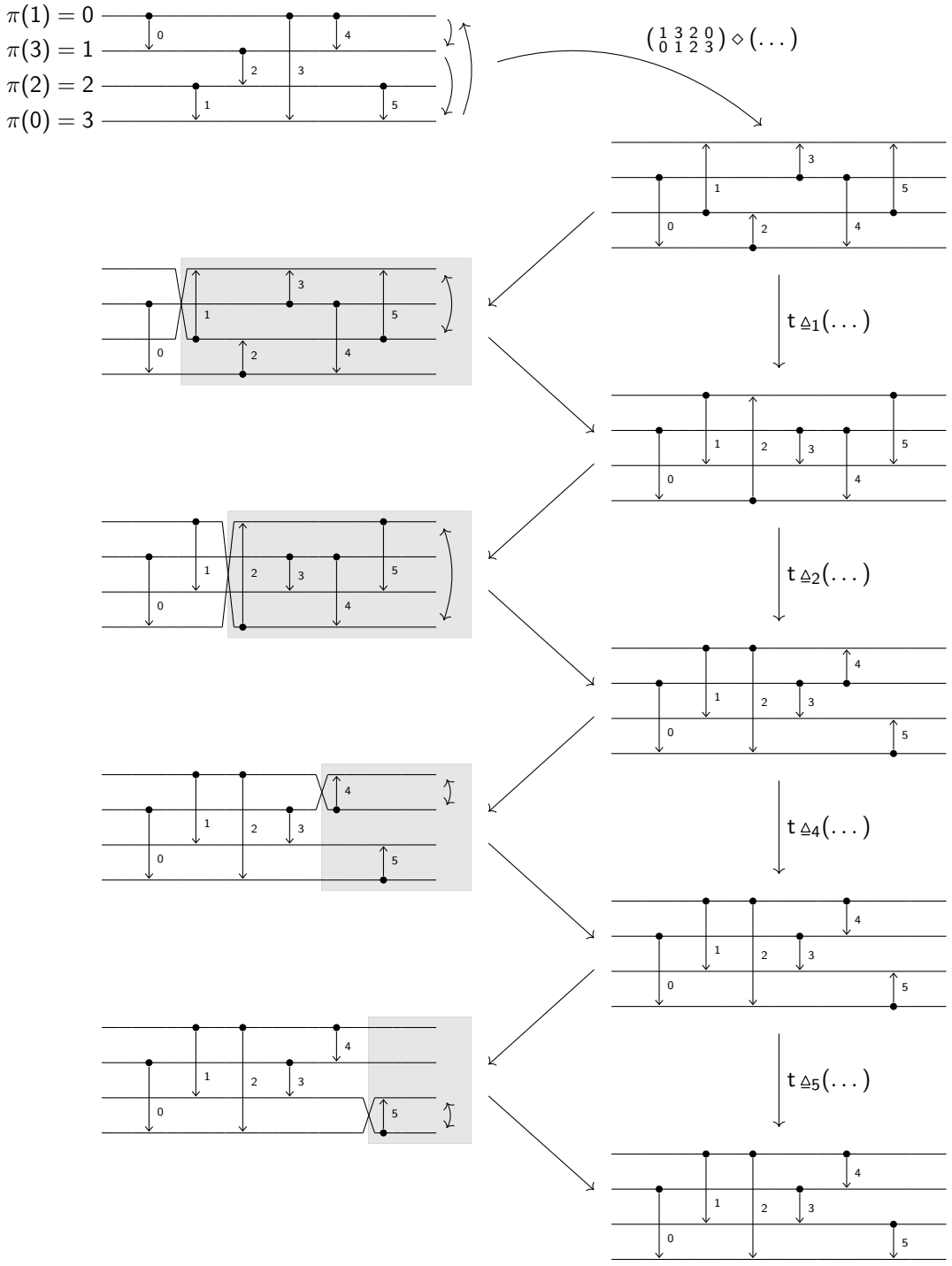

Figure 5.2: *A standard network whose channels are permuted and untangled to another standard network.*

be defined such that it permutes the channels of c_k, that is

$$\pi_k(i) = \begin{cases} k_{\tau(0)}, & i = k_0 \\ k_{\tau(1)}, & i = k_1 \\ i, & \text{otherwise.} \end{cases}$$

Then we define $\triangleq_k$ such that it permutes the output channels of c_k in the suffix network starting with c_k.

$$\tau \triangleq_k N = c_0, c_1, \ldots, c_{k-1} \,;\, \pi_k \diamond (c_k, c_{k+1}, \ldots, c_{s-1})$$

Note that any $\tau : \{0,1\} \to \{0,1\}$ from $\mathsf{Sym}([2])$ is either $\mathsf{id}(x) = x$ or $\mathsf{t}(x) = 1 - x$, as those two are the only elements in $\mathsf{Sym}([2])$. The former permutation id maps 0 to 0 and 1 to 1 while the latter permutation t swaps 0 with 1.

Figure 5.2 shows an example where the channels of a standard network are permuted and then untangled to another standard network by a sequence of output twists.

Likewise we define $\triangledown_k : \mathsf{Sym}([2]) \times \mathcal{N}_n^s \to \mathcal{N}_n^s$ with k from $[s]$ such that τ from $\mathsf{Sym}([2])$ permutes the input channels of c_k in the suffix network ending with c_{k-1}.

$$\tau \triangledown_k N = \pi_k \diamond (c_0, c_1, \ldots, c_{k-1}) \,;\, c_k, c_{k+1}, \ldots, c_{s-1}$$

This induces equivalence relations $O_k \subseteq \mathcal{N}_n^{s2}$ and $I_k \subseteq \mathcal{N}_n^{s2}$ corresponding to a twist of the outputs or inputs respectively. The equivalence O_k relates any network N with the network N' that is obtained from N by twisting the output channels of c_k. The equivalence I_k relates networks similarly but it twists the inputs instead of the outputs.

5.1.3 Conception of Sorting Networks and the Novelty of Input Twists

Although input twists cannot standardize comparators, they have actually been defined previously with the aim to standardize the network [12]. However the previous definition of the dual standardization transformation was not correct under permuted outputs as intended.

It was claimed that there is a dual standardization procedure that iterates through the comparators of the network in the reverse order, i.e. from the last comparator to the first. For every max-min comparator (i, j) the procedure swaps the channels i and j in the prefix network ending with that comparator (i, j). Note that in contrast our definition of input twists swaps i and j in the prefix network ending *before* that comparator (i, j). Only with the former definition that comparator (i, j) becomes min-max.

Figure 5.3: *Counterexample to the claim that input twists standardize comparators.*

Figure 5.3 shows a counterexample that disproves the correctness of this dual standardization procedure. The left network shows a generalized id-sorting network. Applying the dual standardization procedure swaps the only max-min comparator in the first layer. However the standardized network (shown on the right) is not a sorting network because the input vector 0101 is not sorted.

Aside from that I am not aware of any previous study of input twists. Then again work before the turn of the millennium sometimes contains networks with unresolved tanglings [1, 3, 38]. To put this proclaimed novelty of input twists into perspective, we trace how the conception of the term sorting network changed over the previous 60 years.

Initial work on sorting networks in 1962 was conceived as work about (data-oblivious) in-place comparison sorts [56]. The concept of a network was missing. It was introduced later. Batcher used the term sorting network to refer to an electronic circuit with comparator gates in 1968 [3]. He drew comparator networks as graphs where curved edges connect comparator gates. Although his pictures suggest there are separate paths through the network, i.e. channels, these channels are not simply twisted before or after some comparator but rather they may connect comparators arbitrarily within the network.

However later diagrams as they appear in Donald Knuth's compilation of sorting networks [1, p. 221] became the standard way to depict comparator networks. In Knuth's diagrams the channels are straight parallel lines, unlike previous depictions of comparator networks that sometimes contain bends or kinks of the channels, which may tangle them. Knuth diagrams ensure that we may use the channel numbers from the diagram in the formalization, as they reliably match the compared positions in the oblivious sorting algorithm. For example Billardi distinguishes between the graph topology of a sorting network and its line representation in the context of merging networks [57].

I believe the term untangling was first introduced in the current decade in 2014 by Bundala and Závodný in reference to the standardization tranformation detailed by Parberry in 1987 [6, p. 6-8] or Knuth in 1998 [1, p. 238, 667-668]. Neither of them gave any intuition for the untangling transformation. But in 2002 Choi and Moon formally defined a graph representation of a sorting network [13, 58] and claimed that the symmetry that permutes and untangles standard networks is enabled by an isomorphism between graph representations [13]. We address this in Section 7.

5.1.4 Composition of Channel Permutations with Twists

Lemma 7
Twists commute with channel permutations under composition. In detail any permutation π from $\mathsf{Sym}([n])$ acting on a network N in $\mathcal{N}_n^s$ commutes with any twist τ from $\mathsf{Sym}([2])$ at the kth comparator c_k in N, i.e. for any k from $[s]$ we have that

$$\tau \triangleq_k (\pi \diamond N) = \pi \diamond (\tau \triangleq_k N)$$
$$\tau \triangledown_k (\pi \diamond N) = \pi \diamond (\tau \triangledown_k N)\,.$$

Proof. We consider a proof for output twists only as the proof for input twists is analogous. We show that any permutation of channels π commutes with any twist τ at the kth comparator under composition when applied to any network N. We consider the case that τ is the swap t because otherwise if τ is id we know that id commutes with any transformation. Let $c_k = (i, j)$ be the kth comparator in N. Then t twists the channels i and j if t acts before π or otherwise t twists the channels $\pi(i)$ and $\pi(j)$ if it acts after π. Note that any channel other than i and j in N or $\pi(i)$ and $\pi(j)$ in $\pi \diamond N$ respectively is not affected by the action of t.

Thus we only check for commutativity with respect to the channels i and j and observe that in both cases channel i is mapped to $\pi(j)$ and channel j is mapped to $\pi(i)$. I.e. let $p : \{i, j\} \to \{\pi(i), \pi(j)\}$ map i and j like π and let $t : \{i, j\} \to \{i, j\}$ map them like $\triangleq_k$ applied before $\diamond$ and let $t' : \{\pi(i), \pi(j)\} \to \{\pi(i), \pi(j)\}$ map them like $\triangleq_k$ applied after $\diamond$ then

$$\begin{aligned} p \circ t &= \left(\begin{cases} i \mapsto \pi(i) \\ j \mapsto \pi(j) \end{cases}\right) \circ \left(\begin{cases} i \mapsto j \\ j \mapsto i \end{cases}\right) \\ &= \left(\begin{cases} i \mapsto \pi(j) \\ j \mapsto \pi(i) \end{cases}\right) \\ &= \left(\begin{cases} \pi(i) \mapsto \pi(j) \\ \pi(j) \mapsto \pi(i) \end{cases}\right) \circ \left(\begin{cases} i \mapsto \pi(i) \\ j \mapsto \pi(j) \end{cases}\right) \\ &= t' \circ p \end{aligned}$$

□

Definition 8
We define the relations $O \subseteq \mathcal{N}_n^{s2}$ and $I \subseteq \mathcal{N}_n^{s2}$. They relate any networks that are obtained from each other by only twisting inputs or outputs respectively.

$$O = O_0 \,;\, O_1 \,;\, \cdots \,;\, O_s$$
$$I = I_0 \,;\, I_1 \,;\, \cdots \,;\, I_s$$

Proposition 9
The relations O and I are equivalence relations.

Proof. We prove the property only for O since the proof for I works similarly.

We prove that O is an equivalence relation by induction on the size of the network. In particular we show that $O_{0,\dots,k} := O_0 \,;\, O_1 \,;\, \cdots \,;\, O_k$ is an equivalence relation for every k from $[s]$. In the base case we already know that O_0 is an equivalence relation. For the induction step we prove that the composition of two equivalence relations $O_{0,\dots,k-1}$ and O_k yields an equivalence relation $O_{0,\dots,k}$. By Proposition 5 this is the case if and only if $O_{0,\dots,k-1} \,;\, O_k = O_k \,;\, O_{0,\dots,k-1}$.

As every O_k is induced by $\mathsf{Sym}([2])$ acting on networks as output twists, it suffices to show that output twists commute with each other under composition, i.e. for all τ in $\mathsf{Sym}([2])$ acting on any N in $\mathcal{N}_n^s$ it holds that $\tau \triangleq_k (\tau \triangleq_l N) = \tau \triangleq_l (\tau \triangleq_k N)$. Without loss of generality we assume that $0 \leq l \leq k$. Specifically τ acts as a network transformation that is either the identity or an output channel twist that swaps channels in a suffix network. As the identity commutes with every transformation, we assume that τ is the swap $\mathtt{t}$ instead. Further we assume that $l < k$ because otherwise if $l = k$ the swaps would cancel each other such that the twists commute.

Let $N = c_0, c_1, \dots, c_{s-1}$. The comparators $c_0, \dots, c_{l-1}$ are modified by neither $\triangleq_l$ nor $\triangleq_k$. So let us consider the suffix network starting at c_l. The comparator c_l is not modified by $\triangleq_k$ such that $\triangleq_l$ invariably applies the same permutation to the channels in the suffix network starting with c_l. Thus the swap $\mathtt{t}$ represented by $\triangleq_k$ as a twist at c_k commutes with its representation by $\triangleq_l$ as a permutation on that suffix network by Lemma 7. □

Now we can compose P and O to the relation $P \,;\, O$. This relates all networks, in particular all standard networks, that are related by permuting and untangling. Note that untangling is a sequence of output twists of each comparator c such that c becomes a min-max comparator.

Proposition 10
The relation $P \,;\, O$ is an equivalence relation.

Proof. By Proposition 5 the composition $P \,;\, O$ is an equivalence relation if and only if it commutes, i.e. $P \,;\, O = O \,;\, P$. This follows immediately from Lemma 7 and Definition 8. □

5.2 Input Twists

In this section we consider input twists and contrast them with $P\,;O$. Actually I and O are exchangable with respect to $P\,;O$.

Proposition 11
Any transformation that is a permutation followed by a sequence of output twists can alternatively be achieved by some (other) permutation followed by a sequence of input twists. And vice versa any permutation followed by inputs twists can be realized by some permutation followed by output twists.

$$P\,;O = P\,;I$$

To prove this we first prove the following lemma.

Lemma 12
Any output twists can be alternatively achieved with only input twists and channel permutations. Likewise any input twists can be alternatively achieved with only output twists and channel permutations.

$$I \subseteq P\,;O$$
$$O \subseteq P\,;I$$

Proof. We only show that $I \subseteq P\,;O$. The proof for $O \subseteq P\,;I$ is analogous. It suffices to show that $I_k \subseteq P\,;O_k$ for all k, because by Lemma 7 the relation P commutes with any O_k under composition. For any relation in I_k there is an underlying network transformation that is either the identity or an input channel twist that swaps channels in a prefix network. In the case of the identity we have $I_k \subseteq P\,;O_k$ due to reflexivity of $P\,;O_k$. In the other case the channels i_k and j_k of the kth comparator $c_k = (i_k, j_k)$ are swapped in the prefix network before c_k.

Recall that the input or output twists at c_k apply the same permutation π_{i_k,j_k} of the channels in the prefix network or in the suffix network respectively. Here π_{i_k,j_k} is the permutation in $\mathsf{Sym}([n])$ that swaps the channel numbers i_k and j_k. We show that composing the input and output twists at c_k yields that very permutation π_{i_k,j_k} acting as a channel permutation on the whole network. That is for all N in $\mathcal{N}_n^s$ where the kth comparator works on the channels i_k and j_k it holds that

$$\begin{aligned} \mathsf{t} \triangleq_k (\mathsf{t} \triangledown_k N) &= \pi_{i_k,j_k} \diamond N \\ \underset{\substack{\mathsf{t} \triangledown_k (..) \\ \mathsf{t}=\mathsf{t}^{-1}}}{\iff} \qquad \mathsf{t} \triangledown_k N &= \mathsf{t} \triangleq_k (\pi_{i_k,j_k} \diamond N) \end{aligned}$$

With the swap t acting via $\triangleq_k$ as an output twist on both sides of the equation and the fact that t (and its representation) is self inverse, we can rearrange this to prove that $I_k \subseteq P\,;O_k$. Note that composed equivalence relations apply their underlying transformations in the order left to right, while composed left actions are applied in the order right to left.

First we show that $\triangledown_k$ and $\triangleq_k$ commute under composition for the same k to clarify the analogous proof for $O \subseteq P\,;I$ works, i.e.

$$\mathsf{t} \triangleq_k (\mathsf{t} \triangledown_k N) = \mathsf{t} \triangledown_k (\mathsf{t} \triangleq_k N)\,.$$

Note that the output twist action $\triangleq_k$ transforms the suffix network starting with c_k, while the input twist action $\triangledown_k$ transforms the complementary prefix network. The transformation by $\triangleq_k$ is independent of a potential preceding action of $\triangledown_k$, because $\triangledown_k$ does not modify c_k. In contrast $\triangleq_k$ modifies c_k from (i_k, j_k) to (j_k, i_k), but this yields the same transformation for $\triangledown_k$ because twisting the segments of the channels i_k and j_k is the same as twisting the segments of the channels j_k and i_k.

