On the Isomorphism Problem for Decision Trees and Decision Lists[☆]

V. Arvind[a], Johannes Köbler[b], Sebastian Kuhnert[b], Gaurav Rattan[a], Yadu Vasudev[a]

[a]*The Institute of Mathematical Sciences, Chennai, India*
[b]*Institut für Informatik, Humboldt-Universität zu Berlin, Germany*

Abstract

We study the complexity of isomorphism testing for boolean functions that are represented by decision trees or decision lists. Our results are the following:

- Isomorphism testing of rank 1 decision trees is complete for logspace.
- For any constant $r \geq 2$, isomorphism testing for rank r decision trees is polynomial-time equivalent to Graph Isomorphism. As a consequence of our reduction, we obtain our main result for decision trees: A $2^{\sqrt{n}(\log s)^{O(1)}}$ time algorithm for isomorphism testing of decision trees of size s over n variables.
- The isomorphism problem for decision lists admits a Schaefer-type trichotomy: depending on the class of base functions, the isomorphism problem is either in L, or polynomial-time equivalent to Graph Isomorphism, or coNP-hard.

Keywords: Boolean function isomorphism, graph isomorphism, logspace completeness

1. Introduction

Two boolean functions $f, g\colon \{0,1\}^n \to \{0,1\}$ are said to be *isomorphic* (in symbols: $f \cong g$) if there is a permutation $\pi \in S_n$ so that $f^\pi = g$, meaning that $f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$ and $g(x_1, x_2, \ldots, x_n)$ are identical boolean functions. The *boolean function isomorphism problem* (*Boolean Isomorphism* for short) is to test if two given boolean functions f and g are isomorphic. Naturally, the complexity of this problem depends on how the boolean functions f and g are represented when given as input. For example, f and g could be given as input simply by their respective truth-tables. In this case, of course, the input is of size $N = 2^{n+O(1)}$, and the naive isomorphism algorithm that does a brute-force search for $\pi \in S_n$ such that $f^\pi = g$ runs in time $N^{\lg\lg N + O(1)}$. Indeed, there is an $N^{O(1)}$ time algorithm for this case due to Luks [Luk99]. On the other hand, if the input formulas f and g are given as 3-CNF formulas, then the problem is coNP-hard because f is unsatisfiable if and only if f is isomorphic to the constant formula $g = 0$. Thus, the complexity of Boolean Isomorphism crucially depends on the representation of the input functions.

The isomorphism problem for functions given as boolean circuits, boolean formulas (general, as well as CNF/DNF), and branching programs has been studied before [AT00, Thi00]. It is easy to see that the isomorphism for all these representations is in Σ_2^p. And, as observed above, the problem is coNP-hard even for 3-CNF/3-DNF formulas. Furthermore, Agrawal and Thierauf [AT00] also show that boolean circuit

[☆]An abbreviated version of this paper appears in the proceedings of FCT 2013. This work was supported by the Alexander von Humboldt Foundation in its research group linkage program. The third author was supported by DFG grant KO 1053/7-2.

Email addresses: `arvind@imsc.res.in` (V. Arvind), `koebler@informatik.hu-berlin.de` (Johannes Köbler), `kuhnert@informatik.hu-berlin.de` (Sebastian Kuhnert), `grattan@imsc.res.in` (Gaurav Rattan), `yadu@imsc.res.in` (Yadu Vasudev)

isomorphism is not hard for the complexity class Σ_2^p unless the Polynomial-Time Hierarchy, PH, collapses to the third level Σ_3^p. Along similar lines, Thierauf [Thi00] has further shown that the isomorphism problem for *read-once* branching programs is not NP-complete unless PH collapses to Σ_2^p.

However, interesting questions remain regarding Boolean Isomorphism, especially about its connection to Graph Isomorphism: recall that *Graph Isomorphism* (GI) is the problem of checking if two input graphs G_1 and G_2 are *isomorphic* under a bijection of their vertex sets. Suppose f and g are boolean functions given as, say, boolean circuits of size s. Then, in $O(s^2 2^n)$ time, we can convert them into their respective truth-table representations and check if they are isomorphic in time $2^{O(n)}$ using Luks's algorithm [Luk99], already mentioned above. On the other hand, the best known algorithm for Graph Isomorphism has running time $2^{O(\sqrt{n \log n})}$ [BL83]. An obvious bottleneck in obtaining a faster algorithm for Boolean Isomorphism is that any algorithm for it also solves the equivalence problem! Thus, it seems difficult to obtain a $2^{o(n)} s^{O(1)}$ time algorithm for Boolean Isomorphism when f and g are given as boolean circuits of size s because nothing better than a $2^n s^{O(1)}$ time algorithm is known for the satisfiability problem for such circuits [IPZ01].

In this context it is natural to study the following questions:

- For which representations of boolean functions is Boolean Isomorphism polynomial-time equivalent to Graph Isomorphism?
- For a given representation of boolean functions, what influence has the complexity of the corresponding equivalence problem on the complexity of Boolean Isomorphism?

Böhler et al. address these questions in the nice setting of constraint satisfaction problems [BHRV04, BHRV02]. The setting is nice because of dichotomy results in the field, like Schaefer's theorem [Sch78]. Among the several results in [BHRV04, BHRV02], the main contribution is a trichotomy theorem (Theorem 5.3) which classifies Boolean Isomorphism arising from CSP representations as one of: polynomial-time solvable, equivalent to Graph Isomorphism, or coNP-hard.

A key idea in the work of [BHRV04, BHRV02] is the notion of a *normal form* of a boolean function f, represented as a CSP, where: (a) equivalent boolean functions have the same normal form, and (b) the normal form of f^π can be obtained by first computing the normal form of f and then applying π to it. This notion allows us to pass from a semantic to a syntactic notion of isomorphism and then reduce the problem to Graph Isomorphism. In fact, the notion of similar normal forms also plays a crucial role in the Agrawal-Thierauf interactive protocol result [AT00] for Boolean Isomorphism (for the boolean circuit representation). We note that the "normal form" used in [AT00] is actually a probability distribution on formulas and not a single normal form formula. It is the output of a randomized learning algorithm (using an NP oracle) for boolean circuits.

In this paper, our aim is to explore boolean function representations for which the isomorphism problem has faster algorithms than in the general case when the functions are given as circuits. We focus on the problem when the functions are given as decision trees and decision lists. Decision trees are a natural representation for boolean functions and are fundamental to boolean function complexity due to their conceptual simplicity. See, for example, the beautiful survey by Buhrman and de Wolf [BdW02] on complexity measures for boolean functions and the central role of decision tree complexity in the field. Decision lists were introduced as a flexible representation for boolean functions by Rivest [Riv87] in the context of machine learning. In the field of algorithmic learning theory, decision trees too have played a significant role in learnability of boolean functions. The quasipolynomial time PAC learning algorithms of constant-depth circuits under the uniform distribution due to Linial, Mansour and Nisan [LMN93] and the quasipolynomial time PAC learning algorithm of decision trees under uniform distribution by Kushilevitz and Mansour [KM93] are important basic results in the area. Our interest in these representations is the boolean function isomorphism problem, especially in the context of the two questions raised above.

Definition 1.1. *A decision tree T on variables $x_1, \ldots, x_n$ is an ordered binary tree in which each leaf is labeled with a boolean value and each inner node has exactly two children and is labeled with a variable x_i. Any assignment $a_1, \ldots, a_n$ defines a path from the root of T to a leaf: At an inner node labeled with x_i, proceed to the left child if $a_i = 0$ and to the right child otherwise. The function value $T(a_1, \ldots, a_n)$ is the label of the leaf node reached along this path.*

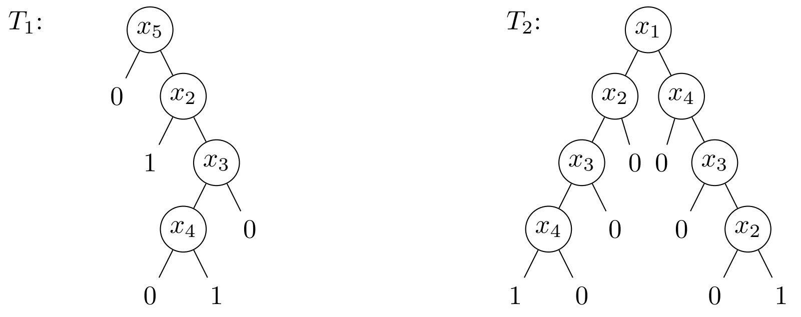

Figure 1: The decision tree T_1 computes the function $x_5 \wedge (\overline{x_2} \vee \overline{x_3} \wedge x_4)$ and has rank 1. The decision tree T_2 computes the function $x_1 \wedge x_2 \wedge x_3 \wedge x_4 \vee \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3} \wedge \overline{x_4}$ and has rank 2.

The *size* $|T|$ of a decision tree T is the number of its leaves. Using a simple preprocessing step, we can assume that on the path from the root to any leaf, each variable occurs at most once as the label of an inner node. Indeed, querying the same variable a second time will always yield the same result as before, so the second occurrence can be removed together with the subtree rooted at its non-reachable child without changing the represented function. From this point on, we will assume that each variable is queried at most once on each path of the decision tree.

The satisfiability and equivalence problems for decision trees have simple polynomial-time algorithms, implying that the isomorphism problem for decision trees, denoted DT-Iso, is in NP. Given a decision tree T, the boolean function represented by T is satisfiable if and only if one of the leaves of T is labeled with the constant 1. For checking the equivalence of two boolean functions f and g given as decision trees T_f and T_g, we can construct a decision tree T for the function $f \oplus g$. Then f and g are equivalent if and only if $f \oplus g$ is unsatisfiable. To construct the decision tree T for $f \oplus g$, we attach the decision tree T_g to the leaves of T_f that are labeled with 0 and we attach the decision tree $T_{\overline{g}}$ (obtained by complementing the leaves of the decision tree T_g) to the leaves of T_f that are labeled with 1. We can then prune this decision tree to remove nodes with the same label on a path to obtain the decision tree T. To check equivalence it is sufficient to check if all the leaves of the decision tree T are labeled with the constant 0.

The rank of a decision tree T is the depth of the largest full binary tree that can be embedded into T; a formal definition of decision tree rank is given at the beginning of Section 2. The rank of a boolean function f is the minimum rank over all decision trees computing f. In Section 3 we describe a logspace canonization algorithm for decision trees that may have arbitrary rank but compute a function of rank 1. Further, it turns out that isomorphism of rank 1 decision trees is complete for deterministic logspace.

Our main result for decision trees is in Section 4 where we give a $2^{\sqrt{n}(\log s)^{O(1)}}$ time algorithm for isomorphism testing of size s decision trees over n variables. We obtain this result by examining the connection between bounded rank decision trees and hypergraphs of bounded rank, where the rank of a hypergraph is the maximum size of its hyperedges. It turns out that a rank r decision tree of size s can be encoded as a hypergraph of rank $O(r)$ and this transformation can be carried out in time $(sn^r)^{O(1)}$. Since decision trees of size s have rank at most $\log s$, this gives the $2^{\sqrt{n}(\log s)^{O(1)}}$ time algorithm for isomorphism by applying the algorithm for bounded rank hypergraph isomorphism described in [BC08].