Specifically $\triangleq_k$ applies the channel permutation π_{i_k,j_k} to the suffix network. Since $\triangledown_k$ invariably applies the same transformation $\pi_{i_k,j_k} = \pi_{j_k,i_k}$ to the complementary prefix network, we have that $\mathsf{t} \triangleq_k (\mathsf{t} \triangledown_k N) = \pi_{i_k,j_k} \diamond N$. □

Now we can prove that $P\,;O = P\,;I$ as stated by Proposition 11.

Proof. We show that $P\,;O \subseteq P\,;I$. The proof for the reverse inclusion is analogous. By Lemma 12 we have that $P\,;O \subseteq P\,;P\,;I$. By Proposition 4 composition of P is idempotent. □

Actually we can even replace P by I in $P\,;O$. This is because by Lemma 12 we can simulate permutations with input and output twists. However this is only possible if comparators exist at the right places in the network. For example we cannot simulate any channel permutation other than id with input and output twists if there are no comparators. However the relations between sorting networks that we care about are not lost.

Proposition 13
The equivalence relation $(O\,;I)^*$ is finer than $P\,;O$ and is like $P\,;O$ correct under permuted outputs. Further $(O\,;I)^*$ is complete under permuted outputs relative to $P\,;O$. That is if a generalized π-sorting networks and a generalized π'-sorting network are related by $P\,;O$, then they are also related by $(O\,;I)^*$.

Proof. Let two networks N and N' be from $\mathcal{N}_n^s$. First we show that if they are related by $(O\,;I)^*$, then they are related by $P\,;O$, i.e. that $(O\,;I)^*$ is a finer equivalent relation than $P\,;O$. We utilize that $I \subseteq P\,;O$ from Lemma 12, that composition of P with O is commutative and that composition of P or O respectively is idempotent.

$$\begin{aligned}(O\,;I)^* &\underset{12}{\subseteq} (O\,;P\,;O)^* \\ &\underset{5,10}{=} P^*\,;O^* \\ &\underset{4,9}{=} P\,;O\end{aligned}$$

Next we conclude that $(O\,;I)^*$ is correct under permuted outputs because it is a subset of $P\,;O$, which is correct under permuted outputs.

For the completeness of $(O\,;I)^*$ under permuted outputs relative to $P\,;O$ we show the reverse direction, i.e. that a relation by $P\,;O$ implies a relation by $(O\,;I)^*$, under the premiss that each network has some respective π that makes it π-sorting. Clearly O is a subset of $(O\,;I)^*$. Thus it suffices to show that $P \subseteq (O\,;I)^*$.

As shown in the proof for Lemma 12 we can replicate the action $\diamond$ of a channel permutation π_{i_k,j_k} that swaps the channels i_k and j_k of the kth comparator with the composition of the twist actions $\triangleq_k$ and $\triangledown_k$ applied to swaps. Note that we cannot replicate the swap of any two channels this way, only if they share a comparator. Further note that letting some $\pi_{i,j}$ act as channel swap, changes which channels share a comparator.

However every channel is connected to any other channel via a sequence of comparators. That is because N and N' are π-sorting, i.e. they are able to channel any input on some channel to an output position on any channel according to their rank. Thus for any two channels a and b there is a path through the network that connects the input on a to the output on b and the comparators on that path connect a and b. The existence of this connection is independent of swapping the channels of some comparator because $O\,;I$ is correct under permuted outputs.

Even if two channels a and b do not share a comparator, they are at least connected by a path with a comparator sequence $c_l, c_m, \ldots, c_o, c_p$ such that we can replicate a swap of a and b in a network N with $\pi_{a,b} \diamond N = \mathsf{t} \triangledown_l (\mathsf{t} \triangleq_l \ldots \mathsf{t} \triangledown_p (\mathsf{t} \triangleq_p \ldots \mathsf{t} \triangledown_l (\mathsf{t} \triangleq_l N)))$. We assume without loss of generality that for each channel d on the path there is exactly one continuous segment where the path coincides with d. If there are two or more separate segments where the path coincides with the channel, then we are able to shortcut via the channel such that there is only one continuous segment.

We have shown in the proof for Lemma 12 that $\triangledown_k$ and $\triangleq_k$ commute under composition for the same k. Let the action $\diamond_k : \mathsf{Sym}([2]) \times \mathcal{N}_n^s \to \mathcal{N}_n^s$ represent a permutation of the two components of the kth comparator as a network transformation replicated by two

commuting twists.

$$\tau \diamond_k N = \tau \triangledown_k (\tau \triangleq_k N) = \tau \triangleq_k (\tau \triangledown_k N)$$

Note that unlike $\diamond$ the action $\diamond$ does not always swap the same channels as this depends on the channels of the kth comparator.

Let the lth comparator work on channels a and w, the mth on channels w and x, the oth on channels y and z and the pth on channels z and b. We assume that $c_l, c_m, \ldots, c_o, c_p = (a, w), (w, x), \ldots, (y, z), (z, b)$ without loss of generality.

First $\diamond_l$ swaps the channels a and w. This yields a network where the mth comparator still works on channel x, but compares it with channel a instead of w. Thus $\diamond_m$ swaps the channels a and x. This pattern of repeatedly swapping a continues until we swap the channels of the pth comparator a and b. Then we swap the channels of the oth comparator a second time. The first time it is (a, z) and we swap a and z and afterwards we swap a with b. Hence the oth comparator becomes (z, b) and we swap b and z the second time. Note that this changes the pth comparator from (b, a) to (z, a). Now we repeat the pattern of swapping b (instead of a) in the reverse direction such that we swap b and y next. This changes the oth comparator from (b, z) to (y, z). For any comparator with a channel number i initially in the uth component it holds that it reappears in the uth component after applying all twists like we have already seen for $i = z$ in the case of the oth comparator. Notable exceptions are the channels a and b, which are effectively swapped.

Recall that left actions are right associative:

$$\begin{aligned}
&\mathsf{t} \triangledown_l \mathsf{t} \triangleq_l \mathsf{t} \triangledown_m \mathsf{t} \triangleq_m \ldots \mathsf{t} \triangledown_o \mathsf{t} \triangleq_o \mathsf{t} \triangledown_p \mathsf{t} \triangleq_p \mathsf{t} \triangledown_o \mathsf{t} \triangleq_o \ldots \mathsf{t} \triangledown_m \mathsf{t} \triangleq_m \mathsf{t} \triangledown_l \mathsf{t} \triangleq_l N \\
&= \mathsf{t} \diamond_l \mathsf{t} \diamond_m \ldots \mathsf{t} \diamond_o \mathsf{t} \diamond_p \mathsf{t} \diamond_o \ldots \mathsf{t} \diamond_m \mathsf{t} \diamond_l N \\
&= \mathsf{t} \diamond_l \mathsf{t} \diamond_m \ldots \mathsf{t} \diamond_o \pi_{a,b} \diamond \pi_{a,z} \diamond \ldots \pi_{a,x} \diamond \pi_{a,w} \diamond N \\
&= \pi_{b,w} \diamond \pi_{b,x} \diamond \ldots \pi_{b,z} \diamond \pi_{a,b} \diamond \pi_{a,z} \diamond \ldots \pi_{a,x} \diamond \pi_{a,w} \diamond N \\
&= \pi_{a,b} \diamond N
\end{aligned}$$

As it is possible to replicate arbitrary swaps of channel numbers, we can compose them to replicate any permutation of channel numbers. We conclude that $P \subseteq (O\,;I)^*$. □

In fact we can completely replace $P\,;O$ with I to obtain a more precise variant of $P\,;O$ that discriminates between π-sorting networks and π'-sorting networks if $\pi \neq \pi'$.

Proposition 14
The equivalence relation I is finer than $(O\,;I)^*$ and strongly correct. Further $(O\,;I)^*$ is strongly complete relative to $(O\,;I)^*$ (and thus to $P\,;O$ by Proposition 13). That is if any two generalized π-sorting networks are related by $(O\,;I)^*$ (or $P\,;O$), then they are also related by I.

Proof. Let two networks N and N' be from $\mathcal{N}_n^s$. Clearly if they are related by I, then they are related by $(O\,;I)^*$, i.e. I is a finer equivalent relation than $(O\,;I)^*$.

Since $I \subseteq (O\,;I)^*$ and $(O\,;I)^*$ is correct under permuted outputs, I is too. We show that I is even strongly correct. Specifically we show that a π-sorting network and a π'-sorting network with $\pi \neq \pi'$ are never related by I. It suffices to show for all k that I_k never relates such networks. The identity cannot transform a π-sorting network to a π'-sorting network with $\pi \neq \pi'$. Note that the input twist action $\triangledown_k$ underlying I_k makes the channel permutation π_{i_k,j_l} act on the prefix network ending immediately before the kth comparator $c_k = (i_k, j_k)$ where π_{i_k,j_l} swaps the channels i_k and j_k.

Consider that we channel an input vector $\overline{x}$ through the aforementioned prefix network P_k yielding the output $P_k(\overline{x})$. For any such vector x we can obtain the same output if we apply the permutation π_{i_k,j_k} to both the input and the prefix network and finally to the output to undo the swap.

$$P_k(\overline{x}) = (\pi_{i_k,j_k} \circ (\pi_{i_k,j_k} \diamond P_k) \circ \pi_{i_k,j_k})\,(\overline{x})$$

Extending this by c_k to the network $P_k\,;c_k$ we can exploit that the input order to c_k does not matter. We can omit the final permutation to undo the swap, i.e. for every input vector $\overline{x}$ we have that

$$P_k\,;c_k(\overline{x}) = ((\pi_{i_k,j_k} \diamond P_k\,;c_k) \circ \pi_{i_k,j_k})\,(\overline{x}).$$

Assume that extending $P_k\,;c_k$ with a complementary suffix S_k yields a π-sorting network $P_k\,;c_k\,;S_k$, which gives the same output for all permutations of some input x. Then complementing $(\pi_{i_k,j_k} \diamond P_k)\,;c_k$ with the same suffix similarly yields a π-sorting network $(\pi_{i_k,j_k} \diamond P_k)\,;c_k\,;S_k$. In particular it will never yield a π'-sorting network with $\pi \neq \pi'$.

For the strong completeness relative to $(O\,;I)^*$ we show that a relation by $(O\,;I)^*$ implies a relation by I, under the premiss that both N and N' are π-sorting for the same π.

By Proposition 13 a relation by $(O\,;I)^*$ implies a relation by $P\,;O$, because $(O\,;I)^*$ refines $P\,;O$. By Proposition 11 the relation $P\,;O$ is equivalent to $P\,;I$. Note that I is strongly correct and N and N' are related by $P\,;I$. Then the underlying channel permutation must be the identity. Otherwise N and N' would not be π-sorting for the same π. As P represents the identity relation in this case, we can drop it. This proves that N and N' are related by I. □

In this section we have seen that I is a suitable replacement for $P\,;O$. We utilize this later in Section 6.3.1.

5.3 Permuting Parallel Comparators

We consider the example network $N = (0,1),(2,3),(0,2),(1,3),(0,1),(2,3)$. The first two comparators $(0,1)$ and $(2,3)$ are independent as they work on different channels. Due to their independence they can be applied in any order e.g. the network $N' = (2,3),(0,1),(0,2),(1,3),(0,1),(2,3)$ works the same as N. Next we define how to apply a permutation on the comparator indices to a network such that the permutation $(1,0,2,3,4,5)$ swaps the comparators c_0 and c_1 in N to obtain N'.

A permutation of comparator numbers ρ from $\mathsf{Sym}([s])$ acts on a generalized comparator network $N = c_0, c_1, \ldots, c_{s-1}$ with the action $\shortparallel : \mathsf{Sym}([s]) \times \mathcal{N}_n^s \to \mathcal{N}_n^s$, that is

$$\rho \shortparallel N = c_{\rho(0)}, c_{\rho(1)}, \ldots, c_{\rho(s-1)}.$$

Note that comparators are only permuted, but not deleted nor added nor made to work on different channels nor changed from min-max to max-min or reverse. Thus the number of min-max (or max-min respectively) comparators between two given channels remains unchanged. Yet an arbitrary permutation of the comparators may yield a different network as shown by ρ_{ba} in Figure 5.4.

Now we restrict ρ to reorder parallel comparators only, i.e. ρ preserves the order of sequential comparators. For that we first consider several partial orders on the comparators.

The set of shared channels of two comparators $c_k = (k_0, k_1)$ and $c_l = (l_0, l_1)$ in a network $N = c_0, \ldots, c_{s-1}$ is

$$c_k \sqcap c_l = \{k_0, k_1\} \cap \{l_0, l_1\}.$$

Further we use $<_i$ or $\leq_i$ to denote that a comparator precedes another on channel i, that is

$$\begin{aligned} c_k <_i c_l &\iff i \in c_k \sqcap c_l \text{ and } k < l \\ c_k \leq_i c_l &\iff i \in c_k \sqcap c_l \text{ and } k \leq l. \end{aligned}$$

We omit i to indicate that a comparator precedes another on *some* channel, that is

$$\begin{aligned} c_k < c_l &\iff c_k \sqcap c_l \neq \emptyset \text{ and } k < l \\ c_k \leq c_l &\iff c_k \sqcap c_l \neq \emptyset \text{ and } k \leq l. \end{aligned}$$

Likewise we write $\prec_i$ to indicate that a comparator *immediately* precedes another on a channel i, that is

$$c_k \prec_i c_l \iff c_k <_i c_l \text{ and } \nexists\, m \in [s].\ c_k <_i c_m <_i c_l$$

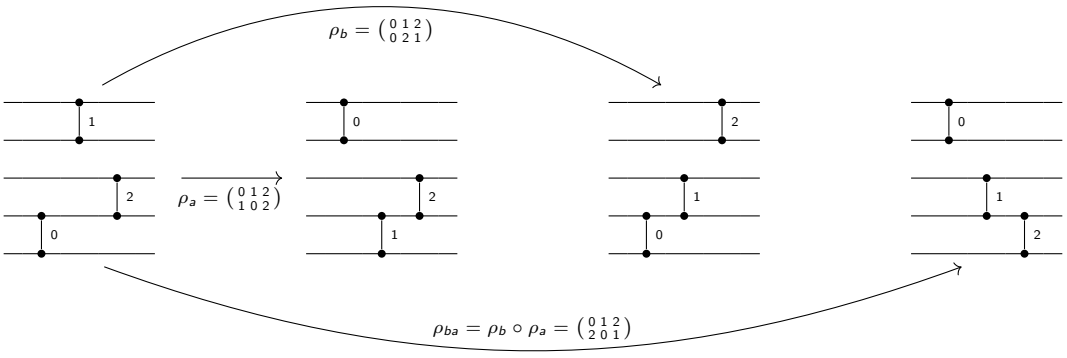

Figure 5.4: *Counterexample that the permutations underlying R are not closed under composition.*

Finally $\prec$ denotes that a comparator immediately precedes another on some channel i. This is even the case if the comparators share both channels and there is an intermediate comparator between them on the other channel.

$$c_k \prec c_l \iff \exists\, i \in [n].\ c_k \prec_i c_l.$$

Now we define a relation $R \subseteq \mathcal{N}_n^{s2}$ between networks. Two networks $N = c_0, c_1, \ldots, c_{s-1}$ and $N' = c'_0, c'_1, \ldots, c'_{s-1}$ are related by R if and only if N' is obtained from N by permuting its comparators by some permutation ρ from $\mathsf{Sym}([s])$ where ρ satisfies the following condition for all k and l in $[s]$: Comparator $c_{\rho(k)}$ precedes $c_{\rho(l)}$ in N if and only if c'_k precedes c'_l in N'. This ensures that ρ only changes the order of parallel comparators.

$$N\ R\ N' \iff \exists\, \rho \in \mathsf{Sym}([s]).\ N' = \rho \mathbin{!!} N \text{ and } \left(\forall\, k, l \in [s].\ c_{\rho(k)} < c_{\rho(l)} \iff c'_k < c'_l\right)$$

Note that all the comparator permutations ρ underlying R do not form a group under composition because they are not closed under composition. Figure 5.4 shows two permutations of comparator numbers ρ_a and ρ_b that both reorder parallel comparators in the left network to yield the middle networks. However the composition $\rho_\mathsf{ba} = \rho_\mathsf{b} \circ \rho_\mathsf{a}$ reorders non-parallel comparators because in the left network $N = c_0, \ldots, c_2$ the comparator c_0 precedes c_2 but in the right network $N^{\rho ab} = c_0^{\rho ab}, \ldots, c_2^{\rho ab}$ the comparator $c_{\rho_{ab}(0)}^{\rho ab} = c_2^{\rho ab}$ succeeds $c_{\rho_{ab}(2)}^{\rho ab} = c_1^{\rho ab}$.