Section 5 treats the next main topic of the paper – the isomorphism problem for decision lists. Decision lists were originally introduced by Rivest [Riv87] in learning theory.

Definition 1.2 (cf. [Riv87]). *Let $\mathcal{C}$ be a finite class of boolean functions. A $\mathcal{C}$-decision list ($\mathcal{C}$-DL) L is a sequence of pairs $\langle f_i, c_i \rangle_{i \le m}$ where $c_i \in \{0,1\}$, $f_m = 1$, and for $i = 1, \ldots, m-1$, $f_i(x_1, \ldots, x_n) = g_i(x_{i_1}, \ldots, x_{i_k})$ for some $g_i \in \mathcal{C}$ and indices $1 \le i_1, \ldots, i_k \le n$. For a boolean assignment b, the decision list L has the value $L(a) = c_i$, where $i = \min\{j \ge 1 \mid f_j(b) = 1\}$.*

In his original definition, Rivest [Riv87] considered r-decision lists (r-DLs in short), which are $\mathcal{C}$-decision lists where $\mathcal{C}$ consists of conjunctions of r literals. He observed that for any r-DNF $T_1 \vee \cdots \vee T_l$, there is an equivalent r-DL $\langle T_1, 1\rangle \cdots \langle T_l, 1\rangle\langle 1, 0\rangle$, and for any r-CNF $C_1 \wedge \cdots \wedge C_l$, there is an equivalent r-DL

$\langle \overline{C_1}, 0\rangle \cdots \langle \overline{C_l}, 0\rangle\langle 1, 1\rangle$. Rivest's observations imply that for $r \geq 3$, the satisfiability problem for r-DLs is NP-complete, and the equivalence problem is coNP-complete. Furthermore, he proved that there are r-decision lists for which neither an equivalent r-DNF nor an equivalent r-CNF formula exists [Riv87, Theorem 2], showing that r-DLs are strictly more expressive than formulas in r-CNF or r-DNF.

The classes of 1-decision lists and rank 1 decision trees coincide. For example, the decision tree T_1 from Figure 1 is equivalent to the decision list $\langle \overline{x_5}, 0\rangle\langle \overline{x_2}, 1\rangle\langle x_3, 0\rangle\langle \overline{x_4}, 0\rangle\langle 1, 1\rangle$. More generally, every rank r decision tree of size s has an r-decision list of length $O(s)$ [Blu92]. Our results on the complexity of the isomorphism problem for $\mathcal{C}$-DLs, denoted $\mathcal{C}$-DL-Iso, are summarized below.

1. $\mathcal{C}$-DL-Iso is in L if all functions in $\mathcal{C}$ are parities of at most 2 literals, or disjunctions of such parities. It is also L-hard if $\mathcal{C}$ contains a non-constant function.

2. $\mathcal{C}$-DL-Iso is GI-complete,[1] when $\mathcal{C}$ consists of one of the following: (i) 2-DNFs, (ii) complements of Horn-CNFs, (iii) complements of anti-Horn-CNFs, and (iv) disjunctions of parities of literals such that at least one parity has size at least 3.

3. In all other cases for $\mathcal{C}$, $\mathcal{C}$-DL-Iso is both coNP-hard and GI-hard.[2]

The above results show a Schaefer-type trichotomy for the $\mathcal{C}$-DL isomorphism problem. It is interesting to compare this with the trichotomy result for $\mathcal{C}$-CSP isomorphism problems proved by Böhler et al. [BHRV04]. They show that $\mathcal{C}$-CSP isomorphism is in P if $\mathcal{C}$ consists of conjunctions of parities of size at most 2; it is GI-complete if $\mathcal{C}$ consists of one of the following: (i) 2-CNFs, (ii) Horn-CNFs, (iii) anti-Horn-CNFs, and (iv) conjunctions of parities such that at least one parity has size at least 3; and that in all other cases, $\mathcal{C}$-CSP isomorphism is both coNP-hard and GI-hard. As any $\mathcal{C}$-CSP can be easily transformed into an equivalent $\overline{\mathcal{C}}$-DL, where $\overline{\mathcal{C}}$ contains all complementary constraints $\neg C$ for $C \in \mathcal{C}$, the representation classes appearing in our trichotomy are extensions of the classes appearing in the Böhler et al. trichotomy result.

Additionally, we generalize the $\mathsf{P}_{\|}^{\mathsf{NP}}$ upper bound of Böhler et al. for $\mathcal{C}$-CSP isomorphism [BHRV02] to $\mathcal{C}$-DL isomorphism. This complexity class contains all problems that can be solved in polynomial time using one round of parallel queries to an NP oracle.

2. Preliminaries and basic facts

We recall the notion of rank for decision trees [EH89]. Let T be a decision tree and let v be a node in T. If v is a leaf node then its rank is $\mathrm{rk}(v) = 0$. Otherwise, suppose v has children v_0 and v_1 in T. If $\mathrm{rk}(v_0) \neq \mathrm{rk}(v_1)$, define $\mathrm{rk}(v) = \max\{\mathrm{rk}(v_0), \mathrm{rk}(v_1)\}$ and $\mathrm{rk}(v) = \mathrm{rk}(v_0) + 1$, otherwise. The *rank* of the decision tree $\mathrm{rk}(T)$ is the rank of its root node. The rank $\mathrm{rk}(f)$ of a boolean function f is the minimum rank over all decision trees computing f.

In general, by a *representation class* of boolean functions we mean a set $\mathcal{R}$ of finite descriptions R for boolean functions $f\colon \{0,1\}^n \to \{0,1\}$, such that for any $R \in \mathcal{R}$ and input $x \in \{0,1\}^n$ we can evaluate $R(x) = f(x)$ in time polynomial in n and the size of R. Examples of representation classes include circuits, branching programs, formulas, decision trees, decision lists etc. Two representations R and R' are *equivalent* (denoted $R \equiv R'$) if they describe the same boolean function.

Let $f\colon \{0,1\}^n \to \{0,1\}$ be a boolean function and let $\pi \in S_n$ be a permutation. Then f^π denotes the boolean function $f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$. Similarly, we assume that we can transform any representation R of the function f into a representation R^π for f^π by replacing each input variable x_i in R by $x_{\pi(i)}$. We call two representations R_1 and R_2 *syntactically isomorphic* if $R_2 = R_1^\pi$ for some permutation π.

Let $\mathcal{R}$ and $\mathcal{R}'$ be representation classes of boolean functions. A *normal form representation* for $\mathcal{R}$ is a mapping $N\colon \mathcal{R} \to \mathcal{R}'$ such that (i) $N_R \equiv R$ for any $R \in \mathcal{R}$, (ii) $R_1 \equiv R_2$ implies $N_{R_1} = N_{R_2}$, and (iii) for each permutation π we have $N_{R^\pi} = (N_R)^\pi$. This definition is a generalization of the definition of a normal

[1]We say that a decision problem is GI-complete if it is polynomial-time equivalent to Graph Isomorphism.
[2]We say that a decision problem is GI-hard if there is a polynomial-time reduction from Graph Isomorphism to it.

form function given by Böhler et al. [BHRV04, Definition 8]. We call N_R the *normal form* of R. Since N_R only depends on the function f represented by R, N_R is also called the normal form of f and we will also denote it by N_f. Usually, only the first two conditions are required for a normal form. Following Böhler et al. we additionally require that it also fulfills the third condition that it is *permutation preserving*.

Notice that a normal form representation $N\colon \mathcal{R} \to \mathcal{R}'$ can be used to reduce the isomorphism problem for representations in $\mathcal{R}$ to the syntactical isomorphism problem for representations in $\mathcal{R}'$. More precisely, for any two representations R_1 and R_2 in $\mathcal{R}$ it holds that R_1 and R_2 represent isomorphic functions if and only if N_{R_1} and N_{R_2} are syntactically isomorphic.

A *canonical representation* for $\mathcal{R}$ is a mapping $C\colon \mathcal{R} \to \mathcal{R}$ such that (i) for any $R \in \mathcal{R}$, the function represented by C_R is isomorphic to the one described by R, and (ii) for any two representations R_1 and R_2, the functions described by R_1 and by R_2 are isomorphic if and only if $C_{R_1} = C_{R_2}$. We call C_R the *canonical form* or simply *canon* of R.

We will use the following approach to compute a canonical representation $C\colon \mathcal{R} \to \mathcal{R}$ for $\mathcal{R}$. First we find a suitable normal form representation $N\colon \mathcal{R} \to \mathcal{R}'$ for $\mathcal{R}$. Secondly, we find a transformation C' on $\mathcal{R}'$ that maps a given representation R to a syntactically isomorphic representation C'_R such that any two syntactically isomorphic representations $R_1, R_2 \in \mathcal{R}'$ are mapped to identical representations $C'_{R_1} = C'_{R_2}$. The last step is to convert a given representation $R \in \mathcal{R}'$ back into an equivalent representation T_R in $\mathcal{R}$. It is easy to verify that the concatenation $C = T \circ C' \circ N$ of these three mappings gives a canonical representation for $\mathcal{R}$.

A *literal* is either a variable x_i or a negated variable $\overline{x_i}$. We call x_i a *positive literal* and $\overline{x_i}$ a *negative literal*. For a set L of literals we denote by $\overline{L} = \{\bar{l} \mid l \in L\}$ the set of all complementary literals, where $\overline{\overline{x_i}} = x_i$. Further, we denote a positive literal x_i also by x_i^1 and a negative literal $\overline{x_i}$ also by x_i^0. Given an n-ary boolean function f, a variable x_i and a bit $b \in \{0,1\}$, the function $f[x_i \leftarrow b]$ is the n-ary boolean function that is obtained from f by setting the value of x_i to b, i.e.,

$$f[x_i \leftarrow b] : (b_1, \ldots, b_n) \mapsto f(b_1, \ldots, b_{i-1}, b, b_{i+1}, \ldots, b_n).$$

For a set L of literals we also use the notation $f[L \leftarrow b]$ for the n-ary boolean function where for any literal $x_i^1 \in L$, variable x_i is set to b and for any literal $x_i^0 \in L$, x_i is set to $1 - b$. If L contains contradictory literals x_i^0 and x_i^1, we let $f[L \leftarrow b]$ be the constant 0 function.

We proceed with a few simple observations.