Nonetheless R is an equivalence relation.

Proposition 15
The relation R is an equivalence relation.

Proof. We show that R is reflexive, symmetric and transitive.

We know that R is reflexive because id is in $\mathsf{Sym}([s])$ and $N = \mathsf{id} \,||\, N$ for all N in $\mathcal{N}_n^s$. Clearly $\prec_a$ for all channels a is preserved.

We know that R is symmetric because for every relation (N, N') of two networks in R by some permutation ρ in $Sym([s])$ there is an inverse permutation ρ^{-1} in $\mathsf{Sym}([s])$ and $\prec_a$ continues to be preserved for all channels a such that (N', N) is also in R.

We know that R is transitive because for any two relations (N, N') and (N, N'') in R by two permutations ρ and ρ' in $\mathsf{Sym}([s])$ there is also a relation (N, N'') in R given by the composition of the permutations $\rho' \circ \rho$. Recall that $\mathsf{Sym}([s])$ is a group and as such it is closed under composition. Additionally composition of the permutations retains the property that for all channels a the comparator order $\prec_a$ is preserved. □

Note that R is transitive, even though the permutations ρ underlying R do not form a group under composition. That is because in the former case a permutation of comparators is considered wrt a single network (the one on the right next to R) respectively, while for the latter case we consider not a single but all networks.

By Proposition 1.2 there is a group of network transformations. However there seems to be no action that intuitively represents this group as a group of comparator number permutations. In Section 7.1.2 we prove Lemma 27, which suggests that this group is better understood as the automorphism group of the underlying graph of a network.

Actually we can exchange $<$ in the definition of R with $<_i$, $\prec$ or $\prec_i$ respectively.

Proposition 16
Let the permutation of comparators ρ from $\mathsf{Sym}([s])$ on a network $N = c_0, c_1, \ldots, c_{s-1}$ in $\mathcal{N}_n^s$ yield the network $\rho \,||\, N = c_0^\rho, c_1^\rho, \ldots, c_{s-1}^\rho$. Then the following are equivalent

1. For all comparators numbers k and l in $[s]$ the comparator $c_{\rho(k)}$ precedes $c_{\rho(l)}$ on some channel iff c_k^ρ precedes c_l^ρ on some channel.
$$\forall\, k, l \in [s].\ c_{\rho(k)} < c_{\rho(l)} \iff c_k^\rho < c_l^\rho$$

2. For all comparators numbers k and l in $[s]$ and all channel numbers i in $[n]$ the comparator $c_{\rho(k)}$ precedes $c_{\rho(l)}$ on channel number i iff c_k^ρ precedes c_l^ρ on the channel with the same number i.
$$\forall\, k, l \in [s]\ \ i \in [n].\ c_{\rho(k)} <_i c_{\rho(l)} \iff c_k^\rho <_i c_l^\rho$$

3. For all comparators numbers k and l in $[s]$ the comparator $c_{\rho(k)}$ immediately precedes $c_{\rho(l)}$ on some channel iff c_k^ρ immediately precedes c_l^ρ on some channel.
$$\forall\, k, l \in [s].\ c_{\rho(k)} \prec c_{\rho(l)} \iff c_k^\rho \prec c_l^\rho$$

4. For all comparators numbers k and l in $[s]$ and all channel numbers i in $[n]$ the comparator $c_{\rho(k)}$ immediately precedes $c_{\rho(l)}$ on channel number i iff c^ρ_k immediately precedes c^ρ_l on the channel with the same number i.

$$\forall\, k, l \in [s]\ \ i \in [n].\ c_{\rho(k)} \prec_i c_{\rho(l)} \iff c^\rho_k \prec_i c^\rho_l$$

Proof. We have that $2 \implies 1$ and $4 \implies 3$ because if there is precedence on some channel, let it be channel number i, then by the premiss 2 or 4 respectively there is precedence between the corresponding comparators on the same channel i in the other network, i.e. there is precedence between the corresponding comparators on some channel in the other network.

Now we show the reverse direction that $1 \implies 2$ or $3 \implies 4$. If there is precedence on channel i we know by the premiss 1 or 3 respectively that there is precedence on some channel between the corresponding comparators in the other network. To show that 2 or 4 respectively is implied, we have to show that the precedence in the other network also occurs on channel number i (according to the premiss it may occur on some other channel instead). This is the case because ρ retains the channels of corresponding comparators, i.e. $c_{\rho(m)} = c^\rho_m$ for all m in $[s]$. Hence it holds that $c_{\rho(k)} \sqcap c_{\rho(l)} = c^\rho_k \sqcap c^\rho_l$ and thus $i \in c_{\rho(k)} \sqcap c_{\rho(l)}$ iff $i \in c^\rho_k \sqcap c^\rho_l$. This shows that $2 \iff 1$ and $4 \iff 3$.

Finally we show that $2 \iff 4$. For $2 \implies 4$ if there is immediate precedence on channel i in one network, w.l.o.g. we assume that network is N, that is $c_{\rho(k)} \prec_i c_{\rho(l)}$, then we show the corresponding immediate precedence on channel i in the other network, that is $c^\rho_k \prec_i c^\rho_l$. By the premiss 2 we only know that $c^\rho_k <_i c^\rho_l$. However this precedence is actually immediate because otherwise there is a channel number m in $[s]$ such that $c^\rho_k <_i c^\rho_m <_i c^\rho_l$. But by the premiss 2 this would yield $c_{\rho(k)} <_i c_{\rho(m)} <_i c_{\rho(l)}$, which contradicts $c_{\rho(k)} \prec_i c_{\rho(l)}$.

For $4 \impliedby 2$ if there is a precedence on channel i in one network, w.l.o.g. we assume that network is N, that is $c_{\rho(k)} <_i c_{\rho(l)}$, then we show the corresponding precedence on channel i in the other network, that is $c^\rho_k <_i c^\rho_l$. Note that there is a chain of immediate predecessors $c_{\rho(k)} \prec_i c_{\rho(m)} \prec_i \ldots \prec_i c_{\rho(l)}$ in N. By the premiss 4 there a chain of immediate predecessors in the other network, i.e. $c_{\rho(k)} \prec_i c_{\rho(m)} \prec_i \ldots \prec_i c_{\rho(l)}$ and thus we have $c_{\rho(k)} <_i c_{\rho(l)}$. □

Lemma 17
Permuting the channels in a network commutes with permuting the comparators.

$$\forall\, N \in \mathcal{N}^s_n\ \ \pi \in \mathsf{Sym}([n])\ \ \rho \in \mathsf{Sym}([s]) \colon \pi \diamond \rho \mathbin{||} N = \rho \mathbin{||} \pi \diamond N$$

Proof. Recall that a permutation π of the channel numbers π acts on comparators with the action $\bullet$.

We consider any network $N = c_0, c_1, \ldots, c_{s-1}$ from $\mathcal{N}_n^s$. A channel permutation π from $\mathsf{Sym}([s])$ acting on a network N yields the network $\pi \diamond N = c_0^\pi, \ldots, c_{s-1}^\pi$ such that $c_k^\pi = \pi \bullet (c_k)$ for any k in $[s]$. Likewise a comparator permutation ρ from $\mathsf{Sym}([s])$ yields the networks $\rho \mid\mid N = c_0^\rho, \ldots, c_{s-1}^\rho$ and $\rho \mid\mid (\pi \diamond N) = c_0^{\pi\rho}, \ldots, c_{s-1}^{\pi\rho}$ respectively such that $c_k^\rho = c_{\rho(k)}$ and $c_k^{\pi\rho} = c_{\rho(k)}^\pi$ for any k in $[s]$.

The comparator at position $\rho(k)$ in $\pi \diamond N$ is equal to the kth comparator in $\pi \diamond (\rho \mid\mid N)$ and $\rho \mid\mid (\pi \diamond N)$ respectively because $c_k^\rho = c_{\rho(k)}$ and $\pi \bullet (c_k) = c_k^\pi$ and $c_{\rho(k)}^\pi = c_k^{\pi\rho}$ for any k in $[s]$ respectively. In particular ρ maps any k in $[s]$ such that $\rho(k)$ is in $[s]$. Thus we have that $\pi \bullet (c_{\rho(k)}) = c_{\rho(k)}^\pi$.

$$\pi \bullet (c_k^\rho) = \pi \bullet (c_{\rho(k)}) = c_{\rho(k)}^\pi = c_k^{\pi\rho}$$

The commutativity of the actions $\diamond$ and $\mid\mid$ under composition follows.

$$\begin{aligned}
\pi \diamond (\rho \mid\mid N) &= \pi \diamond (c_0^\rho, c_1^\rho, \ldots, c_{s-1}^\rho) \\
&= \pi \bullet (c_0^\rho), \pi \bullet (c_1^\rho), \ldots, \pi \bullet (c_{s-1}^\rho) \\
&= \pi \bullet (c_{\rho(0)}), \pi \bullet (c_{\rho(1)}), \ldots, \pi \bullet (c_{\rho(s-1)}) \\
&= c_{\rho(0)}^\pi, c_{\rho(1)}^\pi, \ldots, c_{\rho(s-1)}^\pi \\
&= \rho \mid\mid (c_0^\pi, c_1^\pi, \ldots, c_{s-1}^\pi) \\
&= \rho \mid\mid (\pi \diamond N)
\end{aligned}$$

□

Composing R with $P\,;O$ or $(O\,;I)^*$ or P or O or I yields an equivalence relation in each case. This is captured by the following proposition.

Proposition 18
The composition of R with any equivalence relation $T \subseteq P\,;O$ yields an equivalence relation $R\,;T$.

Proof. This follows from Lemma 17 and Proposition 5. □

6 Breaking Symmetries

In this section we consider attempts to break the symmetry $R\,;P\,;O$ in the state of the art encoding. First we consider how to break any symmetry by decomposition in Section 6.1. Then we break the permutation of parallel comparators R in Section 6.2. In particular we distinguish between different representations of comparator networks that may or may not model the order within layers or layers at all. Thereafter in Section 6.3 we turn to the other component $P\,;O$, that is permuting channels and twisting outputs. Specicifically we first identify the problem of breaking $P\,;O$ by decomposition. In Section 6.3.1 we apply our insights on input twists from Section 5.2 to resolve the problem for certain cases. In particular we show how to break $P\,;O$ in such cases. We show that this is compatible with the encoded model of comparator networks in Section 6.3.2. Finally we give an alternative encoding of comparator networks that resolves the problem for all cases and discuss its practicality in Section 6.4.

6.1 Breaking Symmetries by Decomposition

A way to break the composition of two equivalence relations $T\,;U \subseteq S^2$ is to break its decomposition T and U in a compatible way.

Let $[x]_T$ be the equivalence class that contains all complete assignments y that are T-symmetric to the complete assignment x in S, that is $x\ T\ y$. Let φ be a complete and consistent symmetry breaking constraint for T such that exactly one representative in each equivalence class satisfies φ. Then $\mathrm{repr}^T_\varphi(x)$ denotes that representative assignment in $[x]_T$.

Let φ be a complete symmetry breaking constraint for T as mentioned and let ψ be a complete symmetry breaking constraint for S. Further let $S/(T\,;U)$ be the set of all equivalence classes on S under the equivalence $T\,;U$. Then φ and ψ are compatible if there is exactly one x in each equivalence class C from $S/(T\,;U)$ that is the representative in both its equivalence class $[x]_T$ and $[x]_U$ respectively, that is

$$\forall\, C \in S/(T\,;U) \quad \exists!\, x \in C. \quad x = \mathrm{repr}^T_\varphi(x) = \mathrm{repr}^U_\psi(x).$$

Note that any complete assignment y satisfies φ iff $y = \mathrm{repr}^T_\varphi(y)$. And y satisfies ψ iff $y = \mathrm{repr}^U_\psi(y)$. Hence y satisfies $\varphi \wedge \psi$ iff $y = \mathrm{repr}^T_\varphi(y)$ and $y = \mathrm{repr}^U_\psi(y)$. Thus this

compatibility condition corresponds to $\varphi \wedge \psi$ being a consistent and complete symmetry breaking constraint for $T\,;U$ where that one assignment x is $\mathrm{repr}^{T;U}_{\varphi\wedge\psi}(x)$ of each equivalence class C under $T\,;U$ respectively. If there is no such assignment x, then $\varphi \wedge \psi$ is an inconsistent symmetry breaking constraint for $T\,;U$. If there are more than one of such assignments, then it is a consistent but incomplete symmetry breaking constraint for $T\,;U$.

In the rest of this thesis whenever we break a composition $T\,;U$ by breaking its decomposition with φ and ψ we enforce the following condition that is sufficient for compatibility:

Proposition 19
The symmetry breaking constraints φ and ψ are compatible if the mappings to the representative under the respective constraint commute under composition, i.e.

$$\mathrm{repr}^T_\varphi \circ \mathrm{repr}^U_\psi = \mathrm{repr}^U_\psi \circ \mathrm{repr}^T_\varphi\,.$$

Proof. Let the mappings commute under composition. We show that then φ and ψ are compatible. First we show there is at least one assignment in each equivalence class C under $T\,;U$ that satisfies $\varphi \wedge \psi$. In a second step we show that there is at most one such assignment in each class C.

Mapping every assignment in C by $\mathrm{repr}^T_\varphi \circ \mathrm{repr}^U_\psi$ gives us some assignment x respectively. By definition x is the one representative of its equivalence under T that satisfies φ. Mapping every assignment by $\mathrm{repr}^U_\psi \circ \mathrm{repr}^T_\varphi$ instead gives us the same assignment x respectively because repr^T_φ and repr^U_ψ commute under composition. Thus x likewise satisfies ψ. Note that the mappings preserve the equivalence class of the assignment. Since each equivalence class C contains at least one element, mapping every element in C yields at least one such assignment x that satisfies $\varphi \wedge \psi$.

Finally we prove there is at most one assignment in C under $T\,;U$ satisfying the constraints $\varphi \wedge \psi$. For the sake of contradiction we assume there are two different assignments x and y in the same equivalence class under $(T\,;U)$ with $x = \mathrm{repr}^T_\varphi(x) = \mathrm{repr}^U_\psi(x)$ and $y = \mathrm{repr}^T_\varphi(y) = \mathrm{repr}^U_\psi(y)$ such that φ and ψ are incompatible. By definition of $T\,;U$ there is a z such that $x\ T\ z$ and $z\ U\ y$. We have $\mathrm{repr}^T_\varphi(z) = x$ because $x\,T\,z$ and $x = \mathrm{repr}^T_\varphi(x)$. Analogously we have $\mathrm{repr}^U_\psi(z) = y$. Utilizing that repr^T_φ and repr^U_ψ commute under composition and both map x to x and y to y we arrive at the contradiction

$$\begin{aligned} x &= \mathrm{repr}^T_\varphi(z) \\ &= \mathrm{repr}^U_\psi(\mathrm{repr}^T_\varphi(z)) \\ &= \mathrm{repr}^T_\varphi(\mathrm{repr}^U_\psi(z)) \\ &= \mathrm{repr}^U_\psi(z) \\ &= y. \end{aligned}$$

□

There is another condition that is sufficient for compatibility: Breaking the symmetry T with φ preserves the symmetry component U.

$$\forall\, x, y \in S. \quad x\ (T\,;U)\ y \implies \mathrm{repr}^T_\varphi(x)\ U\ \mathrm{repr}^T_\varphi(y)$$

Note that $x\ (T\,;U)\ y$ does not imply $x\ U\ y$. Instead it means that U is a component of the relation. This component is preserved.

Consider any equivalence class C from $S/(T\,;U)$. We assume that there are two different complete assignments x and y in C that satisfy φ. We have $x\ (T\,;U)\ y$ since x and y are both in C. If breaking T with φ preserves U, then $x\ U\ y$ because x and y being satisfiable corresponds to them being the representative of their equivalence class. In particular all assignments in C are in the same equivalence class C' under U. Further note that C' is a subset of C. For that we recall that U is finer than $T\,;U$ and thus every equivalence class under U is fully contained in one class in $T\,;U$, which in this case is C. Hence all assignments in C satisfying φ make up the equivalence class C'. We observe that adding ψ to the constraints breaks C' in a way that is compatible with φ.

Likewise we can alternative ensure that breaking U with ψ preserves the symmetry component T, i.e.

$$\forall\, x, y \in S. \quad x\ (T\,;U)\ y \implies \mathrm{repr}^U_\psi(x)\ T\ \mathrm{repr}^U_\psi(y).$$

Note that both breaking T preserving U and breaking U preserving T are sufficient conditions for compatibility respectively but we have shown neither to be a necessary condition.

Further note that if φ and/or ψ are partial symmetry breaking constraints for T and/or U respectively, then the sufficient conditions for compatibility at least ensure consistency.