Observation 2.1. *Given a decision tree T of size s, it can be checked in time $O(s)$ and in space $O(\log s)$ whether $f = 0$ or $f = 1$, where f is the boolean function represented by T.*

Proof. The first step of the algorithm is to obtain a decision tree T' that is equivalent to T and in which all variables occur only once on each path from the root to a leaf. As mentioned before, this amounts to removing repeated variables and the unreachable subtrees below them. This step can be implemented in linear time and in logspace. Afterwards, it suffices to check whether all leaves of T' are labeled with the desired constant. □

Observation 2.2. *Let T be a decision tree of size s and rank $r > 0$ for some n-ary boolean function f and let x_i^b be a literal. Then a decision tree $T[x_i^b]$ of rank at most r computing the function $f[x_i \leftarrow b]$ can be computed in time $O(s)$ and in space $O(\log s)$. Moreover, there exists some literal x_i^b such that $T[x_i^b]$ has rank at most $r - 1$.*

Proof. To obtain $T[x_i^b]$ from T, remove each inner node labeled with x_i along with the subtree rooted at its unreachable child. Then $T[x_i^b]$ computes the function $f[x_i \leftarrow b]$ and its rank is not larger than the rank of T.

To show the second part, assume that the root of T is labeled with variable x_i. Since T has rank $r > 0$, one of the children of the root must have rank at most $r - 1$. If it is the left child, then $T[x_i^0]$ has rank at most $r - 1$. Otherwise $T[x_i^1]$ has rank at most $r - 1$. □

Notice that if we eliminate several variables x_i from T by using operations of the form $T \mapsto T[x_i^b]$, the result is independent of the order in which we apply these operations as long as the conjunction of the corresponding literals is satisfiable. Hence, for a set L of non-contradicting literals we can denote the resulting tree by $T[L]$.

Using the above observations, we can minimize the rank of a given decision tree.

Theorem 2.3. *Given as input a number r and a decision tree T for a boolean function f, we can check if f has rank at most r and, if so, construct a decision tree of minimal rank for f in time $(n^r \cdot |T|)^{O(1)}$.*

Proof. Let f be the function represented by the given decision tree T. The algorithm is recursive. In the base case $r = 0$, it suffices to check if $f = 0$ or $f = 1$; this can be done by Observation 2.1. If it is, the algorithm returns a decision tree whose root is a leaf labeled by the respective constant.

In case $r > 0$, the algorithm first computes, for each literal x_i^b such that T has a node labeled by x_i, the decision tree $T[x_i^b]$ using Observation 2.2. Then it recursively checks for each tree $T[x_i^b]$ if the represented function $f[x_i \leftarrow b]$ has rank at most $r-1$. In the positive case it also obtains a decision tree $T_{i,b}$ of minimum rank $r_{i,b}$ for the function $f[x_i \leftarrow b]$, otherwise it lets $r_{i,b} = r$. If all answers are negative, the algorithm rejects. Otherwise, it computes for each variable x_i that occurs in T the value

$$r_i = \begin{cases} \max(r_{i,0}, r_{i,1}) & \text{if } r_{i,0} \neq r_{i,1} \\ r_{i,0} + 1 & \text{if } r_{i,0} = r_{i,1} \end{cases}$$

and determines $r_{\min} = \min_i r_i$ as well as the smallest index j for which $r_j = r_{\min}$. If $r_{j,b} = r$ for some $b \in \{0,1\}$, the algorithm recursively checks if the function $f[x_j \leftarrow b]$ represented by $T[x_j^b]$ has rank at most r. If the answer is negative, the algorithm rejects, otherwise it obtains a decision tree $T_{j,b}$ of rank r for $f[x_j \leftarrow b]$. Finally, the algorithm returns the decision tree T' with x_j at its root, and $T_{j,0}$ and $T_{j,1}$ as its left and right subtree, respectively.

By Observation 2.2, the algorithm never rejects if f has rank at most r. The returned decision tree T' has minimum rank, because the recursively computed subtrees $T_{j,0}$ and $T_{j,1}$ of T' have minimum rank and because the algorithm selects the root x_j of T' in such a way that the rank r_j of the resulting tree is minimal.

In order to give a bound on the running time of the algorithm, let $t(n, r, s)$ denote the worst case running time on all inputs (T, r), where T is a decision tree of size at most s whose inner nodes are labeled with n variables. Since there are exactly $2n$ recursive calls with parameters $(n-1, r-1, s-1)$ and at most one with parameters $(n-1, r, s-1)$, we have the recurrence

$$t(n, r, s) \leq 2n \cdot t(n-1, r-1, s-1) + t(n-1, r, s-1) + O(n \cdot s).$$

It is easy to verify by induction that $t(n, r, s) = O(sn^{2r})$. □

3. Canonizing decision trees for rank 1 functions

In this section we show that the isomorphism problem for boolean functions f having rank 1 is decidable in logarithmic space if f is given as a decision tree (of arbitrary rank). In fact, we will first give a polynomial-time algorithm and then a logspace algorithm for computing a canonical representation for this class. Additionally, we show that the isomorphism problem for decision trees of rank 1 is complete for logspace.

3.1. Computing a normal form for rank 1 functions

Let f be an n-ary boolean function of rank 1 given by some decision tree T, not necessarily of rank 1. For $c \in \{0,1\}$, let $L(f, c) = \{x_i^b \mid f[x_i \leftarrow b] = c\}$.

Let $f_0 = f$ and for $k \geq 1$, define $L_k(f) = L(f_{k-1}, k \bmod 2) \setminus \bigcup_{j<k} L_j(f)$ and $f_k = f_{k-1}[L_k(f) \leftarrow 0]$, and let m be the smallest index k for which f_k has rank 0.

Intuitively, the set $L_k(f)$ contains all literals x_i^b for which the assignment $x_i = b$ forces f_{k-1} to the constant function $f_{k-1}[x_i \leftarrow b] = k \bmod 2$, and f_k is obtained from f_{k-1} by assigning the opposite value to these variables. Observation 2.2 implies that for each literal x_i^b there is an index $k \leq m+1$ such that $x_i^b \in L_k(f)$. We call this index the *level* of x_i^b and denote it by $\mathrm{lv}_f(x_i^b)$.

We also notice that there might be no literals at level 1.

To compute a normal form, we transform the decision tree T into a list

$$N_T = S_1(f), \ldots, S_m(f), \{\langle 1, m+1 \bmod 2\rangle\},$$

where $S_k(f) = \{\langle x_i^b, k \bmod 2\rangle \mid x_i^b \in L_k(f)\}$. By definition, the list N_T represents the same function as the 1-decision list L_T obtained from it by replacing each set with the list of pairs contained in it.

Example 3.1. *Consider the 5-ary boolean function $f(x_1, \ldots, x_5)$ represented by the decision tree T_1 in Figure 1. Then the literal sets are $L_1(f) = \emptyset$, $L_2(f) = \{x_5^0\}$, $L_3(f) = \{x_2^0\}$, $L_4(f) = \{x_3^1, x_4^0\}$, and $L_5(f) = \{x_1^0, x_1^1, x_2^1, x_3^0, x_4^1, x_5^1\}$. The functions f_k at level k are $f_1 = f$, $f_2 = \overline{x_2} \vee \overline{x_3} \wedge x_4$, $f_3 = \overline{x_3} \wedge x_4$ and $f_4 = 1$. Thus, m gets value 4 and the normal form for T_1 is*

$$N_{T_1} = \emptyset, \{\langle x_5^0, 0\rangle\}, \{\langle x_2^0, 1\rangle\}, \{\langle x_3^1, 0\rangle, \langle x_4^0, 0\rangle\}, \{\langle 1, 1\rangle\}.$$

The next lemma proves that N_T is indeed a normal form for T.

Theorem 3.2. *The mapping $T \mapsto N_T$ defined above is a polynomial-time computable normal form representation for decision trees T that represent boolean functions of rank 1.*

Proof. To compute N_T on input T, starting with $k = 1$ iteratively compute the sets $L_k(f) = L(f_{k-1}, k \bmod 2)$ and the decision trees $T_k = T_{k-1}[\overline{L_k(f)}]$ for the function $f_k = f_{k-1}[L_k(f) \leftarrow 0]$ using Observations 2.1 and 2.2, until f_k has rank 0 (i.e., $k = m$).

We next show that N_T and T represent the same function. For a given assignment $a = a_1 \cdots a_n \in \{0,1\}^n$, let $j(a)$ be the smallest index j such that $L_j(f)$ contains a literal x_i^b satisfied by a (i.e., $b = a_i$). Then, by the way the semantics of N_T is defined it follows that $N_T(a) = j(a) \bmod 2$. On the other hand, by the definition of the sets $L_j(f)$ it follows that also $T(a) = j(a) \bmod 2$, implying that $N_T(a) = T(a)$.

Further, since N_T only depends on the function f represented by T (and not on the structure of T), equivalent decision trees $T \equiv T'$ yield identical lists $N_T = N_{T'}$.

It remains to prove the third property of a normal form. For a permutation π, let T^π be the decision tree obtained from T by replacing each label x_i in it by the label $x_{\pi(i)}$. Then we have to show that the list N_{T^π} of T^π coincides with the list $(N_T)^\pi$ that is obtained from N_T by replacing each literal x_i^b in it with the literal $x_{\pi(i)}^b$. Since T^π computes the function $g = f^\pi$ and since the normal form N_{T^π} only depends on g, it suffices to prove for any level $k = 1, \ldots, m$ that π maps $L_k(f)$ to $L_k(g)$.

In fact, for all bits $c, b \in \{0,1\}$, we have $f[x_i \leftarrow b] = c$ if and only if $g[x_{\pi(i)} \leftarrow b] = c$, so π maps $L(f, c)$ to $L(g, c)$, proving the claim for level $k = 1$. Additionally it follows that π is an isomorphism from $f_1 = f[L_1(f) \leftarrow 0]$ to $g_1 = g[L_1(g) \leftarrow 0]$. Hence, the claim follows inductively over k by using the induction hypothesis that π is an isomorphism from f_{k-1} to g_{k-1}. □

Since it is easy to syntactically canonize a given normal form N_T, we immediately get the following result.

Corollary 3.3. *Given a decision tree T that represents a boolean function of rank 1, a canonical form C_T of T can be computed in polynomial time. Thus, isomorphism for such decision trees is decidable in* P.

Proof. Define C_T as the rank 1 decision tree that is obtained from the rank 1 decision list corresponding to N_T by renaming its variables (and adding missing variables if necessary) such that the inner nodes of the longest path starting from the root are labeled by $x_1, \ldots, x_n$. □

We now show that rank 1 functions can even be canonized in logspace.

Lemma 3.4. *Let T be a decision tree that represents a function f of rank 1, let x_i^b be a literal and let x_j be a variable. Then $\mathrm{lv}_f(x_i^b) \leq \min\{\mathrm{lv}_f(x_j^0), \mathrm{lv}_f(x_j^1)\}$ if and only if $f[x_i \leftarrow b]$ does not depend on x_j. Moreover, the latter condition can be checked in logspace.*

Proof. Let $N_T = S_1(f), \dots, S_m(f), \{\langle 1, m+1 \bmod 2\rangle\}$ be the normal form of T as defined above and suppose that the literal x_i^b has level k, i.e., $\langle x_i^b, k \bmod 2\rangle \in S_k(f)$. Cutting off N_T before level k, we obtain a list $R = S_1(f), \dots, S_{k-1}(f), \{\langle 1, k \bmod 2\rangle\}$ that represents $f[x_i \leftarrow b]$. If the levels of x_j^0 and of x_j^1 are at least k, the variable x_j does not occur in R and thus $f[x_i \leftarrow b]$ does not depend on x_j. Conversely, if the level of x_j^0 or of x_j^1 is less than k, the structure of R makes it easy to find two assignments that differ only on x_j and that lead to different values of $f[x_i \leftarrow b]$.

To prove checkability in logspace, notice that $f[x_i \leftarrow b]$ does not depend on x_j if and only if the two functions $g = f[x_i \leftarrow b, x_j \leftarrow 0]$ and $h = f[x_i \leftarrow b, x_j \leftarrow 1]$ are the same. Since by Observation 2.2, it is possible to compute the decision trees $T_g = T[x_i^b, x_j^0]$ for g and $T_h = T[x_i^b, x_j^1]$ for h in logspace, it remains to note that the polynomial-time equivalence test for decision trees described in the introduction can also be implemented in logspace. □

Using this lemma, the collection of sets $L_k(f)$, $k = 1, \dots, m$ of literals having the same level can be found in logspace by determining for each literal x_i^b the set $D(f[x_i \leftarrow b])$ of variables x_j on which the function $f[x_i \leftarrow b]$ depends. Based on the sizes of these dependency sets, the obtained literal sets can also be ordered by level. If level 1 is non-empty, i.e., if there is a literal x_i^b such that $f[x_i \leftarrow b] = 1$, then these literal sets can be assigned ascending levels starting from 1, otherwise the smallest non-empty level is 2. Once the sets $L_k(f)$ are known, we can compute the canonical form of T as before.

Example 3.5. *Consider once more the 5-ary boolean function $f(x_1, \dots, x_5)$ represented by the decision tree T_1 in Figure 1. Then the dependency sets $D(f[x_i \leftarrow b])$ are given by the following table:*

i	1	2	3	4	5
$D(f[x_i \leftarrow 0])$	$\{x_2, x_3, x_4, x_5\}$	$\boldsymbol{\{x_5\}}$	$\{x_2, x_4, x_5\}$	$\boldsymbol{\{x_2, x_5\}}$	$\boldsymbol{\emptyset}$
$D(f[x_i \leftarrow 1])$	$\{x_2, x_3, x_4, x_5\}$	$\{x_3, x_4, x_5\}$	$\boldsymbol{\{x_2, x_5\}}$	$\{x_2, x_3, x_5\}$	$\{x_2, x_3, x_4\}$

The dependency sets corresponding to literals of level at most $m = 4$ are in bold print.

Theorem 3.6. *Given a decision tree T that represents a boolean function of rank 1, a canonical form C_T of T can be computed in logspace. Thus, isomorphism for such decision trees is decidable in logspace.*

3.2. Isomorphism of rank 1 decision trees is hard for logspace

All our L-hardness results are w.r.t. DLOGTIME-uniform AC^0 reductions. To show the logspace completeness, we will give a reduction from the problem ORD, which is known to be complete for L [Ete97]. The input to ORD is a directed path P (given as a set of edges) and two vertices s and t. The problem is to test if s occurs before t on P (i.e., whether t is reachable from s).

As an intermediate step, we show that the problem DIPATHCENTER is L-complete, which asks whether a given vertex u is the center of a given directed path P (which is again given as a set of its edges). The problem PATHCENTER, which asks the same question for undirected paths, is already known to be L-complete [ADKK12]. The following lemma adapts the reduction given there to the directed setting.

Lemma 3.7. DIPATHCENTER *is* L*-complete. The hardness also holds for paths of even length.*

Proof. The problem can easily be solved in logspace. To prove the hardness, we reduce from ORD using $(P, s, t) \mapsto (P', n)$ as reduction, where n is the vertex having no successor and 1 is the vertex having no predecessor in P, and where P' is defined by

$$\begin{aligned} V(P') &= V(P) \cup \big\{ i' \mid i \in V(P) \big\} \cup \big\{ \hat{s} \big\} \\ E(P') &= \big\{ (i,j) \mid (i,j) \in E(P) \wedge j \neq t \big\} \cup \big\{ (j', i') \mid (i,j) \in E(P) \wedge j \notin \{s,t\} \big\} \\ &\quad \cup \big\{ (\hat{s}, i') \mid (i, s) \in E(P) \big\} \cup \big\{ (s', \hat{s}), (t', 1), (1', t), (n, n') \big\}. \end{aligned}$$

The path P' consists of a forward and a reversed copy of P that are joined together, where the part before the first copy of t is swapped with the part after the second copy of t, and where the second copy of s is duplicated; see Fig. 2 for an illustration. If s precedes t in P (left side), then n is the center of P', but if t precedes s then n' is the center of P' (right side). □

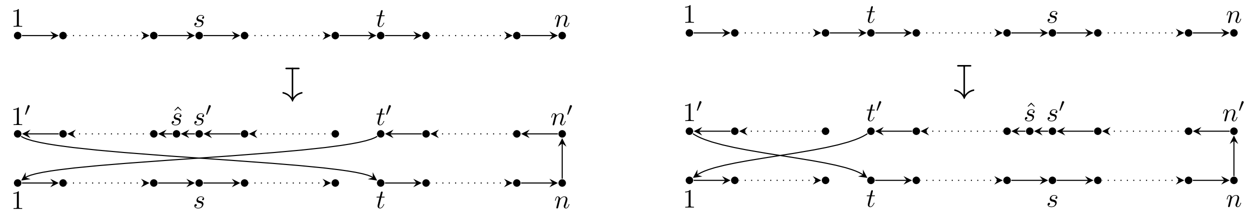

Figure 2: The reduction from Ord to DiPathCenter

Now let (P, u) be an instance of DiPathCenter. In our reduction to isomorphism of rank 1 decision trees, we construct two decision trees T and T' from P: For each $v \in V(P)$ there is a variable x_v, and both T and T' contain one internal node for each x_v. If v is the successor of v' in P, x_v becomes the right child of $x_{v'}$ in T, and $x_{v'}$ becomes the right child of x_v in T'. Let v_1 be the vertex having no predecessor, and let v_n be the vertex having no successor in P. In T, the node x_{v_1} is the root and the right child of x_{v_n} is a leaf labeled with 0. In T', these roles are reversed: x_{v_n} is the root, and the right child of x_{v_1} is a leaf labeled with 0. In both trees, the left child of x_u is a leaf labeled with 0, and the left children of all other variables are leaves labeled with 1.

Lemma 3.8. *Let T and T' be the decision trees constructed from an instance (P, u) of* DiPathCenter. *Let f and g be the functions represented by the decision trees, respectively. Then, $f \cong g$ if and only if $(P, u) \in$* DiPathCenter.

Proof. For the purposes of this proof, we identify the vertices of P with the integers $1, \dots, n$ such that the vertex i is the successor of $i-1$. To prove the lemma, it is sufficient to show that f is isomorphic to g if and only if $u = (n+1)/2$.

In the case of T, we obtain $L_1(f) = \{x_1^0, \dots, x_{u-1}^0\}$, $L_2(f) = \{x_u^0\}$, and $L_3(f) = \{x_{u+1}^0, \dots, x_n^0\}$. Similarly for T', we obtain $L_1(g) = \{x_{u+1}^0, \dots, x_n^0\}$, $L_2(g) = \{x_u^0\}$, and finally $L_3(g) = \{x_1^0, \dots, x_{u-1}^0\}$. Thus $f \cong g$ if and only if $u = (n+1)/2$. □

Combining Theorem 3.6 and Lemma 3.8, we obtain the following completeness result.

Corollary 3.9. *The isomorphism problem for rank 1 decision trees is* L*-complete.*

4. Isomorphism of decision trees for arbitrary functions

In this section we first generalize the normal form representation for decision trees computing rank 1 functions to arbitrary decision trees. Next, we exploit the structure of this normal form to give a polynomial-time reduction of the isomorphism problem for decision trees computing functions of rank at most r to the isomorphism problem for $O(r)$ rank hypergraphs. This yields a moderately exponential time algorithm for isomorphism testing of decision trees. We conclude this section by showing that graph isomorphism reduces to isomorphism of rank r decision trees, for any fixed $r \geq 2$.

4.1. Computing a normal form for decision trees

Let f be an n-ary boolean function of rank $r \geq 0$ given by some decision tree T. If $r = 0$ (which can be checked by Observation 2.1), we use the decision tree whose root is a leaf labeled by the respective constant as its normal form.

For $r > 0$, the normal form is defined as follows. Let $L(f)$ be the set of literals x_i^b such that $f[x_i \leftarrow b]$ has rank at most $r-1$. Let $f_0 = f$, and for $k \geq 1$, let $L_k(f) = L(f_{k-1}) \setminus \bigcup_{j<k} L_j(f)$ and let $f_k = f_{k-1}[L_k(f) \leftarrow 0]$. As before, the *level* $\mathrm{lv}_f(x_i^b)$ of a literal x_i^b is the index k such that $x_i^b \in L_k(f)$. By Observation 2.2, all literals occur in $L_k(f)$ for some k, so the notion of level is well-defined. Further, let m be the smallest index k for which f_k has rank at most $r-1$. Note that $m \leq n$, as for each level $k \leq m$, the set $L_k(f)$ must contain

a literal of a new variable. Indeed, $L_k(f)$ cannot be empty as $\mathrm{rk}(f_{k-1}) = r$. Further, if $L_k(f)$ contains a literal which is complementary to one in $\bigcup_{j<k} L_j(f)$, then $\mathrm{rk}(f_{k-1}) \leq r-1$. Similarly, if $L_k(f)$ contains two complementary literals, then $\mathrm{rk}(f_k) \leq r-1$.

Generalizing the normal form for rank 1 functions, which is a list of pairs that comprise of one literal and one constant each and that are grouped into sets, the normal form for f is a list N_f of pairs that comprise of one literal and one normal form of a boolean function of rank at most $r-1$ that are grouped into sets. We define $N_f = S_1(f), \dots, S_m(f), \{\langle 1, N_{f_m}\rangle\}$, where $S_k(f) = \{\langle x_i^b, N_{f_{k-1}[x_i \leftarrow b]}\rangle \mid x_i^b \in L_k(f)\}$ and where the normal forms $N_{f_{k-1}[x_i \leftarrow b]}$ and N_{f_m} for the functions $f_{k-1}[x_i \leftarrow b]$ and f_m are defined recursively based on the decreasing rank.

To evaluate N_f on a given assignment $a = a_1 \cdots a_n \in \{0,1\}^n$, find the first set in N_f that contains a pair whose first component is satisfied by a, and recursively evaluate the normal form in the second component of this pair.