6.2 Breaking the Permutation of Parallel Comparators

In this section we tackle breaking the composition $R\,;P\,;O$ by breaking R in a way that is compatible with the attempt to break $P\,;O$ in the next section.

First we observe that the state of the art encoding differs from the model of sorting networks used in the definition of $R\,;P\,;O$ insofar as it encodes sequences of unordered layers instead of sequences of comparators. Recall that a sequence of ordered layers corresponds to exactly one sequence of comparators and exactly one sequence of unordered layers. However there may be different partitions of the same comparator sequence into layers.

Instead of the set of comparator gate sequences ${}^{\mathrm{g}}\mathcal{N}^s_n$ with n channels, size s we now consider a different set of networks, that is ${}^{\mathrm{ol}}\mathcal{N}^s_n$ the set of sequences of ordered layers

with the same dimensions n and s. There is a symmetry $K \subset {}^{\mathrm{ol}}\mathcal{N}_n^{s\,2}$ by that two sequences of ordered layers are symmetric if they correspond to the same sequence of ordered comparator gates in ${}^{\mathrm{g}}\mathcal{N}_n^s$.

Similarly we can consider the set of sequences of unordered layers ${}^{\mathrm{ul}}\mathcal{N}_n^s$. Then there is a symmetry $L \subset {}^{\mathrm{ol}}\mathcal{N}_n^{s\,2}$ by that two sequences of ordered layers are symmetric if they correspond to the same sequence of unordered layers in ${}^{\mathrm{ul}}\mathcal{N}_n^s$.

Let $\mathsf{g} : {}^{\mathrm{ol}}\mathcal{N}_n^s \to {}^{\mathrm{g}}\mathcal{N}_n^s$ and $\mathsf{ul} : {}^{\mathrm{ol}}\mathcal{N}_n^s \to {}^{\mathrm{ul}}\mathcal{N}_n^s$ give that one corresponding sequence in ${}^{\mathrm{g}}\mathcal{N}_n^s$ or ${}^{\mathrm{ul}}\mathcal{N}_n^s$ respectively, then we have defined that

$$\begin{aligned} N\ K\ N' &\iff \mathsf{g}(N) = \mathsf{g}(N') \\ N\ L\ N' &\iff \mathsf{ul}(N) = \mathsf{ul}(N'). \end{aligned}$$

We can represent R (or P, O and I respectively) as a relation R' on ${}^{\mathrm{ol}}\mathcal{N}_n^s$ the same way, i.e.

$$\begin{aligned} N\ R'\ N' &\iff \mathsf{g}(N)\ R\ \mathsf{g}(N') \\ N\ P'\ N' &\iff \mathsf{g}(N)\ P\ \mathsf{g}(N') \\ N\ O'\ N' &\iff \mathsf{g}(N)\ O\ \mathsf{g}(N') \\ N\ I'\ N' &\iff \mathsf{g}(N)\ I\ \mathsf{g}(N') \\ N\ O'_k\ N' &\iff \mathsf{g}(N)\ O_k\ \mathsf{g}(N') \\ N\ I'_k\ N' &\iff \mathsf{g}(N)\ I_k\ \mathsf{g}(N'). \end{aligned}$$

Note that K, L and R' are closely related. The symmetry K relates different partitions of the same comparator sequence into layers, i.e. K relates networks where the same comparator c may be put into either one or the other layer because all comparators in both candidate layers are independent/parallel to c. Section 2.1 gives an example of this case in Figure 2.1. The symmetry L relates networks where the comparators are reordered within each layer. Obviously comparators within the same layer are parallel. So while K relates different partitions into layers and L admits permutations within parallel subsequences, the symmetry R' relates different partitions of two sequences where the sequences are equivalent modulo permutations of parallel comparators. Thus we have that $R' = K\,;L^*$. We only need to break K because by our choice to encode networks as sequences of unordered layers we break L with $\mathrm{repr} = \mathsf{ul}$.

We show below that the mapping to the our chosen representative of K commutes with L to prove that our break of K is compatible with this symmetry break by remodeling of L. Though this commutativity is actually a pseudo-commutativity because there are two variants of the mapping to the representative of the equivalence class under K, i.e. one variant on ${}^{\mathrm{ol}}\mathcal{N}_n^s$ and one variant on ${}^{\mathrm{ul}}\mathcal{N}_n^s$.

More precisely we break the symmetry K' on ${}^{\mathrm{ul}}\mathcal{N}_n^s$. Let the fiber of a network N from ${}^{\mathrm{ul}}\mathcal{N}_n^s$ under ul be the set of all networks M in ${}^{\mathrm{ol}}\mathcal{N}_n^s$ for that $N = \mathsf{ul}(M)$. We denote that

fiber by $\mathsf{ul}^{-1}(N)$. The symmetry K relates different partitions of the same comparator sequence into layers. Let K relate the networks N and N'. Mapping such networks by ul forgets the order within the layers but retains the partition into layers such that the following two definitions of K' are equivalent:

$$\begin{aligned} N\ K'\ N' &\iff \exists\, M \in \mathsf{ul}^{-1}(N) \quad M' \in \mathsf{ul}^{-1}(N') \colon M\ K\ M' \\ &\iff \forall\, M \in \mathsf{ul}^{-1}(N) \quad M' \in \mathsf{ul}^{-1}(N') \colon M\ (K\,;L)\ M'. \end{aligned}$$

The second definition shows that breaking L by remodeling preserves the component K in the composition $K\,;L$ as K'.

We can extend this by $P'\,;O'$ such that any symmetry break (that includes remodeling via ul) of $L\,;P'\,;O'$ preserves the component K in the composition $K\,;L\,;P'\,;O'$ as K'. To make this explicit let $\mathrm{repr} : {}^{\mathrm{ol}}\mathcal{N}_n^s \to {}^{\mathrm{ul}}\mathcal{N}_n^s$ give us the representative network of each equivalence class under $L\,;P'\,;O'$. I.e. for this extension ul is generalized to repr. As both P' and O' retain the partition into layers like ul does, breaking them preserves the component K:

$$N\ K'\ N' \iff \forall\, M \in \mathrm{repr}^{-1}(N) \quad M' \in \mathrm{repr}^{-1}(N') \colon M\ (K\,;L\,;P'\,;O')\ M'.$$

Because K' like K is a symmetry between different partitions of the same comparator sequence into layers, we can break K' by considering only one representative partition. For that we add the following symmetry breaking constraint

$$\bigwedge_{k\in[d-1]} \bigwedge_{\substack{i,j\in[n]\\ i<j}} \neg u_i^k \vee \neg u_j^k \vee \neg g_{i,j}^{k+1}.$$

Say we consider setting $g_{i,j}^{k+1}$ to true. Then at least one of the channels i and j in the previous layer k has to be used by a comparator too. Otherwise we violate the constraint. If neither i or j is used in layer k we have to assign false to $g_{i,j}^{k+1}$, but then we assign $g_{i,j}^{k}$ to true instead. Thus the constraint moves every comparator to the earliest layer possible.

There is a similar constraint that moves every comparator to the latest layer possible.

$$\bigwedge_{k\in[d-1]} \bigwedge_{\substack{i,j\in[n]\\ i<j}} \neg u_i^{k+1} \vee \neg u_j^{k+1} \vee \neg g_{i,j}^{k}.$$

But its interaction with constraints that we add in the next section is more complicated. Thus we decide on moving every comparator to the earliest layer possible.

Finally we observe that moving every comparator to the earliest or latest layer repectively commutes with losing the order within each layer. This is the pseudo-commutitativity of the mapping to the representative of the equivalence class under K with ul.

In the next section we attempt to break $P'\,;O'$ in a way that is compatible with ul.

6.3 Breaking the Permutation and Untangling of Channels

In this section we attempt to break $P\,;O$. First we show two similar statements for networks that are a sequence of comparators.

1. O_k never relates two standard sorting networks

2. P never relates two standard sorting networks

Let $N = c_0, \ldots, c_{s-1}$ be a standard network. For Statement 1 twisting the outputs of any comparator c_k in N yields the network $\mathsf{t}_{\triangleq k}N$. We observe that $\mathsf{t}_{\triangleq k}N$ is nonstandard, because the output twist action $\triangleq_k$ changes the min-max comparator c_k to a max-min comparator.

Additionally for Statement 2 let N be sorting. In a standard sorting network there is at least one comparator $(i, i+1)$ in the network for every i in $[n-1]$, otherwise the input 0^i101^{n-i-2} is not sorted. Permuting the channels of N by any permutation π in $\mathsf{Sym}([n])$ yields a network $\pi \diamond N$. We observe that unless π is the identity, the network $\pi \diamond N$ is nonstandard, because there is at least one pair of adjacent channels i and $i+1$ with $\pi(i) > \pi(i+1)$ such that the min-max comparator $(i, i+1)$ changes to the max-min comparator $(\pi(i), \pi(i+1))$.

We conclude that the restriction to standard networks completely breaks P and O respectively. This is because each of the components P, O_0, O_1, ... relates every standard network to an irrepresentable nonstandard network. However since there are many standard networks related by $P\,;O$, restricting the models to standard networks breaks P and O in an incompatible way. We know that the restriction is a consistent symmetry break because every generalized π-sorting network can be untangled to a standard sorting network. The problem is that the restriction to standard sorting networks does not completely break the symmetry $P\,;O$.

This result equally holds for networks as sequences of comparators, ordered layers or unordered layers. In the previous section we have accounted for the difference between sequences of comparator and sequences of ordered layers by breaking K and showing that it preserves $P'\,;O'$. For the following attempt at breaking the variant of $P'\,;O'$ on ${}^{\mathsf{ul}}\mathcal{N}^s_n$ we pay attention to compatibility with ul in Section 6.3.2.

6.3.1 Breaking Input Twists

We turn to I. Recall the Propositions 13 and 14 state that I is finer than $P\,;O$ and while both are correct under permuted outputs only I is also strongly correct. In particular

they state that I is strongly complete relative to $P\,;O$. That is if any two generalized π-sorting networks are related by $P\,;O$, then they are also related by I. Unlike I the symmetry $P\,;O$ relates π- and π'-sorting networks where $\pi \neq \pi'$. While I is finer, it is a suitable stand-in for $P\,;O$ on the domain restricted to standard networks where any generalized sorting network is necessarily an id-sorting network.

Can we hope to break I by breaking each I_k for all k in a standard network encoding? We want to add a symmetry breaking constraint that is only satisfied by exactly one of the (up to) two assignments that model the symmetric networks where the input channels of the kth comparator are twisted. The idea is that we only add the symmetry breaking constraint if I_k relates two standard networks.

Proposition 20
Twisting the inputs of the kth comparator $c_k = (i, j)$ in a standard network N yields the generalized network $\mathtt{t}\triangledown_k N$. The network $\mathtt{t}\triangledown_k N$ is nonstandard iff there is a comparator c_l in N with $l < k$ that compares either i or j with some channel h where $i \leq h \leq j$.

Proof. We show that if there is no such comparator c_l, then $\mathtt{t}\triangledown_k N$ is standard. And we show that if there is such a comparator c_l, then $\mathtt{t}\triangledown_k N$ is nonstandard.

It suffices to consider every comparator c in N, determine whether it qualifies as c_l and show that ζ_k changes it to a max-min comparator or that it remains min-max accordingly.

If c does not precede c_k,
then c does not qualify as c_l. It is also not in the prefix network modified by the action $\triangledown_k$ and remains min-max.

If otherwise c precedes, but does not share a channel with c_k,
then c does not qualify as c_l. It is unaffected by the action $\triangledown_k$ and remains min-max.

If otherwise c precedes and shares exactly one channel with c_k,
then we assume that channel is i without loss of generality. It qualifies as c_l and is either of the form (h, i) with $h < i$ or (i, h) with $i < h$. Without loss of generality we assume the former. The twist action $\triangledown_k$ changes c to the max-min comparator (i, h).

If otherwise c precedes and shares both channels with c_k, that is $c = (i, j)$,
then c qualifies as c_l and the action $\triangledown_k$ changes it to the max-min comparator (j, i).

□

This proposition applies for the variant of $P\,;O$ on ${}^{\mathrm{ul}}\mathcal{N}_n^s$ if we take k and l to denote

the layer number instead of the comparator number, i.e. k and l denote a position in the sequence of unordered layers instead of a position in the sequence of comparators.

In general we do not statically know for each comparator in any layer k whether there is a comparator in layer l with $l < k$ that compares either i or j with some channel h where $i \leq h \leq j$. We do know statically that this is not the case if layer k is the first layer, but input twists of comparators in the first layer yield the same network. Thus we add symmetry breaking constraint clauses for every comparator in every layer k with an additional premiss part. The premiss part ensures the symmetry breaking constraint applies only under the premiss of nonexistence of such a comparator in layer l. The premiss part of the clause is satisfied if the twist yields a nonstandard network. Then the symmetry breaking constraint part of the clause does not apply because the clause is already satisfied by a literal in the premiss part. This way we can dynamically break the symmetry depending on comparator placements in each layer l before layer k.

To encode the premiss part we introduce another type of variable $\text{toBetweenBefore}_{i,j}^{k}$ that indicates that at least one of the following two conditions is satisfied: Either there is a comparator $g_{h,j}^{l}$ for some channel h with $i \leq h < j$ in some layer l with $l < k$. Or analogously there is a comparator $g_{i,h}^{l}$ for some channel h with $i < h \leq j$ in some layer l with $l < k$.

The naming of the variable draws on variable names from Ehlers and Müller [42, 12]. As mentioned in Section 4.3.5 the state of the art approach introduces additional redundant clauses and variables to facilitate propagation. In particular Ehlers and Müller introduce the variables $\text{oneDown}_{i,j}^{k}$ and $\text{oneUp}_{i,j}^{k}$ that indicate that there is a comparator $g_{h,j}^{k}$ for some $i \leq h < j$ or $g_{i,h}^{k}$ for some $i < h \leq j$ respectively.

We ensure that $\text{toBetweenBefore}_{i,j}^{k}$ is set appropriately by induction on the layer. For the induction basis, i.e. $k = 0$, we already observed that there is no comparator gate before layer k because k is the first layer. For the induction step we consider $\text{toBetweenBefore}_{i,j}^{k+1}$. There is a gate before layer $k+1$ comparing the value of either channel i or j to the value on channel h with $i \leq h \leq j$ if there is such a comparator either before layer k or in layer k. The former case is indicated by $\text{toBetweenBefore}_{i,j}^{k}$. The latter case is indicated by $\text{oneDown}_{i,j}^{k}$ for a comparator on j and likewise it is indicated by $\text{oneUp}_{i,j}^{k}$ for a comparator on i. If a comparator compares i with j both $\text{oneDown}_{i,j}^{k}$ and $\text{oneUp}_{i,j}^{k}$ are set. Thus it suffices to ensure that $\text{toBetweenBefore}_{i,j}^{k+1}$ is assigned the maximum of

toBetweenBefore$^k_{i,j}$, oneUp$^k_{i,j}$ and oneDown$^k_{i,j}$.

$$\bigwedge_{\substack{i,j\in[n]\\ i<j}} \neg\,\text{toBetweenBefore}^0_{i,j}$$

$$\bigwedge_{k\in[d-1]} \bigwedge_{\substack{i,j\in[n]\\ i<j}} \max(\text{toBetweenBefore}^{k+1}_{i,j}, \{\text{toBetweenBefore}^k_{i,j}, \text{oneDown}^k_{i,j}, \text{oneUp}^k_{i,j}\})$$

Now the premiss part added to a clause that breaks an input twist of a standard comparator between the channels i and j in layer k is $\neg g^k_{i,j} \vee \text{toBetweenBefore}^k_{i,j}$. Note that the clause is satisfied by the premiss part if there is no such comparator $g^k_{i,j}$ or if the input twist of $g^k_{i,j}$ yields a nonstandard network. In both cases the symmetry breaking constraint that makes up the rest of the clause does not affect the set of solutions.

Now we derive symmetry breaking constraints for the variant of I' on ${}^{\text{ul}}\mathcal{N}^s_n$ by considering I'. Say we break I'_x, that is a twist of the comparator (i, j) in layer k where that comparator is the xth comparator in the network. We break the symmetry I'_x by adding a constraint that is satisfied by exactly one of the symmetric networks. In search for such a constraint we recognize that at least one of the channels i and j is used in the preceding layer $k-1$ because every comparator is moved to the earliest layer possible in Section 6.2. If either channel j or channel i is not used in layer $k-1$ we can twist the input channels such that the other channel is not used instead. Thus we add a constraint such that channel i is always used.

$$\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{i,j\in[n]\\ i<j}} \neg g^k_{i,j} \vee \text{toBetweenBefore}^k_{i,j} \vee \neg u^{k-1}_i$$

It remains to break the case where i and j are both used in layer $k-1$. First we consider a naive and ultimately inconsistent symmetry breaking constraint to highlight a difficulty. The twist of (i, j) in layer k yields a nonstandard network if i is compared with j in the layer $k-1$. As this case is irrelevant, we assume that two different comparators use i and j in layer $k-1$. Let i be compared with o in layer $k-1$ and let j be compared with p in layer $k-1$. We observe that $o \neq p$ because all comparisons in the same layer are independent. Now the idea is to twist the input channels i and j such that $o < p$.