Theorem 4.1. *Given a number r and a decision tree T of size s for a boolean function $f\colon \{0,1\}^n \to \{0,1\}$ of rank at most r, the normal form N_f described above can be computed in time $(sn^r)^{O(1)}$.*

Proof. We first show that the function $N\colon T \mapsto N_f$ is indeed a normal form representation for decision trees. Clearly, N_f represents the function f. Also, N_f only depends on the function f (and not on the structure of T), so equivalent decision trees T and T' receive the same normal form $N(T) = N(T') = N_f$.

To prove that N is permutation preserving, we use induction over the rank r of f. The base case is clear. If $r > 0$, let T and T' be isomorphic decision trees for n-ary boolean functions f and g of rank r and let π be an isomorphism from f to g. Then $x_i^b \in L(f)$ implies $x_{\pi(i)}^b \in L(g)$ and vice versa, meaning that π maps $L(f)$ to $L(g)$. Additionally, $(f_1)^\pi = (f[L(f) \leftarrow 0])^\pi = f^\pi[L(f)^\pi \leftarrow 0] = g[L(g) \leftarrow 0] = g_1$. Hence, an inductive argument over k gives $(f_k)^\pi = g_k$ and $L_k(f)^\pi = L_k(g)$ for $k = 1, \dots, m$. Further, as $(f_{k-1}[x_i \leftarrow b])^\pi = g_{k-1}[x_{\pi(i)} \leftarrow b]$ for $k = 1, \dots, m$ and $(f_m)^\pi = g_m$, we can use the induction hypothesis for rank at most $r-1$ functions to get the identities $(N_{f_{k-1}[x_i \leftarrow b]})^\pi = N_{g_{k-1}[x_{\pi(i)} \leftarrow b]}$ and $(N_{f_m})^\pi = N_{g_m}$, implying that $(N(f))^\pi = N(g)$.

The algorithm for computing N_f on input T is very similar to the algorithm in the proof of Theorem 2.3 and in fact has the same worst case running time. The only difference is that in each recursive step, the algorithm does not only select a single variable x_j for which $r_j = r_{\min}$ but it determines all such variables and stores all literals x_j^b for which x_j occurs in T and $r_{j,b} \leq r-1$ in the set L. (We assume that initially all n variables occur in T.) For each $x_j^b \in L$, the algorithm collects the pair $\langle x_j^b, N_{f[x_j \leftarrow b]}\rangle$ in S, where $N_{f[x_j \leftarrow b]}$ is the normal form of $T[x_j^b]$ returned by the recursive call. Finally, it computes the decision tree $T[\overline{L}]$ and, using a last recursive call, its normal form $N_{T[\overline{L}]}$. If the rank of $T[\overline{L}]$ is at most $r-1$, the algorithm returns the list $S, \{\langle 1, N_{T[\overline{L}]}\rangle\}$, otherwise it returns the list $S, N_{T[\overline{L}]}$ obtained by prepending the set S to the list $N_{T[\overline{L}]}$. $\square$

4.2. Reducing decision tree isomorphism to hypergraph isomorphism

We now describe our reduction of rank r decision tree isomorphism to rank $O(r)$ hypergraph isomorphism, where the rank of a hypergraph is the maximum size of its hyperedges.

Let T be a decision tree of size s for an n-ary boolean function f of rank r. We first compute the normal form N_f for f in time $(sn^r)^{O(1)}$, as described in Theorem 4.1. The next step is to construct a vertex-colored hypergraph $\mathcal{H}_f$ that encodes the normal form in a suitable way. The vertex set $V(n,r)$ for $\mathcal{H}_f$ is

$$V(n,r) = \{v_i^d \mid 1 \leq i \leq n, 0 \leq d \leq r\} \cup \{0,1\} \cup \{(j,b,d) \mid 0 \leq j \leq n, b \in \{0,1\}, 1 \leq d \leq r\}.$$

The hyperedge set of $\mathcal{H}_f$ is $E(n,r) \cup E(N_f)$, where $E(n,r) = \{\{v_i^d \mid 0 \leq d \leq r\} \mid 1 \leq i \leq n\}$ and where $E(N_f)$ is defined inductively below. The vertex coloring c of $\mathcal{H}_f$ is defined by $c(0) = -2$, $c(1) = -1$, $c(v_i^d) = d$ for $1 \leq i \leq n$ and $0 \leq d \leq r$, and by $c\big((j,b,d)\big) = (j,b,d)$. Each permutation $\pi \in S_n$ induces a permutation $\tilde{\pi}$ on $V(n,r)$ that maps v_i^d to $v_{\pi(i)}^d$ for $1 \leq i \leq n$ and $0 \leq d \leq r$ and is the identity on all other vertices. Note that all permutations on $V(n,r)$ that respect the coloring c and the hyperedges in $E(n,r)$ have this form.

We now turn to the definition of $E(N_f)$. In the base case $r = 0$, we define $E(N_f) = \{\{0\}\}$ if $f = 0$, and $E(N_f) = \{\{1\}\}$ if $f = 1$. For $r > 0$, let $N_f = S_1(f), \dots, S_m(f), \{\langle 1, N_{f_m}\rangle\}$ be the normal form of f

as defined above. Then for each entry $\langle x_i^b, N_h\rangle \in S_j(f)$ and for each hyperedge $e \in E(N_h)$, we include the hyperedge $e \cup \{v_i^r, (j,b,r)\}$ in $E(N_f)$. We also include all hyperedges $e \in E(N_{f_m})$ into $E(N_f)$. Notice that $E(N_f)$ is well-defined since h and f_m have rank at most $r-1$. Further, the algorithm of Theorem 4.1 can be easily modified to compute $\mathcal{H}_f$ on input T in time $(sn^r)^{O(1)}$.

Lemma 4.2. *Let $\mathcal{H}_f$ and $\mathcal{H}_g$ be the hypergraphs corresponding to the decision trees T_f and T_g. Then, T_f and T_g compute isomorphic functions f and g if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

Proof. We use induction over the rank r of f and g to prove the stronger claim that every permutation $\pi \in S_n$ is a syntactic isomorphism from N_f to N_g if and only if the mapping $\tilde{\pi}$ defined as above is an isomorphism from $\mathcal{H}_f$ to $\mathcal{H}_g$.

In the base case $r = 0$, the functions f and g are constant, implying that $E(N_f) = \{\{c_f\}\}$ and $E(N_g) = \{\{c_g\}\}$ for the respective constants $c_f, c_g \in \{0,1\}$. As the vertices 0 and 1 have different colors, the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic if and only if $f = g$. In this case, each permutation $\pi \in S_n$ maps N_f to N_g, and its induced vertex permutation $\tilde{\pi}$ maps $\mathcal{H}_f$ to $\mathcal{H}_g$.

For rank $r > 0$, let $\pi \in S_n$ be a syntactic isomorphism from $N_f = S_1(f), \dots, S_m(f), \{\langle 1, N_{f_m}\rangle\}$ to $N_g = S_1(g), \dots, S_m(g), \{\langle 1, N_{g_m}\rangle\}$. In particular, for any entry $\langle x_i^b, N_h\rangle \in S_k(f)$, there must be a corresponding entry $\langle x_{\pi(i)}^b, (N_h)^\pi\rangle \in S_k(g)$. By the induction hypothesis, $\tilde{\pi}$ maps $E(N_h)$ to $E((N_h)^\pi) = E(N_{h^\pi})$. Thus for any $e \in E(N_h)$, the hyperedge $e \cup \{v_i^r, (k,b,r)\}$ included in $E(N_f)$ is mapped by $\tilde{\pi}$ to the hyperedge $e^{\tilde{\pi}} \cup \{v_{\pi(i)}^r, (k,b,r)\}$ included in $E(N_g)$. Also, the hyperedges in $E(N_{f_m})$ are mapped to the hyperedges in $E(N_{g_m})$. Thus $\tilde{\pi}$ maps $E(N_f)$ to $E(N_g)$.

For the converse direction, suppose that for some $\pi \in S_n$, the vertex permutation $\tilde{\pi}$ maps $E(N_f)$ to $E(N_g)$. For each entry $\langle x_i^b, N_h\rangle$ in $S_k(f)$, $E(N_f)$ contains for every hyperedge $e' \in E(N_h)$ the hyperedge $e = e' \cup \{v_i^r, (k,b,r)\}$. Since the image $e^{\tilde{\pi}}$ of e contains the two vertices $v_{\pi(i)}^r$ and (k,b,r), it follows that the set $S_k(g)$ contains an entry of the form $\langle x_{\pi(i)}^b, N_{h'}\rangle$. Further, by the construction of E_f and of E_g it follows that

$$E(N_h) = E(N_f)_{i,k,b,r} \text{ and } E(N_{h'}) = E(N_g)_{\pi(i),k,b,r},$$

where for a set E of hyperedges, $E_{i,k,b,r} = \{e \in E \mid v_i^r \in e, (k,b,r) \in e\}$. Hence, $\tilde{\pi}$ maps $E(N_h)$ to $E(N_{h'})$. Also, $\tilde{\pi}$ must map the remaining hyperedges of $E(N_f)$ to those of $E(N_g)$ and thus maps $E(N_{f_m})$ to $E(N_{g_m})$. By the induction hypothesis we get $N_{h'} = (N_h)^\pi$ and $N_{g_m} = (N_{f_m})^\pi$, implying that $N_g = (N_f)^\pi$. $\square$

According to the construction, the hypergraph $\mathcal{H}_f$ corresponding to the rank r function f has $O(nr)$ vertices and rank $2r+1$.

Theorem 4.3. *Let f and g be n-ary boolean functions of rank at most r given by decision trees T_f and T_g of size at most s. There is an algorithm running in time $(sn^r)^{O(1)}$ that outputs two hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ of rank at most $2r+1$ having $O(nr)$ vertices such that f and g are isomorphic if and only if the hypergraphs $\mathcal{H}_f$ and $\mathcal{H}_g$ are isomorphic.*

By combining this with the isomorphism algorithm for hypergraphs of rank R on N vertices of Babai and Codenotti [BC08], which takes $2^{R^2\sqrt{N}(\log N)^{O(1)}}$ time, we get an $(sn^r)^{O(1)} + 2^{\sqrt{nr^5}(\log n)^{O(1)}}$ time isomorphism algorithm for n-ary boolean functions of rank r that are given by decision trees of size s. Since any decision tree of size s has rank at most $O(\log s)$, this gives the following corollary.

Corollary 4.4. *Given two n-ary boolean functions f and g as decision trees of size s, there is a $2^{\sqrt{n}(\log s)^{O(1)}}$ time algorithm to check if $f \cong g$.*

We note that the canonization problem for bounded rank decision lists also polynomial-time reduces to the canonization problem for bounded rank hypergraphs. However, to the best of our knowledge, not even a $2^{O(n)}$ algorithm is known for computing canonical forms for hypergraphs of rank 3.

4.3. Isomorphism of decision trees is GI*-hard*

We next show that isomorphism testing even for rank 2 decision trees is GI-hard.

Let $G = (V, E)$ be a graph with $V = \{v_1, v_2 \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. We encode G as a boolean function f_G on the variable set $V \cup E$ as follows: $f_G(v_1, \ldots, v_n, e_1, \ldots, e_m) = 1$ if and only if exactly three variables e_i, v_j, v_k are 1, all remaining variables are 0, and $e_i = \{v_j, v_k\} \in E$. Here the boolean variables v_i and e_j correspond, by abuse of notation, to elements of $V \cup E$. We can write f_G as $f_G = \bigvee_{e=\{u,v\}\in E}\big(e \wedge (\bigwedge_{e'\neq e} \overline{e'}) \wedge u \wedge v \wedge (\bigwedge_{w\neq u,v} \overline{w})\big)$.

Lemma 4.5. *For any graph $G = (V, E)$, the function f_G is of rank 2 and can be represented by a rank 2 decision tree of size $O\big(|E|^2|V|\big)$.*

Proof. Note that if any edge variable e is set to 1 where $e = \{u, v\}$, all the terms in f_G disappear, except the one where the variable appears un-negated. Thus, $f_G[e \leftarrow 1] = \bigwedge_{e'\neq e} \overline{e'} \wedge u \wedge v \wedge \bigwedge_{w\neq u,v} \overline{w}$. Since $f_G[e \leftarrow 1]$ is a conjunction of literals, it is a rank 1 function. Since f_G is zero if all the edge variables are set to 0, this proves that f_G is a rank 2 function. □

Theorem 4.6. *Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs in which all vertices have at least two neighbors, and let f_G and f_H be the functions as defined above. Then, $G \cong H$ if and only if $f_G \cong f_H$.*

Proof. The function f_G encodes the graph G in the sense that for an assignment a to the variables, $f_G(a) = 1$ exactly if a encodes an edge $e = \{u, v\} \in E_G$, i.e., $a_e = a_u = a_v = 1$ and $a_x = 0$ for all $x \in (V_G \cup E_G)\setminus\{e, u, v\}$.

Any isomorphism π from G to H can be extended to map each edge $e = \{u, v\} \in E_G$ to $\pi(e) = \{\pi(u), \pi(v)\}$. Then π sends the satisfying assignments of f_G to the satisfying assignments of f_H, implying that $f_G \cong f_H$.

Conversely, if π is an isomorphism from f_G to f_H, it induces a bijection between the satisfying assignments of the two functions. As a variable is an edge variable if and only if it occurs in only one satisfying assignment, π maps edge variables to edge variables and vertex variables to vertex variables. It follows that π restricted to V_G is an isomorphism from G to H. □

As any pair of graphs can be modified to meet the degree requirement of Theorem 4.6 by adding two universal vertices to each graph, we have the following corollary.

Corollary 4.7. GI $\leq^p_m$ DT-Iso.

5. Isomorphism of decision lists

In this section, we consider $\mathcal{C}$-DL isomorphism as defined in Section 1. We first observe that satisfiability of $\mathcal{C}$-DLs is related to the Constraint Satisfaction Problem (CSP) where the constraints come from the class $\mathcal{C}$.

Definition 5.1. *A* constraint *of arity k is a boolean function $C\colon \{0,1\}^k \to \{0,1\}$. For a constraint C of arity k and a sequence of variables $x_{i_1}, \ldots, x_{i_k}$ with $1 \leq i_1, \ldots, x_{i_k} \leq n$, the corresponding n*-ary constraint application *is the boolean function $f(x_1, \ldots, x_n) = C(x_{i_1}, \ldots, x_{i_k})$. For a finite class $\mathcal{C}$ of constraints, a* $\mathcal{C}$-CSP instance *I is a set of n-ary applications of constraints in $\mathcal{C}$ and represents the conjunction of these constraint applications. A constraint C is called*

- 0-valid*, if $C(0, \ldots, 0) = 1$,*
- 1-valid*, if $C(1, \ldots, 1) = 1$,*
- Horn*, if it is a Horn-CNF, i.e., each clause has at most one positive literal,*
- anti-Horn*, if it is an anti-Horn-CNF, i.e., each clause has at most one negative literal,*
- bijunctive*, if it is a 2-CNF,*
- affine*, if it is a conjunction of parities of literals, and*
- 2-affine*, if it is a conjunction of parities where each parity consists of at most 2 literals.*

A class $\mathcal{C}$ of constraints is called 0-valid, 1-valid, Horn, anti-Horn, bijunctive, affine, or 2-affine, if every constraint in it has the respective property. $\mathcal{C}$ is called Schaefer *if it is Horn, anti-Horn, bijunctive or affine.*

Schaefer proved the following dichotomy result regarding the satisfiability of CSP instances.

Theorem 5.2 ([Sch78])**.** *Let $\mathcal{C}$ be a class of constraints. The satisfiability problem for $\mathcal{C}$-CSP instances is in* P *if $\mathcal{C}$ is 0-valid, 1-valid or Schaefer, and* NP*-complete otherwise.*

Böhler et al. considered the isomorphism problem for CSP instances, which asks whether the boolean functions computed by two given CSP instances are isomorphic.

Theorem 5.3 ([BHRV04])**.** *Let $\mathcal{C}$ be a class of constraints. The isomorphism problem for $\mathcal{C}$-CSP instances is in* P *if $\mathcal{C}$ is 2-affine,* GI*-complete if $\mathcal{C}$ is Schaefer but not 2-affine, and both* coNP*-hard and* GI*-hard otherwise.*

Our main result on decision lists is the following extension of this trichotomy result to $\mathcal{C}$-DLs. Most of this section is devoted to its proof.

Theorem 5.4. *Let $\mathcal{C}$ be a class of constraints. The isomorphism problem for $\mathcal{C}$-DLs is in* L *if $\overline{\mathcal{C}}$ is 2-affine,* GI*-complete if $\overline{\mathcal{C}}$ is Schaefer but not 2-affine, and both* coNP*-hard and* GI*-hard otherwise.*

Recall that in a $\mathcal{C}$-DL $L = \langle f_1, c_1\rangle, \ldots, \langle f_m, c_m\rangle$, for each f_i there are a function $C \in \mathcal{C}$ and indices $i_1, \ldots, i_k$ such that $f_i(x_1, \ldots, x_n) = C(x_{i_1}, \ldots, x_{i_k})$; cf. Definition 1.2. Thus f_i can be viewed as an application of the constraint C. For a class of constraints $\mathcal{C}$, we define its complement as $\overline{\mathcal{C}} = \{\neg C \mid C \in \mathcal{C}\}$, and observe the following.

Lemma 5.5. *For any $\mathcal{C}$-CSP instance I, there is an equivalent $\overline{\mathcal{C}}$-DL L_I.*

Proof. Given a $\mathcal{C}$-CSP instance $I = \big\{C_j(x_{i(j,1)}, \ldots, x_{i(j,k_j)}) \mid j = 1, \ldots, m\big\}$, we define

$$L_I = \langle \neg C_1(x_{i(1,1)}, \ldots, x_{i(1,k_1)}), 0\rangle, \ldots, \langle \neg C_m(x_{i(m,1)}, \ldots, x_{i(m,k_m)}), 0\rangle, \langle 1, 1\rangle\ .$$

The decision list L_I represents the function $\bigwedge_{j=1}^{m} C_j(x_{i(j,1)}, \ldots, x_{i(j,k_j)})$ and thus is equivalent to I. ☐

Combining this lemma with Theorem 5.3 gives the lower bounds featuring in our trichotomy for $\mathcal{C}$-DLs. It remains to show the upper bounds for the cases where $\overline{\mathcal{C}}$ is Schaefer, i.e., where $\mathcal{C}$ consists of either (i) disjunctions of conjunctions of literals of which at most one is negative, (ii) disjunctions of conjunctions of literals of which at most one is positive, (iii) disjunctions of parities of literals or (iv) disjunctions of conjunctions of two literals.

In Section 5.2, we show that in all these cases, the isomorphism problem for $\mathcal{C}$-DLs is reducible to GI. Moreover, in Section 5.1 we show that the isomorphism problem is even decidable in logspace when $\overline{\mathcal{C}}$ is 2-affine, i.e., $\mathcal{C}$ consists of disjunctions of parities of at most two literals. Further, we show that $\mathcal{C}$-DL-Iso is L-hard, unless $\mathcal{C}$ contains only constant functions. Finally, in Section 5.3 we consider the general case and show for any class $\mathcal{C}$ of constraints that $\mathcal{C}$-DL-Iso can be solved in polynomial time by asking parallel queries to an NP oracle.

5.1. Canonizing decision lists in the 2-affine case

A $2\oplus$-*condition* is a formula of the form x_i, $x_i \oplus 1$, $x_i \oplus x_j$, or $x_i \oplus x_j \oplus 1$. A $2\oplus$-DL L is a list of pairs $\langle D_1, c_1\rangle, \ldots, \langle D_m, c_m\rangle$ where $c_i \in \{0, 1\}$ for $i = 1, \ldots, m$ and each D_i is a disjunction of $2\oplus$-conditions. We assume that each D_i is represented as a set containing all the $2\oplus$-conditions in the disjunction. We say that a pair $\langle D_i, c_i\rangle$ *fires* on an assignment a, if i is the least index such that $p(a) = 1$ for some $p \in D_i$.

We next describe a normal form representation for $2\oplus$-DLs. The construction is very similar to the one for functions of rank 1 which correspond to 1-DLs. We then exploit the structure of the normal form to give a logspace algorithm for the $2\oplus$-DL isomorphism problem.

The normal form N_L for a $2\oplus$-DL L computing an n-ary boolean function f is a $2\oplus$-DL of the form

$$N_L = \langle P_1(f), 1\rangle, \langle P_2(f), 0\rangle, \ldots, \langle P_m(f), m \bmod 2\rangle, \langle 1, m + 1 \bmod 2\rangle.$$

The idea behind the definition of N_L is to include in each level k as many $2\oplus$-conditions as possible, even if some of them are redundant, as this choice does not depend on the order of the variables.

More precisely, for two n-ary boolean functions f, g and a bit $c \in \{0,1\}$, let $P(f,g,c)$ denote the set of all $2\oplus$-conditions p such that f evaluates to c under all assignments that satisfy $p \wedge g$. Let $g_0 = 1$ and for $k \geq 1$, let $P_k(f) = P(f, g_{k-1}, k \bmod 2) \setminus \bigcup_{j<k} P_j(f)$ and let g_k be the function $g_{k-1} \wedge \bigwedge_{p \in P_k(f)} \neg p$. Further let m be the largest index k for which g_k is not the constant 0 function. The *level* $\mathrm{lv}_f(p)$ of a $2\oplus$-condition p is the smallest index k such that $p \in P_k(f)$.