For an example we consider the four symmetric networks in Figure 6.1. Specifically the networks are either symmetric by twisting the inputs of comparator x or y. Related networks are connected by an arrow labeled with the symmetry, i.e. I'_x or I'_y such that the arrow points to the representative network that satisfies $o < p$. The Table 6.1 shows the concrete values for i, j, o and p that determine the representative of each equivalence class. There are four equivalence classes that contain the two top, right, bottom and left networks respectively.

In fact Figure 6.1 shows a counterexample. We see that breaking I'_x does not preserve the component I'_y in the composition $I'_x ; I'_y$ and vice versa. Not only do we fail to satisfy the sufficient conditions for a consistent and complete symmetry break, Figure 6.1 actually shows that this approach is inconsistent. This is because no network in the depicted equivalence class under $I'_x ; I'_y$ is the representative in both its equivalence classes under I'_x and I'_y respectively.

We fix this by enforcing at least one of the conditions that are sufficient for compatibility. For example to give an intuition we may try to make the symmetry breaks (the arrows in Figure 6.1) commute. For that we have to come up with a different constraint such that for all x and y in $[s]$ we have

$$\mathrm{repr}^{I'_x} \circ \mathrm{repr}^{I'_y} = \mathrm{repr}^{I'_x} \circ \mathrm{repr}^{I'_y}.$$

Considering our example I'_x is broken for each equivalence class under I'_x. However breaking I'_x does not preserve I'_y, i.e. the representatives of each equivalence class under I'_x are in different equivalence classes under I'_y that is being broken. The problem is that the criteria for breaking the symmetry I'_x, e.g. o and p may vary depending on the equivalence class under I'_x. The networks in different equivalence classes under I'_x differ in the way the channels of comparator y are used in the layer below, that is layer $k-1$. In our example one of those channels is connected to either i or j such that it is considered for the symmetry break of I'_x as either o or p. Concretely for the top networks in the same equivalence class under I'_x the variables o and p have the values 3 and 4, whereas for the bottom networks in the complementary equivalence class the value 3 is replaced by 5 due to the input twist of comparator y. In the first case 4 is the greater value but in the second case 4 is the smaller value such that the representative of the top networks connects 4 to the first component of x, i.e. channel 2, in layer $k-1$ and the representative of the bottom networks connects 4 to the second component of x, that is channel 0. Since networks in different equivalence classes under I'_y differ in how the channels of comparator x are connected, the representatives of the equivalence classes under I'_x are in different equivalence classes under I'_y.

We fix this by choosing different symmetry breaking criteria that are independent from other twists in the same layer k. First we realize that if o and p are both not used in k, then they are independent from other twists in k. Otherwise o may be compared with some channel q in layer k and p may be compared with some channel r in layer k respectively. Instead of twisting such that $o < p$ we may then substitute $\min(q, o)$ for o and/or $\min(p, r)$ for p. For example if both i and j are used, we twist such that $\min(o, q) < \min(p, r)$. Thus our symmetry breaking criteria depend on the comparators in layer k but not on the comparator placements in layer $k-1$ that are affected by input twists of comparators in k. Figure 6.2 and Table 6.2 show the fixed example.

Note that if y is in an earlier layer than x, then the representatives of the equivalence classes under I'_x are always in the same equivalence class under I'_y. That is because

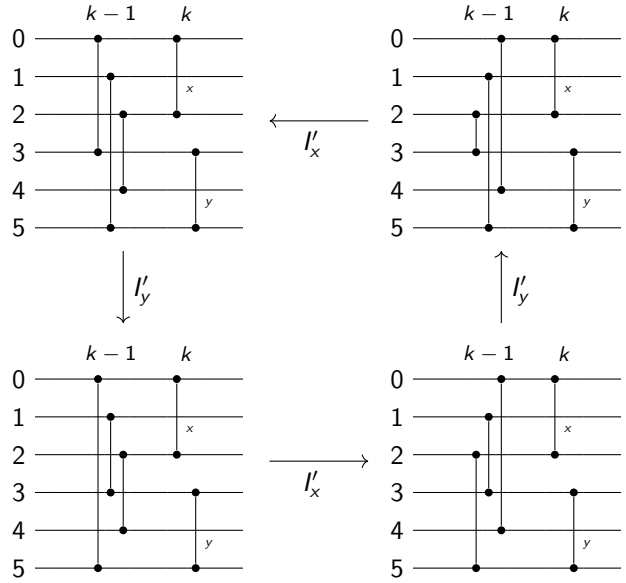

Figure 6.1: *Four networks symmetric under* l'_x ; l'_y.

	i	j	o	p	repr
top left	0	2	3	4	✓
top right	0	2	4	3	
bottom left	0	2	5	4	
bottom right	0	2	4	5	✓
top left	3	5	0	1	✓
bottom left	3	5	1	0	
top right	3	5	2	1	
bottom right	3	5	1	2	✓

Table 6.1: *The representatives of each equivalence class from Figure 6.1.*

breaking symmetric twists of x does not depend on how y is broken.

To come up with a terse encoding we consider the positions at channel o and p in layer k respectively. In particular we consider the cases where $o > p$. We introduce yet another auxiliary variable $\text{viaWrongTwist}^k_{p,o}$ that is guaranteed to be set whenever there are some channels i and j different from o and r with $i < j$ such that i and j are compared in layer k and there are two gates in layer $k-1$ that compare i with o and j with p respectively. The variable name hints at the fact that the positions o and p in layer k are connected via the three comparator gates in the layers k and $k-1$. In particular the connection between o and p via $g^k_{i,j}$ is wrong in the sense that it broken by our naive symmetry breaking constraint because $o > p$. In contrast there is a right twist with o and p swapped and the inputs of $g^k_{i,j}$ twisted such that the gates in layer $k-1$ still compare i with o and j with p.

The following constraint ensures that $\text{viaWrongTwist}^k_{p,o}$ is set if o and p are connected via some $g^k_{i,j}$ with its inputs twisted in the wrong way.

$$\bigwedge_{\substack{k\in[d]\\k>0}} \bigwedge_{\substack{o,p\in[n]\\o\neq p}} \bigwedge_{\substack{i,j\in[n]\setminus\{o,p\}\\i<j}} \neg g^k_{i,j} \vee \neg\,\text{standGate}^{k-1}_{i,o} \vee \neg\,\text{standGate}^{k-1}_{j,p} \vee \text{toBetweenBefore}^k_{i,j} \vee \text{viaWrongTwist}^k_{p,o}$$

Note that this constraint does not imply the reverse, i.e. that $\text{viaWrongTwist}^k_{p,o}$ is set *only* if p and o are connected via some $g^k_{i,j}$ with its inputs twisted in the wrong way.

Further note that we require $o \neq p$ instead of $o > p$ such that for every $\text{viaWrongTwist}^k_{p,o}$ that is disallowed in the naive encoding, there is $\text{viaWrongTwist}^k_{o,p}$, which is symmetric under I'. Thus we can selectively break one or the other case.

Now we again consider that o and p are not used in k. We disallow wrong twists by adding a symmetry constraint that is not satisfied in this case.

$$\bigwedge_{\substack{k\in[d]\\k>0}} \bigwedge_{\substack{o,p\in[n]\\o>p}} \neg\,\text{viaWrongTwist}^k_{p,o} \vee \neg u^k_o \vee \neg u^k_p$$

Similarly we disallow wrong twists if only o is compared with q in layer k or only p is compared with r in k respectively

$$\bigwedge_{\substack{k\in[d]\\k>0}} \bigwedge_{\substack{o,p\in[n]\\o\neq p}} \bigwedge_{\substack{q\in[n]\setminus\{o,p\}\\\min(q,o)>p}} \neg\,\text{viaWrongTwist}^k_{p,o} \vee \neg\,\text{standGate}^k_{o,q} \vee \neg u^k_p$$

$$\bigwedge_{\substack{k\in[d]\\k>0}} \bigwedge_{\substack{o,p\in[n]\\o\neq p}} \bigwedge_{\substack{r\in[n]\setminus\{o,p\}\\o>\min(r,p)}} \neg\,\text{viaWrongTwist}^k_{p,o} \vee \neg u^k_o \vee \neg\,\text{standGate}^k_{p,r}$$

And finally we disallow wrong twists if both o and p are compared in k. For that we observe that $q = p$ implies $r = o$ and vice versa $r = o$ implies $q = p$ in a valid network. However in this case $\min(q, o) = \min(r, p)$ such that we do not break. Thus we omit these cases.

$$\bigwedge_{\substack{k\in[d]\\k>0}} \bigwedge_{\substack{o,p\in[n]\\o\neq p}} \bigwedge_{\substack{q,r\in[n]\setminus\{o,p\}\\\min(q,o)>\min(r,p)}} \neg\,\text{viaWrongTwist}^k_{p,o} \vee \neg\,\text{standGate}^k_{o,q} \vee \neg\,\text{standGate}^k_{p,r}$$

Note that introducing the auxiliary variable viaWrongTwist achieved a terser encoding because otherwise we would not only quantify over k, o, p, q and r but also over i and j. Keep in mind the size and number of clauses added to break I' scale polynomially with n, while the size and number of clauses for the counterexamples scale exponentially. Thus with growing n the share that the symmetry breaking constraint makes up of the whole encoding decreases.

6.3.2 Breaking Input Twists in Unordered Layers

Finally we consider how the current encoding φ with the added symmetry breaking constraint for the input twists interacts with ul.

As ul discards the order of comparators within each layer, it maps different networks in ${}^{\text{ol}}\mathcal{N}^s_n$ to the same network in ${}^{\text{ul}}\mathcal{N}^s_n$. For φ to be consistent, we need to ensure that representatives in ${}^{\text{ol}}\mathcal{N}^s_n$ are not mapped to the same network N in ${}^{\text{ul}}\mathcal{N}^s_n$ as networks in

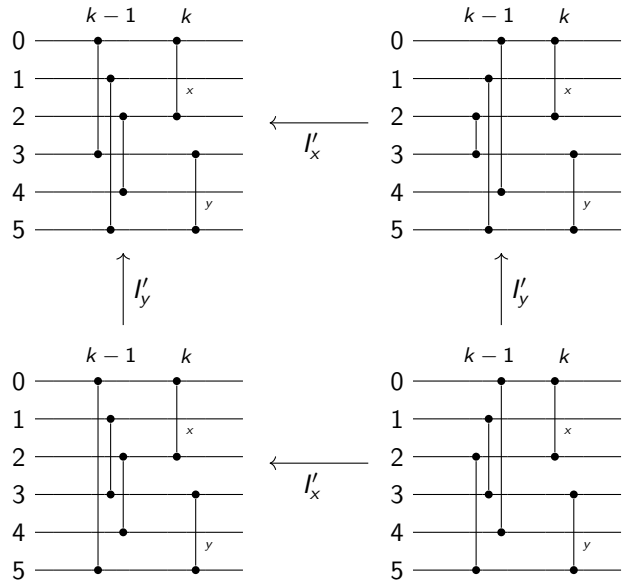

Figure 6.2: *Four networks symmetric under I'_x ; I'_y.*

	i	j	o	p	q	r	repr
top left	0	2	3	4	5		✓
top right	0	2	4	3		5	
bottom left	0	2	5	4	3		✓
bottom right	0	2	4	5		3	
top left	3	5	0	1	2		✓
bottom left	3	5	1	0		2	
top right	3	5	2	1	0		✓
bottom right	3	5	1	2		0	

Table 6.2: *The representatives of each equivalence class from Figure 6.1.*

${}^{\text{ol}}\mathcal{N}_n^s$ that are not representatives such that N would be disallowed by φ despite being a representative. We can formulate this differently as a Proposition.

Proposition 21
For every two networks M and M' in ${}^{\text{ol}}\mathcal{N}_n^s$ that are symmetric under L it holds that M is a representative of its equivalence class under I' iff M' is.

Proof. For the sake of contradiction we assume there are two networks M and M' with $M\ L\ M'$ where M is a representative but M' is not. Since M' is not a representative of its equivalence class under I' it is twisted the wrong way at at least one comparator. Let S be the subset of the comparators (i, j) in M' for that φ is violated. We consider the subset S_k of S that contains the comparators in layer k where k is the latest layer containing comparators in S. We consider the input twist of some comparatator (i, j) in S_k.

We know that for M every comparator is moved to the earliest layer possible such that at least one of i or j is used in $k-1$. Actually we know that i is used because M is a the representative. Due to $M\ L\ M'$ we know that i is used in M' too. Furthermore we know that j is used in layer $k-1$ in both networks too because otherwise the twist yields a network where i is not used in $k-1$. This shows the existence of o and p.

Consider that $o = j$ and $p = i$. This is not possible because this twists the comparator in $k-1$ from min-max to max-min or vice versa, which contradicts $M\ L\ M'$. We observe

that twisting swaps o and p such that we contradict $M\ L\ M'$. However this can be fixed by twisting other gates that compare o and p in layers after $k-1$. One obvious candidate would be a gate comparing o and p in layer k. However this corresponds to the case that the comparator that compares o with q is the very comparator that compares p with r, which is not broken by φ. This contradicts the fact that the comparator (i,j) that we consider does by definition violate φ.

If the gate comparing o and p was in a layer after k then k would not be the latest layer containing input twists. □

6.4 Encoding of a Generalized Network

Note that we can relax the restriction that $i < j$ for variables $g^k_{i,j}$ to $i \neq j$ such that we can model generalized comparators and thus generalized comparator networks.

The generalization of the encoding is straightforward. We replace $\text{standGate}^k_{i,j}$ by $g^k_{i,j}$ and relax the restriction that $i < j$ to $i \neq j$ when quantifying.

$$\text{valid} = \bigwedge_{i\in[n]} \bigwedge_{k\in[d]} \text{exactlyOne}\Big(\{u^k_i, g^k_{i,j} \mid j \in [n], j \neq i\}\Big)$$

$$\begin{aligned}{}^{\overline{x}}\text{update} = &\bigwedge_{\substack{i,j\in[n]\\ i\neq j}} \bigwedge_{k\in[d]} g^k_{i,j} \rightarrow \Big(\min({}^{\overline{x}}v^{k+1}_i, \{{}^{\overline{x}}v^k_i, {}^{\overline{x}}v^k_j\}) \wedge \max({}^{\overline{x}}v^{k+1}_j, \{{}^{\overline{x}}v^k_i, {}^{\overline{x}}v^k_j\})\Big)\\ &\wedge \bigwedge_{i\in[n]} \bigwedge_{k\in[d]} u^k_i \rightarrow \Big({}^{\overline{x}}v^{k+1}_i \leftrightarrow {}^{\overline{x}}v^k_i\Big)\end{aligned}$$

$${}^{\overline{x}}\text{values}^k_{\overline{y}} = \bigwedge_{i\in[n]} \begin{cases} \neg\, {}^{\overline{x}}v^k_i, & \text{if } y_i = 0\\ {}^{\overline{x}}v^k_i, & \text{if } y_i = 1 \end{cases}$$

$${}^{\overline{x}}\text{sorts} = {}^{\overline{x}}\text{values}^0_{\overline{x}} \wedge {}^{\overline{x}}\text{update} \wedge {}^{\overline{x}}\text{values}^d_{\text{sort}(\overline{x})}$$

Further we discard the toBetweenBefore variable because we can model nonstandard networks in this encoding. This means more input twist symmetries are broken. Then

the CNF formula is

$$
\begin{aligned}
&\text{valid} \\
\wedge &\bigwedge_{\overline{x}\in\mathbb{B}^n} {}^{\overline{x}}\text{sorts} \\
\wedge &\bigwedge_{k\in[d-1]} \bigwedge_{\substack{i,j\in[n]\\ i\neq j}} \neg u_i^{k+1} \vee \neg u_j^{k+1} \vee \neg g_{i,j}^k \\
\wedge &\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{o,p\in[n]\\ o\neq p}} \bigwedge_{\substack{i,j\in[n]\setminus\{o,p\}\\ i\neq j}} \neg g_{i,j}^k \vee \neg g_{i,o}^{k-1} \vee \neg g_{j,p}^{k-1} \vee \text{viaWrongTwist}_{p,o}^k \\
\wedge &\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{i,j\in[n]\\ i\neq j}} \neg g_{i,j}^k \vee \neg u_i^{k-1} \\
\wedge &\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{o,p\in[n]\\ o>p}} \neg\,\text{viaWrongTwist}_{p,o}^k \vee \neg u_o^k \vee \neg u_p^k \\
\wedge &\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{o,p\in[n]\\ o\neq p}} \bigwedge_{\substack{q\in[n]\setminus\{o,p\}\\ \min(q,o)>p}} \neg\,\text{viaWrongTwist}_{p,o}^k \vee \neg g_{o,q}^k \vee \neg u_p^k \\
\wedge &\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{o,p\in[n]\\ o\neq p}} \bigwedge_{\substack{r\in[n]\setminus\{o,p\}\\ o>\min(r,p)}} \neg\,\text{viaWrongTwist}_{p,o}^k \vee \neg u_o^k \vee \neg g_{p,r}^k \\
\wedge &\bigwedge_{\substack{k\in[d]\\ k>0}} \bigwedge_{\substack{o,p\in[n]\\ o\neq p}} \bigwedge_{\substack{q,r\in[n]\setminus\{o,p\}\\ \min(q,o)>\min(r,p)}} \neg\,\text{viaWrongTwist}_{p,o}^k \vee \neg g_{o,q}^k \vee \neg g_{p,r}^k.
\end{aligned}
$$

Unfortunately this encoding seems impractical. This is not because it doubles the gate variables but because encoding counterexamples now requires significantly more clauses. In particular the number of ${}^{\overline{x}}$update constraints is doubled for every counterexample input $\overline{x}$. Recall that while we encode only a subset of the input runs, the number of encoded runs still scale exponentially with n.