Example 5.6. *The* $2\oplus$*-DL*

$$L = \langle\{x_1 \oplus x_2 \oplus 1, x_2 \oplus x_3, x_5 \oplus x_6 \oplus 1\}, 1\rangle, \langle\{x_1 \oplus 1, x_4 \oplus x_5\}, 0\rangle, \langle\{x_1 \oplus x_6 \oplus 1\}, 1\rangle, \langle 1, 0\rangle$$

has the $2\oplus$*-DL* $N_L = \langle P_1, 1\rangle, \langle P_2, 0\rangle, \langle P_3, 1\rangle, \langle 1, 0\rangle$ *as its normal form, where*

$$\begin{aligned} P_1 &= \{x_1 \oplus x_2 \oplus 1, x_1 \oplus x_3 \oplus 1, x_2 \oplus x_3, x_5 \oplus x_6 \oplus 1\}, \\ P_2 &= \{x_1 \oplus 1, x_2, x_3, x_4 \oplus x_5, x_4 \oplus x_6 \oplus 1\} \text{ and} \\ P_3 &= \{x_1 \oplus x_4, x_1 \oplus x_5, x_1 \oplus x_6 \oplus 1, x_2 \oplus x_4 \oplus 1, x_2 \oplus x_5 \oplus 1, \\ &\qquad x_2 \oplus x_6, x_3 \oplus x_4 \oplus 1, x_3 \oplus x_5 \oplus 1, x_3 \oplus x_6, x_4 \oplus 1, x_5 \oplus 1, x_6\}. \end{aligned}$$

Moreover, $g_0 = 1$*,* $g_1 = (x_1 \oplus x_2) \wedge (x_1 \oplus x_3) \wedge (x_5 \oplus x_6)$*,* $g_2 = x_1 \wedge \overline{x_2} \wedge \overline{x_3} \wedge (x_4 \oplus x_6) \wedge (x_5 \oplus x_6)$*,* $g_3 = x_1 \wedge \overline{x_2} \wedge \overline{x_3} \wedge x_4 \wedge x_5 \wedge \overline{x_6}$ *and* $g_4 = 0$ *as the set* P_4 *contains all* $2\oplus$*-conditions over the variable set* $\{x_1, \ldots, x_6\}$ *that are missing in* $P_1 \cup P_2 \cup P_3$*.*

The following lemma summarizes our normal form construction.

Lemma 5.7. *The function* $L \mapsto N_L$ *is a normal form representation for* $2\oplus$*-decision lists.*

Proof. Let L be a $2\oplus$-DL computing some n-ary boolean function f. To show that L and N_L represent the same function, we assume that all pairs in L can fire and split L into subsequences $C_1, \ldots, C_\ell$ so that all pairs $\langle D_i, c_i\rangle$ in C_k satisfy $c_i = k \bmod 2$ and all C_k (except possibly C_1) are non-empty. For $k = 1, \ldots, \ell$, it follows by induction over k that the disjunction of the conditions D_i with $\langle D_i, k \bmod 2\rangle$ in C_k is equivalent to the disjunction of the $2\oplus$-conditions p with $p \in P_k(f)$ under all assignments for which no pair in $C_1, \ldots, C_{k-1}$ fires. This also proves that $\ell = m + 1$.

Further, since the sets $P_k(f)$ depend only on f, equivalent $2\oplus$-DLs will lead to the same sets. To show that N_L is permutation preserving, suppose that π is an isomorphism between two boolean functions f and g. Then a $2\oplus$-conditions p is in the set $P_k(f)$ if and only if p^π is in the set $P_k(g)$. Thus $(N_f)^\pi = N_{f^\pi} = N_g$. □

Next, we describe a logspace algorithm for computing N_L. For this we need a generalized notion of restriction of boolean functions represented by decision lists: For a decision list $L = \langle f_1, c_1\rangle, \ldots, \langle f_\ell, c_\ell\rangle$ and a function g, let $L[g] = \langle f_1, c_1\rangle, \ldots, \langle f_{i-1}, c_{i-1}\rangle\langle 1, c_i\rangle$, where i is the smallest index for which $g \wedge \bigwedge_{j=1}^{i-1} \neg f_j$ implies f_i. It is clear that $L[g]$ yields the same value as L on all assignments a for which $g(a) = 1$. Moreover, an inductive argument along the same lines as in the proof of Lemma 5.7 shows that $L[g]$ and $N_L[g]$ compute the same function.

Lemma 5.8. *Let* L *be a* $2\oplus$*-DL that represents a function* f *and let* p *and* p' *be* $2\oplus$*-conditions. Then* $\mathrm{lv}_f(p) \leq \min\{\mathrm{lv}_f(p'), \mathrm{lv}_f(\neg p')\}$ *if and only if* $L[p, p']$ *and* $L[p, \neg p']$ *are equivalent.*

Proof. Let $N_L = \langle P_1, 1\rangle, \ldots, \langle P_m, m \bmod 2\rangle, \langle 1, m+1 \bmod 2\rangle$ be the normal form of L. As argued above, the decision list $L[p]$ is equivalent to $N_L[p] = \langle P_1(f), 1\rangle, \ldots, \langle P_{i-1}(f), i-1 \bmod 2\rangle, \langle 1, i \bmod 2\rangle$, where $i = \mathrm{lv}_f(p)$. If $\mathrm{lv}_f(p) \leq \min\{\mathrm{lv}_f(p'), \mathrm{lv}_f(\neg p')\}$, the constraint applications p' and $\neg p'$ do not occur in $N_L[p]$. By construction of N_L, this implies $N_L[p, p'] = N_L[p, \neg p'] = N_L[p]$. Thus $L[p, p']$ and $L[p, \neg p']$ are equivalent.

In case $j = \min\{\mathrm{lv}_f(p'), \mathrm{lv}_f(\neg p')\} < \mathrm{lv}_f(p)$, let a be an assignment such that $\langle P_{j+1}(f), j+1 \bmod 2\rangle$ fires upon evaluation of $N_L[p]$. As $\max\{\mathrm{lv}_f(p'), \mathrm{lv}_f(\neg p')\} = m+1$, we have $N_L[p, p'](a) \neq N_L[p, \neg p'](a)$. Thus $L[p, p']$ and $L[p, \neg p']$ are not equivalent. □

Lemma 5.9. *Given a 2⊕-DL L, its normal form N_L can be computed in logspace.*

Proof. The satisfiability of a conjunction of 2⊕-conditions of size at most 2 can be checked in logspace by constructing the equivalence graph of literals and checking that no variable occurs together with its negation in the same connected component. The latter is possible in logspace using Reingold's algorithm [Rei08].

The decision list $L[p] = \langle D_1, c_1\rangle, \dots, \langle D_{i-1}, c_{i-1}\rangle\langle 1, c_i\rangle$ for a given 2⊕-DL $L = \langle D_1, c_1\rangle, \dots, \langle D_\ell, c_\ell\rangle$ and a 2⊕-condition p can be computed in logspace by finding the smallest i such that $p \wedge \bigwedge_{j=1}^{i} \neg D_i$ is not satisfiable.

Two 2⊕-DLs $L = \langle D_1, c_1\rangle, \dots, \langle D_\ell, c_\ell\rangle$ and $L' = \langle D'_1, c'_1\rangle, \dots, \langle D'_{\ell'}, c'_{\ell'}\rangle$ are not equivalent if and only if there are entries $\langle D_i, c_i\rangle$ of L and $\langle D'_j, c'_j\rangle$ of L' such that $c_i \neq c'_j$ and for some $p_i \in D_i$ and $p'_j \in D'_j$ the following conjunction of 2⊕-conditions is satisfiable:

$$p_i \wedge p'_j \wedge \left(\bigwedge_{k=1}^{i-1} \neg D_k\right) \wedge \left(\bigwedge_{k=1}^{j-1} \neg D'_k\right)$$

Thus the condition given by Lemma 5.8 can be evaluated in logspace. This allows to partition the set of all 2⊕-conditions by level and to find the linear order between the levels. It remains to check whether P_1 is empty. This can be done by checking if $L[p] = 1$ for some 2⊕-condition p in the smallest level. □

Let L be a 2⊕-DL and let $N_L = \langle P_1, 1\rangle, \dots, \langle P_m, m \bmod 2\rangle, \langle 1, m+1 \bmod 2\rangle$ be its normal form. We will encode the structure of N_L as a graph G_L of tree distance width 2 such that N_L and $N_{L'}$ are syntactically isomorphic if and only if G_L and $G_{L'}$ are isomorphic. Das et al. proved that this graph class can be canonized in logspace [DTW12].

We can rewrite a 2⊕-condition as an equation over literals and constants. For example, the 2⊕-condition $x_1 \oplus x_2$ can be rewritten as the equations $x_1^0 = x_2^1$ or $x_1^1 = x_2^0$ and the 2⊕-condition $x_1 \oplus 1$ can be rewritten as $x_1^0 = 1$ or $x_1^1 = 0$. The idea is to represent N_L as a sequence of equation sets representing the conjunctions of 2⊕-conditions $\bigwedge_{i=1}^{k} \neg P_i$ for $k = 0, \dots, m+1$. More precisely, we call two literals or constants $l, l' \in X(n) = \{0, 1\} \cup \{x_i^b \mid i \in \{1, \dots, n\}, b \in \{0, 1\}\}$ *equivalent after level* k (denoted $l \equiv_k l'$) if $l = l'$ or if the 2⊕-condition corresponding to $l \oplus l'$ is contained in $P_1 \cup \dots \cup P_k$.

Lemma 5.10. *The relations $\equiv_k$ are transitive on $X(n)$ for $k = 0, \dots, m+1$.*

Proof. Assume that $x_i \equiv_k x_j$ and $x_j \equiv_k x_h$ (the cases that some of these three variables are negated or replaced by a constant are handled in exactly the same way). Then $P_1 \cup \dots \cup P_k$ contains the two 2⊕-conditions $p = x_i \oplus x_j$ and $p' = x_j \oplus x_h$.

If p and p' belong to the same set P_r, then $p'' = x_i \oplus x_h$ is in $P_1 \cup \dots \cup P_r$, since any assignment a that satisfies $g_{r-1} \wedge p''$, either satisfies $g_{r-1} \wedge p$ or $g_{r-1} \wedge p'$, implying that $f(a) = r \bmod 2$.

If p and p' belong to different sets P_r and P_s with $r < s$, then any assignment that satisfies g_{s-1} assigns to x_i and x_j the same value, implying that $g_{s-1} \wedge p'$ and $g_{s-1} \wedge p''$ describe the same function. Hence, the level of p'' is at most the level of p', implying that $p'' \in P_1 \cup \dots \cup P_s$. □

The relation $\equiv_k$ partitions the set $X(n)$ into the equivalence classes $[l]_k = \{y \in X(n) \mid l \equiv_k y\}$, $l \in X(n)$. The *complementary* equivalence class of $[l]_k$ is $[\bar{l}]_k = \{y \in X(n) \mid \bar{l} \equiv_k y\}$.