Additionally for nonstandard networks we lose the ability to encode only the window of a counterexample run as mentioned in Section 4.3.3. Encoding only the window, i.e. discarding the leading zeros and trailing ones, reduces the number of variables and clauses by an amount that scales exponentially with n. It does not seem worth to add exponentially many clauses (and variables) for a more complete or even complete symmetry break of the input twists.

7 Sorting Network Graphs

This section is dedicated to the underlying graphs of sorting networks. In Section 5.1.3 we learned that one common depiction of sorting networks is a graph, hence the name network. We contrasted this with the conception of sorting networks as a sequence of position pairs that are compared and swapped by the sorting algorithm such that the network can be depicted as a Knuth diagram.

Choi and Moon (CM) formalized how (standard) comparator networks are represented as graphs in 2002 [13, 58]

Definition 22
The underlying graph of a generalized comparator network $N = c_0, c_1, \ldots, c_{s-1}$ in $\mathcal{N}_n^s$ is an acyclic directed graph that contains the comparators as vertices.

$$V = \{c_0, c_1, \ldots, c_{s-1}\}$$

If an output of a comparator c_k is channeled into comparator c_l, add a directed edge from vertex c_k to vertex c_l with a label $w \in \{-, +\}$ as a triple (c_k, w, c_l) to the set of edges E. The label w indicates whether edge e uses the minimum or maximum output of c_k. The underlying graph is then given as (V, E).

CM further formalized an isomorphism between comparator network graphs [13].

Definition 23
We write $G(N) \simeq G(N')$ to say comparator network graph $G(N)$ is isomorphic to comparator network graph $G(N')$.

For two comparator network graphs $G(N) = (V, E)$ and $G(N') = (V', E')$ an isomorphism of two comparator network graphs is a bijection $f : V \to V'$ such that $(c_i, w, c_j) \in E$ if and only if $(f(c_i), w, f(c_j)) \in E'$.

7.1 Graph Isomorphism as Network Transformations

Next we consider how CM related graph isomorphism to permuting and untangling. For this we need further setup because CM establish the relation via an intermediate isomorphy that considers formulas describing the inputs of comparators.

Let $x_0, \ldots, x_{n-1}$ be Boolean variables for the inputs to the network. We define the set of input formulas $\text{inp}(c)$ inductively. The formula x_i is in $\text{inp}(c)$ if c is the first comparator that works on channel i. Furthermore $\bigvee_{\varphi \in \text{inp}(c_k)} \varphi$ is in $\text{inp}(c_k)$ if the max output of c_l is channeled into c_k. The case for the min output is analogous with $\bigwedge$. Similarly we define the output formulas as $\text{out}(c) = \{\bigvee_{\varphi \in \text{inp}(c)} \varphi, \bigwedge_{\varphi \in \text{inp}(c)} \varphi\}$.

Any set $\text{inp}(c)$ contains at most two formulas as every comparator works on two channels. Any set $\text{out}(c)$ contains at most two comparators by definition. In fact any set $\text{inp}(c)$ or any set $\text{out}(c)$ contains exactly two formulas. For the sake of contradiction we assume that there is a network that contains comparators whose set of input formulas contained less than two formulas. Then we consider the first of those comparators. As two inputs are channeled into every comparator, there is only one way to contain less than two formulas. The formulas corresponding two the two inputs have to be equivalent such that the input formula set of the comparator has that formula as its single element. However this would violate the following lemma wrt the prefix network ending before that comparator.

Lemma 24
Given a generalized comparator network N, let the Boolean formula φ_i describe the output value of N on channel i with the network input variables $x_0, \ldots, x_{n-1}$.

$$\forall i, j \in [s]: \quad i \neq j \rightarrow \varphi_i \not\equiv \varphi_j$$

Proof by contradiction. First we prove the proposition for standard networks and later we make an argument to extend the proposition to generalized networks.

Assume that $i < j$ without loss of generality. We observe that an assignment of the network input variables corresponds to a vector of Boolean values that is input to the network. Consider the input assignment $x_a = \text{false}$ if $a \leq i$ and $x_a = \text{true}$ otherwise for all $0 \leq a < n$. This input is already sorted. Since standard comparators only exchange unsorted value pairs, the input is channeled through the network unchanged. This gives us $\varphi_i = \text{false} \neq \text{true} = \varphi_j$. Because two equivalent formulas cannot evaluate to different values for the same assignment, it follows $\varphi_i \not\equiv \varphi_j$. This proves the proposition for standard networks.

Now let N be a generalized comparator network. We know it can be untangled to a

standard network N' where φ'_i describes the output value of N' on channel i and there is a permutation $\pi \in \mathsf{Sym}([n])$ such that $\varphi_{\pi(o)} \equiv \varphi'_o$ for all o in $[n]$ [6].

$$\begin{aligned} & i \neq j \to \varphi'_i \not\equiv \varphi'_j \\ \underset{\varphi_{\pi(o)} \equiv \varphi'_o}{\iff} \quad & i \neq j \to \varphi_{\pi(i)} \not\equiv \varphi_{\pi(j)} \\ \underset{\pi \in \mathsf{Sym}([n])}{\iff} \quad & i \neq j \to \varphi_i \not\equiv \varphi_j \end{aligned}$$ □

In order to work with sets of two formulas we define equivalence of sets with two formulas as

$$\{\varphi_0, \varphi_1\} \equiv \{\psi_0, \psi_1\} \iff \exists\, \pi \in \mathsf{Sym}([2]) \;\; \forall\, i \colon \varphi_i \equiv \psi_{\pi(i)}.$$

Given some π from $\mathsf{Sym}([n])$ we write $\pi(S)$ to to apply it to every formula in S. Applying π to a formula replaces every occurence of the variable x_i with the variable $x_{\pi(i)}$.

I should mention that Choi and Moon did not work with arbitrary input formulas but rather they normalized their input formulas to a special kind of disjunctive normal form [1, p. 239, 669][13].

7.1.1 Graph Isomorphism as Permutation and Untangling of Channels

With this setup we turn to CM's intermediate isomorphy between generalized comparator networks (CM intended them to be standard) that "are structured essentially in the same way, and their input [channel numbers] may be different. In other words, a network can be transformed into another isomorphic network by appropriate permutation of the [channel numbers] and comparator [untangling]."

Definition 25 (Choi and Moon)
The isomorphism relation $\simeq$ on $\mathcal{N}_n^s$ between two generalized comparator networks $N = c_0, c_1, \ldots, c_{s-1}$ and $N' = c'_0, c'_1, \ldots, c'_{s-1}$ is defined as

$$N \simeq N' \iff \exists\, \pi \in \mathsf{Sym}([n]).\ \forall k \in [s].\ \pi(\mathrm{inp}(c_k)) \equiv \mathrm{inp}(c'_k).$$

CM also claimed that $N \simeq N' \iff G(N) \simeq G(N')$ although they omitted the proof due to limited space [13]. I attempted to reconstruct the proof but failed. The problem is that a graph isomorphism can map the kth comparator in N to the lth comparator in N', while the network isomorphism necessarily relates the kth comparator in N to the kth comparator in N'. Figure 7.1 shows a counterexample where the graphs are isomorphic but there is no π for a network isomorphism such that e.g. the inputs of the

third comparator in both networks are equivalent. That is because the inputs of the third comparator in the top network are described by a disjunction respectively, while the inputs of the third comparator in the bottom network are described by network input variables. Notably the networks in Figure 7.1 both sort.

With the motivation to understand how CM relate permuting and untangling with their graph isomorphy we wonder whether at least $N\ (P\,;O)\ N' \impliedby G(N) \simeq G(N')$ or even $N\ (P\,;O)\ N' \iff G(N) \simeq G(N')$. We observe that permuting and untangling fails in the same way that CM's intermediate network isomorphy about input formulas did. For the sake of contradiction we assume that there is a permutation π and output twists for the counterexample in Figure 7.1 such that $N\ (P\,;O)\ N'$ where N is the top network. Then π maps the channels 1 and 2 of the second comparator in the top network to the channels 5 and 6 of the second comparator in the bottom network. But there is no way to choose π or to twist the outputs of the first two comparators such that the third comparator takes two network inputs. That is because any permutation or output twist that makes the third comparator work on different channels (it has to work on channels 1 and 2) preserves the fact that the third comparator works on the outputs of two other comparators, although after all transformations the third comparator must work on two network inputs.

However we observe that the networks in Figure 7.1 are related by R, i.e. the networks are the same modulo a permutation of parrallel comparators. In particular the top network is transformed to the bottom network by letting the permutation $\rho = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 0 & 5 & 1 & 2 & 6 & 3 & 8 & 9 & 4 & 11 & 12 & 7 & 10 \end{smallmatrix}\right)$ act via $\shortparallel$ on the top network. Let the top network be $c_0, c_1, \ldots, c_{12}$ and the bottom network be $\rho \shortparallel (c_0, c_1, \ldots, c_{12}) = c'_0, c'_1, \ldots, c'_{13}$. For example the comparator c'_4 in the bottom network corresponds to comparator c_6 in the top network. That is because $c_6 = c_{\rho(4)} = c'_4$.

7.1.2 Graph Isomorphism as Permutation of Parallel Comparators

It turns out that adding R to the composition $P\,;O$ admits a proof that a CM-graph-isomorphism enables some composition of such transformations between the networks.

Proposition 26
For two generalized networks N and N' from $\mathcal{N}_n^s$ we have that

$$N\ (R\,;P\,;O)\ N' \iff G(N) \simeq G(N').$$

To prove this we first prove the following lemma.

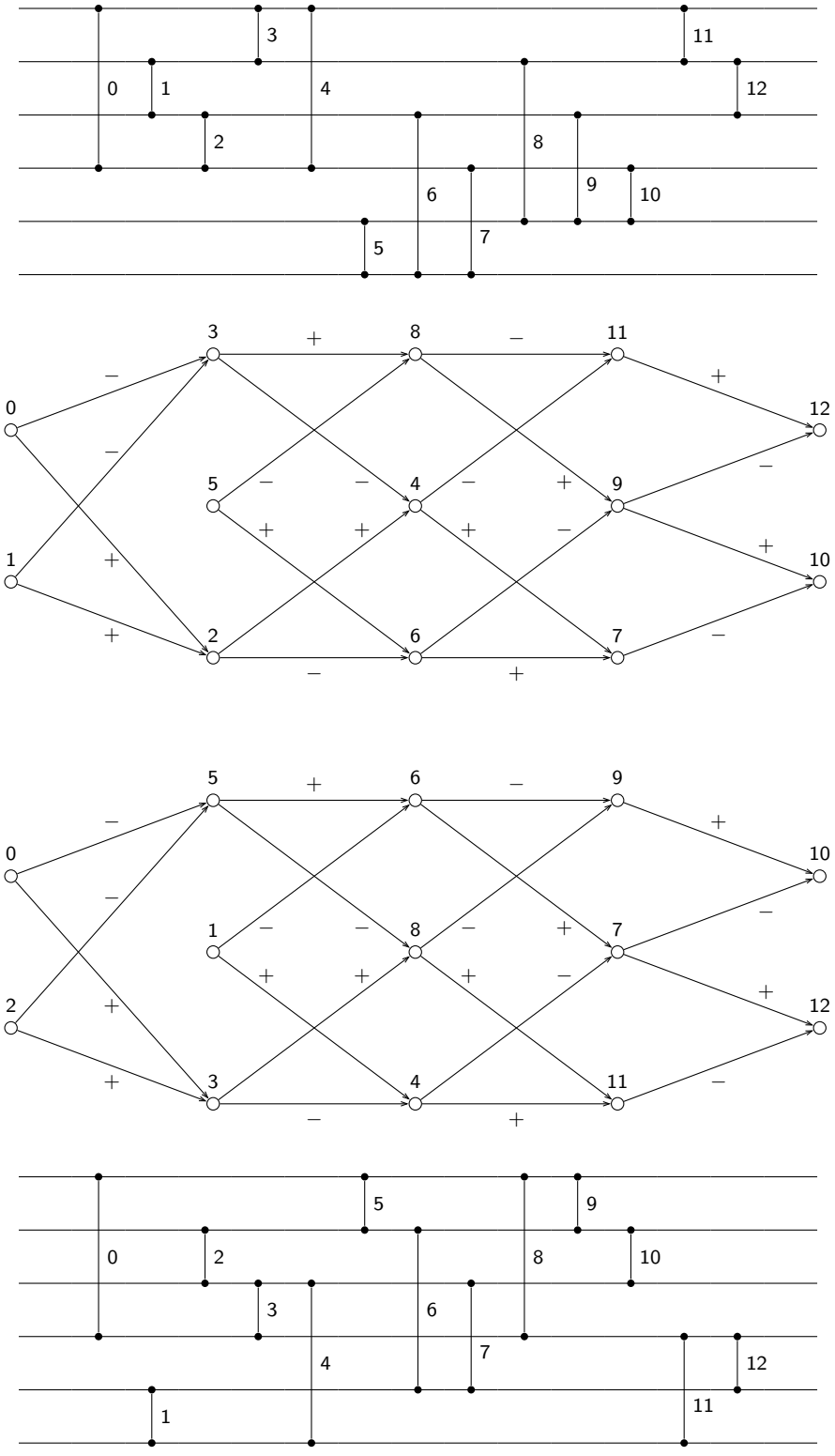

Figure 7.1: *A counterexample for* $N \simeq N' \iff G(N) \simeq G(N')$.

Lemma 27
For any generalized network $N = c_0, c_1, \ldots, c_{s-1}$ in $\mathcal{N}_n^s$ and any permutation of comparators numbers ρ from $\mathsf{Sym}(s)$ yielding the network $\rho \mid\mid N = c_0^\rho, c_1^\rho, \ldots, c_{s-1}^\rho$ the following holds. The permutation ρ only changes the order of parallel comparators iff it preserves the structure of the underlying network graph.

$$N \; R \; (\rho \mid\mid N) \iff c_{\rho(k)} \mapsto c_k^\rho \text{ is a network graph isomorphism from } G(N) \text{ to } G(\rho(N))$$

Intuitively ρ preserves the order of comparators on any channel if and only if it preserves the order of comparators on any path through the network graph.

Proof. Let $G(N) = (V, E)$ and $G(\rho(N)) = (V^\rho, E^\rho)$. We prove both directions.

$\Longrightarrow$ We show that $c_{\rho(k)} \mapsto c_k^\rho$ is a bijection such that $(c_{\rho(k)}, w, c_{\rho(l)})$ is an edge in E if and only if (c_k^ρ, w, c_l^ρ) is an edge in E^ρ.

From the premiss we know that $c_{\rho(k)}$ is immediately followed by $c_{\rho(l)}$ on a channel i in N iff c_k^ρ is immediately followed by c_l^ρ on channel i in $\rho \mid\mid N$. Without loss of generality assume that $c_{\rho(k)}$ outputs the smaller value to channel i. By definition of $\rho \mid\mid N$ we know that $c_{\rho(k)} = c_k^\rho$ and thus c_k^ρ also outputs the smaller value to channel i. By definition of edges in the network graph there is an edge $(c_{\rho(k)}, -, c_{\rho(l)})$ in E if and only if $(c_k^\rho, -, c_l^\rho)$ is an edge in E^ρ.

$\Longleftarrow$ We show that $c_{\rho(k)}$ is immediately followed by $c_{\rho(l)}$ on a channel in N iff c_k^ρ is immediately followed by c_l^ρ on the same channel in $\rho \mid\mid N$.

From the premiss we know that $c_{\rho(k)} \mapsto c_k^\rho$ is a bijection such that $(c_{\rho(k)}, w, c_{\rho(l)})$ is an edge in E if and only if (c_k^ρ, w, c_l^ρ) is an edge in E^ρ. By definition of $\rho \mid\mid N$ we know that $c_{\rho(k)} = c_k^\rho = (i, j)$. Assume without loss of generality that $i < j$ such that both $c_{\rho(k)}$ and c_k^ρ are min-max. Further assume without loss of generality that $w = -$. By definition of edges in the network graph c_k we know that $c_{\rho(k)}$ is immediately followed by $c_{\rho(l)}$ on channel i in N iff c_k^ρ is immediately followed by c_l^ρ on channel i in $\rho \mid\mid N$. □

Utilizing Lemma 7.1.2 we prove that $N \; (R \,;\, P \,;\, O) \; N' \iff G(N) \simeq G(N')$ as stated by Proposition 26.

Proof. Let $N = c_0, c_1, \ldots, c_{s-1}$ and $N = c'_0, c'_1, \ldots, c'_{s-1}$. Further let $G(N) = (V, E)$ and $G(N') = (V', E')$.

$\Longrightarrow$ Let N and N' be related by $R \,;\, P \,;\, O$. We show that then there is a network graph isomorphism from $G(N)$ to $G(N')$.

We know that N can be transformed to N' by a sequence of transformations where the first transformation is some comparator number permutation ρ from $\mathsf{Sym}([s])$ acting via $\shortparallel$ on N. By Lemma 7.1.2 we know that $G(N) \simeq G(\shortparallel N)$ where $\rho \shortparallel N$ is related to N' by $P\,;O$.

Hence it suffices to show that $G(\rho \shortparallel N) \simeq G(N')$ Since $\rho \shortparallel N$ is related to N' by $P\,;O$, there is a permutation of channel numbers π in $\mathsf{Sym}([n])$ such that $\pi \diamond (\rho \shortparallel N)$ is related to N' by R. We first show that $G(\rho \shortparallel N) \simeq G(\pi \diamond (\rho \shortparallel N))$.

Let $\rho \shortparallel N = c_0^\rho, c_1^\rho, \ldots, c_{s-1}^\rho$ and let $G(\rho \shortparallel N) = (V^\rho, E^\rho)$. Similarly let $\pi \diamond (\rho \shortparallel N) = c_0^{\rho\pi}, c_1^{\rho\pi}, \ldots, c_{s-1}^{\rho\pi}$ and let $G(\pi \diamond (\rho \shortparallel N)) = (V^{\rho\pi}, E^{\rho\pi})$.

We show that $c_k^\rho \mapsto c_k^{\rho\pi}$ is a bijection such that for all k and l in $[n]$ the edge (c_k^ρ, w, c_l^ρ) is in E^ρ iff the edge $(c_k^{\rho\pi}, c_l^{\rho\pi})$ is in $E^{\rho\pi}$. We only show that the former edge implies the latter as the proof for the reverse direction is analogous. Without loss of generality we assume that $w = -$ if there is such an edge (c_k^ρ, w, c_l^ρ) in E^ρ. Thus $c_k^\rho = (i, j)$ immediately precedes c_l^ρ on channel i. Then (c_k^ρ, c_l^ρ) is an edge in E^ρ iff $(c_k^{\rho\pi}, c_l^{\rho\pi})$ in $E^{\rho\pi}$, i.e.

$$\begin{aligned}
(c_k^\rho, c_l^\rho) \in E^\rho &\iff c_k^\rho \prec_i c_l^\rho \\
&\iff (\pi \bullet c_k^\rho) \prec_{\pi(i)} (\pi \bullet c_l^\rho) \\
&\iff c_k^{\rho\pi} \prec_{\pi(i)} c_l^{\rho\pi} \\
&\iff (c_k^{\rho\pi}, c_l^{\rho\pi}) \in E^{\rho\pi}.
\end{aligned}$$

To see that $c_k^{\rho\pi}$ really is an *immediate* predecessor of $c_l^{\rho\pi}$ on channel $\pi(i)$, we observe that there cannot be another comparator $c_m^{\rho\pi}$ with $c_k^{\rho\pi} <_{\pi(i)} c_m^{\rho\pi} <_{\pi(i)} c_l^{\rho\pi}$ because otherwise $c_k^\rho <_i c_m^\rho <_i c_l^\rho$, which contradicts $c_k^\rho \prec_i c_l^\rho$.

Finally we show that $G(\pi \diamond (\rho \shortparallel N)) \simeq G(N')$ where we know that $\pi \diamond (\rho \shortparallel N)$ and N' are related by O. Therefore $\pi \diamond (\rho \shortparallel N)$ can be transformed to N' by a sequence of output twists.

We show that $c_k^{\rho\pi} \mapsto c_k'$ is a bijection such that for all k and l in $[n]$ the edge $(c_k^{\rho\pi}, c_l^{\rho\pi})$ is in $E^{\rho\pi}$ iff the edge (c_k', w, c_l') is in E'.

We consider any transformation $M' = \tau \triangleq_k M$ in the sequence of output twists where the intermediate network M is transformed to the (intermediate) network M'. For $\tau = \mathsf{id}$ we have $G(M) \simeq G(M')$, henceforth we assume that $\tau = \mathsf{t}$.

Any edge that does not correspond to an immediate comparator precedence on one of the channels of the kth comparator (i, j) is preserved because other channels are not modified by the action $\triangleq_k$. Edges within the prefix network ending at the $k-1$th comparator are preserved because any permutation acting via $\diamond$ preserves the edges for this bijection as we have shown above. The same applies for the suffix

network starting at the kth comparator.

Finally we consider edges from the suffix to the prefix network corresponding to an immediate comparator precedence on channel i or j. If such an edge exists it goes to the k comparator, which is the first comparator of the suffix network. We observe that such edges are preserved because the prefix network remains unchanged and the comparator continues to work on the channels i and j after twisting.

$\Longleftarrow$ Let there be a graph isomorphism f from $G(N)$ to $G(N')$. We show that then N and N' are related by $R\,;P\,;O$.

First we define ρ to permute the comparators according to f:

$$k = \rho(l) \;\leftrightarrow\; f(c_k) = c'_l$$

Let $\rho \mathbin{||} N = c_0^\rho, c_1^\rho, \ldots, c_{s-1}^\rho$. For f we know (c_k, w, c_l) is an edge in $G(N)$ iff $(f(c_k), w, f(c_l))$ is an edge in $G(N')$. By definition of ρ the comparator $f(c_k)$ is the $\rho^{-1}(k)$th comparator and likewise $f(c_l)$ is the $\rho^{-1}(l)$th comparator in N'. Hence we substitute ρ for f such that $(c_{\rho(k)}, w, c_{\rho(l)})$ is an edge in $G(N)$ iff (c'_k, w, c'_l) is an edge in $G(N')$.

$$\begin{aligned}
& & (c_k, w, c_l) \in E &\leftrightarrow (f(c_k), w, f(c_l)) \in E' \\
&\Longleftrightarrow & (c_k, w, c_l) \in E &\leftrightarrow (c'_{\rho^{-1}(k)}, w, c'_{\rho^{-1}(l)}) \in E' \\
&\Longleftrightarrow & (c_{\rho(k)}, w, c_{\rho(l)}) \in E &\leftrightarrow (c'_k, w, c'_l) \in E' \\
&\Longrightarrow & (c_{\rho(k)}, w, c_{\rho(l)}) \in E &\to (\rho(k) < \rho(l) \leftrightarrow k < l) \\
&\Longrightarrow & c_{\rho(k)} \prec c_{\rho(l)} &\to (\rho(k) < \rho(l) \leftrightarrow k < l) \\
&\Longrightarrow & c_{\rho(k)} < c_{\rho(l)} &\to (\rho(k) < \rho(l) \leftrightarrow k < l) \\
&\underset{\forall\, m \in [s].\ c_{\rho(m)} = c_m^\rho}{\Longleftrightarrow} & c_{\rho(k)} < c_{\rho(l)} &\leftrightarrow c_k^\rho < c_l^\rho \\
&\Longleftrightarrow & N \;R\; (\rho \mathbin{||} N) &
\end{aligned}$$

Next we define π to permute the network channels. Let c_k be a comparator receiving a network input on some channel i. Then the comparator vertex lacks at least one of up to two ingoing edges. In particular there is no edge (c_l, w, c_k) in $G(N)$ with c_l working on channel i. Since edges are preserved by the graph isomorphism f, the property to be a comparator on one or two network inputs respectively is also preserved. Thus there is no edge $(f(c_l), w, f(c_k))$ in $G(N')$ on some other channel j. We define π such that $\pi(i) = j$. If both inputs of c_k are inputs to the network the choice of π is arbitrary.

Finally we twist the outputs of $\pi \diamond (\rho \mathbin{||} N)$ with a sequence of twists $\tau_0, \tau_1, \ldots, \tau_{s-1}$ such that the kth comparator in $\tau_{s-1} \triangleq_{s-1} \ldots \tau_1 \triangleq_1 (\tau_0 \triangleq_0 (\pi \diamond (\rho \mathbin{||} N)))$ is min-max iff the kth comparator in N' is min-max.

Now we prove that $N\ (R\,;P\,;O)\ N'$ by showing that

$$\tau_{s-1}\triangleq_{s-1}\ldots\tau_1\triangleq_1(\tau_0\triangleq_0(\pi\diamond(\rho\,||\,N)))=N'.$$

Let $\tau_{s-1}\triangleq_{s-1}\ldots\tau_1\triangleq_1(\tau_0\triangleq_0(\pi\diamond(\rho\,||\,N)))=N^{\rho\pi\tau}c_0^{\rho\pi\tau},c_1^{\rho\pi\tau},\ldots,c_{s-1}^{\rho\pi\tau}$.

Note that by direction $\Longrightarrow$ of this proof for Proposition 26 and in particular Lemma 7.1.2 the mapping $c_{\rho(k)}\mapsto c_k^{\rho\pi\tau}$ is a graph isomorphism from $G(N^{\rho\pi\tau})$ to $G(N)$. Note that the comparators position in the sequence is only changed by Lemma 7.1.2. By definition of graph isomorphisms its inverse $c_k^{\rho\pi\tau}\mapsto c_{\rho(k)}$ is a graph isomorphism mapping back from $G(N)$ to $G(N^{\rho\pi\tau})$.

Additionally by the premiss we have that f is a graph isomorphism from $G(N)$ to $G(N')$. By definition of ρ we have $f(c_k)=c_{\rho^{-1}(k)}$. Composing these two graph isomorphisms from $G(N^{\rho\pi\tau})$ to $G(N)$ and from $G(N)$ to $G(N')$ yields the graph isomorphism $c_k^{\rho\pi\tau}\mapsto c'_k$ that preserves the position k in the comparator sequence.

$$G(\tau_{s-1}\triangleq_{s-1}\ldots\tau_1\triangleq_1(\tau_0\triangleq_0(\pi\diamond(\rho\,||\,N))))\simeq G(N)\simeq G(N').$$

Therefore we can prove $N^{\rho\pi\tau}=N'$ by induction on the size of a prefix network. Let $N_k^{\rho\pi\tau}$ and N'_k denote the prefix network with the first k comparators respectively such that $N_s^{\rho\pi\tau}=N^{\rho\pi\tau}$ and $N'_s=N'$. The two prefix networks $N_0^{\rho\pi\tau}$ and N'_0 without comparators are equal. Assuming the induction hypothesis that $N_k^{\rho\pi\tau}=N'_k$ we show that $N_{k+1}^{\rho\pi\tau}=N'_{k+1}$. Thus we extend the network with comparator $c_k^{\rho\pi\tau}$ that is mapped to c'_k such that we have $G(N_{k+1}^{\rho\pi\tau}\simeq G(N'_{k+1})$.

We show that $c_k^{\rho\pi\tau}=c'_k$. It suffices to show that $c_k^{\rho\pi\tau}$ and c'_k work on the same set of channels. Then τ_k twists them such that $c_k^{\rho\pi\tau}$ is min-max iff c'_k is min-max. Hence we know that $c_k^{\rho\pi\tau}=c'_k$.

For that we make a case distinction on the number of edges that are additionally preserved when going from $G(N_k^{\rho\pi\tau})\simeq G(N'_k)$ to $G(N_{k+1}^{\rho\pi\tau})\simeq G(N'_{k+1})$

- If there are no edges additionally preserved then both c_k^ρ and c'_k work on two network inputs such that π arbitrarily maps the channels of c_k^ρ to the channels of c'_k. Thus $\pi\bullet c_k^\rho$ and c'_k work on the same channels. We cannot twist any preceding comparators on those channels as there are none. Hence $c_k^{\rho\pi\tau}$ works on those channels too.

- If there is exactly one edge additionally preserved then both c_k^ρ and c'_k work on exactly one network input respectively. Let c_k^ρ receive the network input on channel i then π maps i to the channel j on that c'_k receives its network input. Thus both $\pi\bullet c_k^\rho$ and c'_k work on channel $\pi(i)=j$. We cannot twist

any preceding comparator on i as there is none. Hence $c_k^{\rho\pi\tau}$ works on channel $\pi(i) = j$ too. Let the other channel of $c_k^{\rho\pi\tau}$ be channel o. We know that there is some comparator $c_l^{\rho\pi\tau}$ immediately preceding $c_k^{\rho\pi\tau}$ on channel o because there is an edge to $c_k^{\rho\pi\tau}$. This edge is preserved such that c'_l precedes c'_k. The comparator c'_k works on channel o too because by our induction hypothesis $c_l^{\pi\rho\tau} = c'_l$ and the label w of the edge is preserved.

If there are two edges additionally preserved, then the set of channels for $c_k^{\rho\pi\tau}$ and c'_k is shown to be the same analogously to the case that exactly one edge is additionally preserved. In particular if two edges from the same comparator are additionally preserved, then they have different labels $+$ and $-$ such that each edge corresponds unambigously to one shared channel. □

Choi and Moon pointed out that this particular instance of graph isomorphism is decidable in polynomial time as the degree in graphs underlying a comparator network is bounded by four.

7.2 Graph Isomorphism as Network Isomorphism

We turn to Definition 25 of the isomorphism between networks in contrast to the isomorphism between their underlying graphs. CM's isomorphism relates networks with the same set of input formulas at every comparator. While we have resolved how the graph isomorphism corresponds to permuting and untangling, we have not yet proven CM's initial claim that $N \simeq N' \iff G(N) \simeq G(N')$. This seems desirable because CM's network isomorphism clearly defines whether two networks work in the same way. It seems natural to amend the definition of the network isomorphism by adding a permutation of comparator numbers ρ.

Definition 28 (Haslop)
The isomorphism relation $\simeq$ on $\mathcal{N}_n^s$ between two generalized comparator networks $N = c_0, c_1, \ldots, c_{s-1}$ and $N' = c'_0, c'_1, \ldots, c'_{s-1}$ is defined as

$$N \simeq N' \iff \exists\, \pi \in \mathsf{Sym}([n]).\ \exists\, \rho \in \mathsf{Sym}([s]).\ \forall k \in [s].\ \pi(\mathrm{inp}(c_k)) \equiv \mathrm{inp}(c'_{\rho(k)}).$$

There still is a counterexample that utilizes redundant comparators to disprove $N \simeq N' \iff G(N) \simeq G(N')$. One such counterexample is shown in Figure 7.2 where both networks are isomorphic to each other but their underlying graphs are not.

In both networks the first three comparators sort the input such that the last two comparators have no effect. Hence the permutation of channel numbers $\pi = \mathsf{id}$ and the

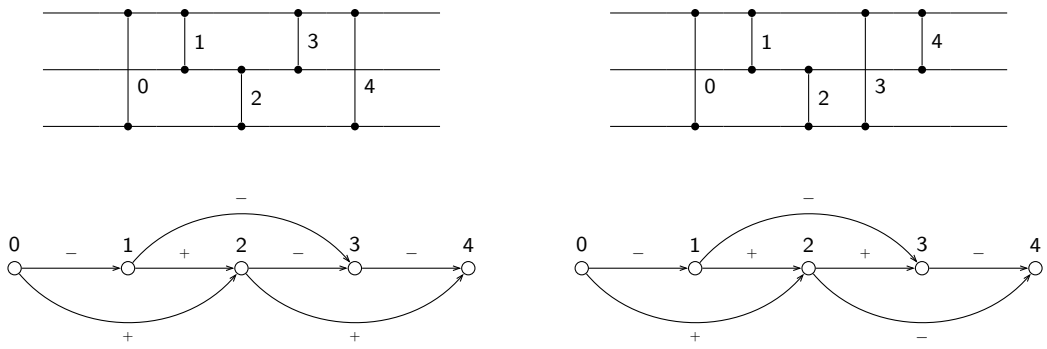

Figure 7.2: *A counterexample for* $N \simeq N' \iff G(N) \simeq G(N')$ *where* $N \simeq N'$ *uses the amended Definition 28 instead of Definition 25 by Choi and Moon.*

permutation of comparator numbers $\rho = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 4 & 3 \end{smallmatrix}\right)$ satisfy the conditions of a network isomorphism. We have to swap the last two comparators because the input set with the formulas evaluating to the smallest and median value of the network input variables respectively differs from the input set with the formulas that evaluate to the smallest and the greatest value of network input variables respectively.

However the graphs of the networks are not isomorphic. For that we consider the two ingoing edges of the single vertex without any outgoing edges. In the left graph both ingoing edges are labeled with $-$, whereas the labels differ in the right graph. An isomorphism would necessarily preserve not having any outgoing edge, but it would necessarily preserve the labels of ingoing edges too. Thus an isomorphism between the underlying graphs is impossible.

Note that such a counterexample may also appear as subnetworks of networks that sort more than three channels where the redundant comparators are not necessarily the last comparators.

Although we encountered yet another counterexample, I conjecture that $N \simeq N' \iff G(N) \simeq G(N')$ holds if both N and N' contain no trivial comparators. To be specific if N and N' are generalized networks, they must contain no comparator c with $\mathrm{inp}(c) \equiv \mathrm{out}(c)$, i.e. there must neither be a comparator that never swaps nor one that always swaps. This notion of a trivial comparator is one possible generalization of the notion of a redundant comparator in a standard sorting network. A redundant comparator never swaps. Notably comparators that always swap can only occur in generalized comparator networks. Thus in a standard network, every trivial comparator is redundant and vice versa.

8 Conclusion

We considered the optimization problems that ask for the minimum size or depth of a sorting network. Specifically we considered their related decisions problems for small n. With respect to that this thesis provides three main results. We saw that sorting networks are rich in symmetries. Breaking these symmetries is crucial to proving upper bounds for the optimization problem as this requires an exhaustive search over all networks.

First Result: Input Twists

Reflection as well as permuting and untangling of channels are well known symmetries on standard networks. We defined symmetries on generalized instead of standard networks. Specifically we defined the permutation of channels or parallel comparators as well as the input or output twist of a comparator. In particular twisting the outputs of every comparator covers untangling. Hence we composed permuting channels with twisting outputs to obtain a variant of permuting and untangling on generalized networks. As the first main result this work provides the, to my knowledge, first correct analysis of input twists, which relates them to permuting and untangling.

While standard networks only admit sorting networks, generalized networks admit π-sorting networks for different π. For the analysis of the symmetries we said that a symmetry between two networks is correct under permuted outputs if it preserves to be π-sorting for some π. In constrast a strongly correct symmetry preserves to be π-sorting for the same π. Permuting and twisting outputs are both correct under permuted outputs, while both twisting inputs and permuting parallel comparators are strongly correct. The highlight of the analysis shows that twisting inputs is in fact complete relative to permuting and untangling. Hence the symmetry by input twists is suited as a more precise substitute for the symmetry by permuting and untangling.

Second Result: Symmetry Breaking Constraints

For the second result we considered the state of the art where the problem is reduced to Boolean SAT. We observed that the restriction to standard networks breaks channel

permutations and outputs twists respectively in a way such that their composition is not completely broken. As the second main result this thesis provides additional symmetry breaking constraints for the state of the art SAT encoding that further break the composed symmetry of permuting channels and twisting outputs. Exploiting the first result these symmetry constraints simply break input twists between standard networks.

We noted that a generalized encoding allows a more complete break of input twists. We did not break the special case that two comparators in the same layer are connected by two comparators in the immediately preceding layer. However I believe this is possible with some tedious case analysis such that input twists are broken completely in the generalized encoding. I did not run a SAT solver to solve the encodings that I propose due to time constraints. I leave both the tedious case analysis as well as the evaluation of the symmetry breaking constraints in each encoding for future work.

Further there is some leeway how to break input twists, i.e. what comparator configuration we allow as the representative in the preceding layer. We saw that in the encoding of a standard network we cannot break the input twists of a comparator on channels i and j if there is a comparator from either i or j to a channel between i and j in an earlier layer. To minimize the cases where this condition applies, it may prove useful to develop heuristics for choosing representatives accordingly in future work.

Third Result: Sorting Network Graph Isomorphism

The third and final result states that the symmetry that permutes channels, twists outputs of comparators and permutes parallel comparators corresponds to an isomorphism between the underlying graphs as defined by Choi and Moon. Unlike claimed by Choi and Moon the permutation of parallel comparators is another component of the isomorphism between underlying graphs.

Additionally I suggested that considering the permutability of parallel comparators similarly fixes CM's proposition that an isomorphism between their graphs corresponds to the networks working in the same way. Proving that this actually rectifies the proposition is yet another open task.

The permutation of parallel comparators seems rather obvious and is rarely explicitely mentioned. E.g. the state of the art approach forces the first layer to be maximal such that every comparator in the two-layer prefix is moved to the earliest layer possible. For the second result we employed a generalization of this scheme to the whole network.

A Case for Extensive Symmetry Breaking

While the impact of the proposed symmetry breaking constraints needs to be evaluated first, this thesis was my attempt to argue for extensive symmetry breaking. With extensive I refer to breaking by symmetries wrt all layers instead of only the first two or last two layers. I agree that precomputing representatives and thus fixing variable assignments is the optimal way to speed up SAT solving with symmetry knowledge. However prior work indicates that the extent of such symmetry breaking is quickly limited by the computational resources if the ambition is to break the considered symmetries completely.

This thesis as well as the work in our bachelor project suggest an extensive albeit incomplete approach to breaking by symmetries. This proposed approach passes on the optimal way of fixing prefixes and increases the size of the encoding for a potential suboptimal (due to its incompleteness) gain in efficiency. Nonetheless I speculate that we do not lose the opportunity to optimally break symmetries by precomputing symmetries outside the SAT encoding because we can alternatively fix the suffix instead of the prefix. This is because the suffix is unaffected by input twists and we can similarly move every comparator to the latest layer possible.

Bibliography

[1] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*, 2nd ed. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[2] I. Wald, P. Slusallek, C. Benthin, and M. Wagner, "Interactive distributed ray tracing of highly complex models," in *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. London, UK, UK: Springer-Verlag, 2001, pp. 277–288. [Online]. Available: http://dl.acm.org/citation.cfm?id=647653.732298

[3] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 307–314. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468121

[4] M. Codish, L. Cruz-Filipe, M. Nebel, and P. Schneider-Kamp, "Applying sorting networks to synthesize optimized sorting libraries," *CoRR*, vol. abs/1505.01962, 2015. [Online]. Available: http://arxiv.org/abs/1505.01962

[5] T. Philipp and P. Steinke, "Pblib – a library for encoding pseudo-boolean constraints into cnf," in *Theory and Applications of Satisfiability Testing – SAT 2015*, ser. Lecture Notes in Computer Science, M. Heule and S. Weaver, Eds. Springer International Publishing, 2015, vol. 9340, pp. 9–16.

[6] I. Parberry, *Parallel Complexity Theory*. New York, NY, USA: John Wiley & Sons, Inc., 1987.

[7] G. Perrot, S. Domas, and R. Couturier, "Fine-tuned high-speed implementation of a gpu-based median filter," *Journal of Signal Processing Systems*, vol. 75, no. 3, pp. 185–190, Jun. 2014. [Online]. Available: https://doi.org/10.1007/s11265-013-0799-2

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[9] J. A. R. Fonollosa, "Joint size and depth optimization of sorting networks," *CoRR*, vol. abs/1806.00305, 2018. [Online]. Available: http://arxiv.org/abs/1806.00305

[10] I. Parberry, "A computer assisted optimal depth lower bound for sorting networks with nine inputs," in *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '89. New York, NY, USA: ACM, 1989, pp. 152–161. [Online]. Available: http://doi.acm.org/10.1145/76263.76280

[11] D. Bundala and J. Závodný, "Optimal sorting networks," *CoRR*, vol. abs/1310.6271, 2013. [Online]. Available: http://arxiv.org/abs/1310.6271

[12] M. Codish, L. Cruz-Filipe, T. Ehlers, M. Müller, and P. Schneider-Kamp, "Sorting networks: to the end and back again," *CoRR*, vol. abs/1507.01428, 2015. [Online]. Available: http://arxiv.org/abs/1507.01428

[13] S.-S. Choi and B.-R. Moon, "Isomorphism, normalization, and a genetic algorithm for sorting network optimization," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 327–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=2955491.2955548

[14] (2002) Sat competition. [Online]. Available: http://www.satcompetition.org/

[15] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960. [Online]. Available: http://doi.acm.org/10.1145/321033.321034

[16] J. Marques-Silva, I. Lynce, and S. K. Malik, "Conflict-driven clause learning sat solvers," in *Handbook of Satisfiability*, 2009.

[17] J. Smock. (2016) A peek inside sat solvers. Clojure/conj. [Online]. Available: http://jonsmock.com/sat-talk/

[18] H. Katebi, K. A. Sakallah, and J. a. P. Marques-Silva, "Empirical study of the anatomy of modern sat solvers," in *Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing*, ser. SAT'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 343–356. [Online]. Available: http://dl.acm.org/citation.cfm?id=2023474.2023510

[19] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, Jun. 2001, pp. 530–535.

[20] I. P. Gent, K. E. Petrie, and J.-F. Puget, "Symmetry in constraint programming," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds. New York, NY, USA: Elsevier Science Inc., Aug. 2006, ch. 10, pp. 329–376.

[21] D. A. R. Wallace, "Permutation groups," in *Groups, Rings and Fields*. Berlin, Heidelberg: Springer-Verlag, 1998, ch. 6, pp. 191–204.

[22] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy, "Symmetry-breaking predicates for search problems," in *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, ser. KR'96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 148–159. [Online]. Available: http://dl.acm.org/citation.cfm?id=3087368.3087386

[23] H. Katebi, K. A. Sakallah, and I. L. Markov, "Symmetry and satisfiability: An update," in *Theory and Applications of Satisfiability Testing – SAT 2010*, O. Strichman and S. Szeider, Eds. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 113–127.

[24] F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Solving difficult instances of boolean satisfiability in the presence of symmetry," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 9, pp. 1117 – 1137, Oct. 2003.

[25] A. Smaill, "Symmetry in boolean constraints: Some complexity issues," in *Tenth Workshop on Automated Reasoning*, 2003. [Online]. Available: http://arw.csc.liv.ac.uk/year/2003/abstracts/abstracts.html

[26] B. Schaafsma, M. J. Heule, and H. Maaren, "Dynamic symmetry breaking by simulating zykov contraction," in *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, ser. SAT '09, O. Kullmann, Ed. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 223–236.

[27] F. A. Aloul, K. A. Sakallah, and I. L. Markov, "Efficient symmetry breaking for boolean satisfiability," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, pp. 271–276. [Online]. Available: http://dl.acm.org/citation.cfm?id=1630659.1630699

[28] J. Devriendt, B. Bogaerts, M. Bruynooghe, and M. Denecker, "Improved static symmetry breaking for sat," in *Theory and Applications of Satisfiability Testing – SAT 2016*, N. Creignou and D. Le Berre, Eds. Cham: Springer International Publishing, 2016, pp. 104–122.

[29] H. Metin, S. Baarir, M. Colange, and F. Kordon, "Cdclsym: Introducing effective symmetry breaking in sat solving," in *Tools and Algorithms for the Construction and Analysis of Systems*, D. Beyer and M. Huisman, Eds. Cham: Springer International Publishing, 2018, pp. 99–114.

[30] R. D. Maddux, "A perspective on the theory of relation algebras," *algebra*

universalis, vol. 31, no. 3, pp. 456–465, Sep. 1994. [Online]. Available: https://doi.org/10.1007/BF01221799

[31] O. Angel, A. E. Holroyd, D. Romik, and B. Virág, "Random sorting networks," *Advances in Mathematics*, vol. 215, no. 2, p. 839–868, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.aim.2007.05.019

[32] M. Ajtai, J. Komlós, and E. Szemerédi, "An 0(n log n) sorting network," in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '83. New York, NY, USA: ACM, 1983, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/800061.808726

[33] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 307–314. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468121

[34] I. Parberry, "The pairwise sorting network," *Parallel Processing Letters*, vol. 2, pp. 205–211, 1992. [Online]. Available: https://doi.org/10.1142/S0129626492000337

[35] ——, *On the Computational Complexity of Optimal Sorting Network Verification*. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 252–269.

[36] S. Arora and B. Barak, *Computational Complexity: A Modern Approach (Draft)*, Jan. 2007. [Online]. Available: http://theory.cs.princeton.edu/complexity/

[37] R. W. Floyd and D. E. Knuth, "The bose-nelson sorting problem," in *A Survey of Combinatorial Theory*, J. N. Srivastava, Ed. North-Holland, 1973, ch. 15, pp. 163–172. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978072042262750020X

[38] D. C. van Voorhis, "Toward a lower bound for sorting networks," in *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, 1972, pp. 119–129. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_12

[39] V. K. Valsalam and R. Miikkulainen, "Using symmetry and evolutionary search to minimize sorting networks," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 303–331, Feb. 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2567709.2502591

[40] L. K. Graham and F. Oppacher, "Symmetric comparator pairs in the initialization of genetic algorithm populations for sorting networks," *2006 IEEE International Conference on Evolutionary Computation*, pp. 2845–2850, Jul. 2006.

[41] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, "Twenty-five comparators is optimal when sorting nine inputs (and twenty-nine for ten)," *CoRR*, vol. abs/1405.5754, 2014. [Online]. Available: http://arxiv.org/abs/1405.5754

[42] T. Ehlers and M. Müller, "New bounds on optimal sorting networks," *CoRR*, vol. abs/1501.06946, 2015. [Online]. Available: http://arxiv.org/abs/1501.06946

[43] S. W. A. Baddar and K. E. Batcher, "An 11-step sorting network for 18 elements," *Parallel Processing Letters*, vol. 19, no. 1, pp. 97–103, 2009. [Online]. Available: https://doi.org/10.1142/S0129626409000092

[44] T. Ehlers, "Merging almost sorted sequences yields a 24-sorter," *Inf. Process. Lett.*, vol. 118, no. C, p. 17–20, Feb. 2017. [Online]. Available: https://doi.org/10.1016/j.ipl.2016.08.005

[45] M. Marinov and D. Gregg, "Higher lower bounds for the minimal depth of n-input sorting networks," 2014.

[46] ——, "Sorting networks: The final countdown," *CoRR*, vol. abs/1502.05983, 2015. [Online]. Available: http://arxiv.org/abs/1502.05983

[47] C. Frăsinaru and M. Răschip, "An improved subsumption testing algorithm for the optimal-size sorting network problem," *CoRR*, vol. abs/1707.08725, 2017. [Online]. Available: http://arxiv.org/abs/1707.08725

[48] T. Ehlers and M. Müller, "Faster sorting networks for 17, 19 and 20 inputs," *CoRR*, vol. abs/1410.2736, 2014. [Online]. Available: http://arxiv.org/abs/1410.2736

[49] A. Solar-Lezama, "Program synthesis by sketching," Ph.D. dissertation, Berkeley, CA, USA, 2008.

[50] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 502–518.

[51] M. Marinov and D. Gregg, "Itemset isomorphism: Gi-complete," *CoRR*, vol. abs/1507.05841, 2015. [Online]. Available: http://arxiv.org/abs/1507.05841

[52] M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp, "The quest for optimal sorting networks: Efficient generation of two-layer prefixes," *CoRR*, vol. abs/1404.0948, 2014. [Online]. Available: http://arxiv.org/abs/1404.0948

[53] M. Marinov and D. Gregg, "Towards optimal sorting networks: The third level," *CoRR*, vol. abs/1502.04748, 2015. [Online]. Available: http:

//arxiv.org/abs/1502.04748

[54] M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp, "Sorting networks: the end game," *CoRR*, vol. abs/1411.6408, 2014. [Online]. Available: http://arxiv.org/abs/1411.6408

[55] J. Krystofiak, G. Marquardt, and T. Haslop, "Sorting networks," in *Project Report - Tackling Practical Problems with Solvers*, 2017, pp. 62–82, not published.

[56] R. C. Bose and R. J. Nelson, "A sorting problem," *J. ACM*, vol. 9, no. 2, pp. 282–296, Apr. 1962. [Online]. Available: http://doi.acm.org/10.1145/321119.321126

[57] G. Bilardi, "Merging and sorting networks with the topology of the omega network," *IEEE Trans. Comput.*, vol. 38, no. 10, p. 1396–1403, Oct. 1989. [Online]. Available: https://doi.org/10.1109/12.35835

[58] S. Choi and B. R. Moon, "More effective genetic search for the sorting network problem," in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*, 2002, pp. 335–342.