Example 5.11. *Consider* $L = \langle\{x_1 \oplus x_2 \oplus 1, x_2 \oplus x_3, x_5 \oplus x_6 \oplus 1\}, 1\rangle, \langle\{x_1 \oplus 1, x_4 \oplus x_5\}, 0\rangle, \langle\{x_1 \oplus x_6 \oplus 1\}, 1\rangle, \langle 1, 0\rangle$. *The equivalence classes of* $\equiv_1$ *are* $\{0\}$, $\{1\}$, $\{x_1, \overline{x_2}, \overline{x_3}\}$, $\{\overline{x_1}, x_2, x_3\}$, $\{x_4\}$, $\{\overline{x_4}\}$, $\{x_5, \overline{x_6}\}$, *and* $\{\overline{x_5}, x_6\}$. *The classes* $[0]_1$ *and* $[1]_1$ *are complementary to each other, as are* $[x_1]_1$ *and* $[x_2]_1$, *and likewise* $[x_4]_1$ *and* $[\overline{x_4}]_1$ *as well as* $[x_5]_1$ *and* $[x_6]_1$.

The equivalence classes of $\equiv_2$ *are* $\{0, x_1, \overline{x_2}, \overline{x_3}\}$, $\{1, \overline{x_1}, x_2, x_3\}$, $\{x_4, x_5, \overline{x_6}\}$, *and* $\{\overline{x_4}, \overline{x_5}, x_6\}$. *The complementary pairs are* $\{[0]_2, [1]_2\}$ *and* $\{[x_4]_2, [x_6]_2\}$.

The equivalence classes of $\equiv_3$ *are* $\{0, x_1, \overline{x_2}, \overline{x_3}, x_4, x_5, \overline{x_6}\}$ *and* $\{1, \overline{x_1}, x_2, x_3, \overline{x_4}, \overline{x_5}, x_6\}$. *They are complementary to each other.*

For $\equiv_4$ *there is only one equivalence class* $X(6)$, *i.e., it contains all literals and constants over* $\{x_1, \dots, x_6\}$ *and is complementary to itself.*

Given a normal form $N_L = \langle P_1, 1\rangle, \ldots, \langle P_m, m \bmod 2\rangle, \langle 1, m+1 \bmod 2\rangle$, we encode the sizes and inclusion structure of these equivalence classes in the graph G_L. For each $k \in \{0, \ldots, m+1\}$ and equivalence class $[l]_k$ of $\equiv_k$, the graph G_L contains a vertex $([l]_k, k)$. By definition, the equivalence classes of $\equiv_0$ are singletons, the equivalence classes of $\equiv_k$ are a refinement of the equivalence classes of $\equiv_{k+1}$ for $k = 1, \ldots, m+1$, and $\equiv_{m+1}$ has just one equivalence class. Thus the edges $\big\{\{([l]_k, k), ([l]_{k-1}, k-1)\} \bigm| l \in X(n), 1 \le k \le m\big\}$ form a tree and we add them all to G_L. Additionally, G_L contains an edge between each pair of complementary equivalence classes. The vertices $([0]_0, 0)$, $([1]_0, 0)$, and $([0]_m, m)$ receive the colors `0`, `1`, and `root` respectively, and for each variable x_i, the vertex $([x_i]_0, 0)$ is colored with `var`. See Figure 3 for an example.

Lemma 5.12. *If two $2\oplus$-DLs L and L' represent isomorphic functions f and f', then the graphs G_L and $G_{L'}$ are isomorphic.*

Proof. Let $N_L = \langle P_1, 1\rangle, \ldots, \langle P_m, m \bmod 2\rangle, \langle 1, m+1 \bmod 2\rangle$ and $N_{L'} = \langle P'_1, 1\rangle, \ldots, \langle P'_m, m \bmod 2\rangle, \langle 1, m+1 \bmod 2\rangle$ be the normal forms of L and L', and let $\equiv_k$ and $\equiv'_k$ denote equivalence after level k in N_L and $N_{L'}$, respectively. By Lemma 5.7, any isomorphism π from f to f' is also a syntactic isomorphism from N_L to $N_{L'}$. In particular, for any $l, l' \in X(n)$ and $k \in \{0, \ldots, m+1\}$, we have $l \equiv_k l'$ if and only if $\pi(l) \equiv'_k \pi(l')$. This implies that π induces an isomorphism from G_L to $G_{L'}$. $\square$

A tree distance decomposition for a graph $G = (V, E)$ is a rooted tree $T = (X, F)$ whose nodes X (which are called *bags*) form a partition of V, such that

- for every edge $\{u, v\} \in E$, the vertices u and v are either in the same bag of T or in adjacent bags of T, and
- for each vertex $v \in V$, the minimum distance in G from v to a vertex in the root bag of T equals the distance in T of the bag containing v to the root bag.

The *width* of T is the maximum size of its bags.

Using one bag for each pair of complementary equivalence classes results in a tree distance decomposition of width 2 for G_L, which has edges between $\{([l]_k, k), ([\bar{l}]_k, k)\}$ and $\{([l]_{k-1}, k-1), ([\bar{l}]_{k-1}, k-1)\}$ for each $l \in X(n)$ and $k \in \{1, \ldots, m+1\}$. Thus we can use the logspace algorithm of Das et al. [DTW12] to obtain a canon C_L of G_L.

The next lemma shows that a syntactically isomorphic copy of N_L can be reconstructed from the canon C_L of G_L.

Lemma 5.13. *Let L be a $2\oplus$-DL, and let G_L be the graph that encodes the structure of its normal form N_L as described above. Given an isomorphic copy G'_L of G_L, a syntactically isomorphic copy $S(G'_L)$ of N_L can be computed in logspace.*

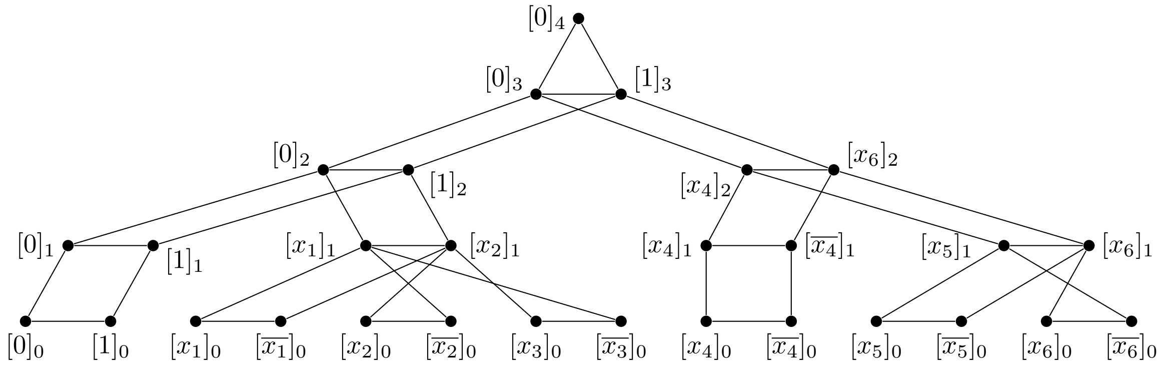

Figure 3: The graph representation G_L of the normal form N_L considered in Example 5.11. The levels are omitted from the vertex names as they are clear from the graph structure.

Proof. To construct $S(G'_L) = \langle P'_1, 1\rangle, \ldots, \langle P'_m, m \bmod 2\rangle, \langle 1, m+1 \bmod 2\rangle$ from G'_L, let n be the number of vertices with color `var` in G'_L and let m be the distance of the vertex colored `root` to any color `var` vertex in G'_L. Call a vertex v of G'_L a *level k vertex* if it has distance $m-k$ to the vertex colored `root`. This can be checked in logspace [DTW12]. Let σ be the bijection from the color `var` vertices in G'_L to $\{1, \ldots, n\}$ that maps the ith color `var` vertex in G'_L to i. Because of the colors `0` and `1` and the edges between the vertices of complementary equivalence classes, σ induces a bijection σ' between the level m vertices of G'_L and $X(n) = \{0,1\} \cup \{x_i^b \mid i \in \{1, \ldots, n\}, b \in \{0,1\}\}$. Let T'_L be the tree rooted at the vertex colored `root` that is obtained from G'_L by dropping all edges between vertices of the same level. For each $k \in \{1, \ldots, m\}$, the set P'_k consists of all 2$\oplus$-conditions $\sigma'(v) \oplus \sigma'(w)$ such that v and w are level m vertices and there is a level k vertex u in G'_L such that v and w are in subtrees of different children of u in T'_L.

We claim that for any isomorphism π from G_L to G'_L, the mapping $\tilde{\pi}(i) = j$, where $\sigma'\big(\pi(([x_i]_0, 0))\big) = x_j$, is a syntactic isomorphism from N_L to $S(G'_L)$. This holds because for two literals or constants l and l', the set P_k contains the 2$\oplus$-condition $l \oplus l'$ if and only if $[l]_k = [l']_k$ and $[l]_{k-1} \neq [l']_{k-1}$, i.e., if and only if $([l]_0, 0)$ and $([l']_0, 0)$ are in the subtrees of different children of $([l]_k, k)$ in G_L. ☐

Theorem 5.14. *Let $\overline{\mathcal{C}}$ be a 2-affine class of constraints. Then a canonical representation for $\mathcal{C}$-DLs can be computed in logspace.*

Proof. Given a 2$\oplus$-DL L, compute its normal form N_L (by Lemma 5.9), construct its graph representation G_L (which is easily possible in logspace), obtain a canon C_L for the latter using the algorithm of Das et al. [DTW12], and return the isomorphic copy $S(C_L)$ of N_L reconstructed from C_L using the algorithm of Lemma 5.13 as the canonical form of L.

By Lemmas 5.7 and 5.13, $S(C_L)$ and L represent isomorphic functions. By Lemma 5.12, two 2$\oplus$-DLs L and L' that represent isomorphic input functions have isomorphic graph representations G_L and $G_{L'}$, implying that $C_L = C_{L'}$. As the canonical form $S(C_L)$ only depends on C_L, we have $S(C_L) = S(C_{L'})$ as required. ☐

We conclude this subsection by showing that deciding isomorphism of $\mathcal{C}$-CSP instances is L-hard, unless $\mathcal{C}$ contains only constant functions.

Theorem 5.15. *Let $\mathcal{C}$ be a class of constraints that contains a non-constant function. Then* $\mathcal{C}$-DL-ISO *is* L*-hard.*

Proof. As Graph Isomorphism is L-hard even for trees [JKMT03], by Theorem 5.3 and Lemma 5.5 it suffices to consider the case that $\overline{\mathcal{C}}$ is 2-affine. Note that if $\neg C$ is a non-constant and 2-affine constraint, then there is a constraint application $f(x_1, \ldots, x_n) = C(x_{i_1}, \ldots, x_{i_k})$ that either corresponds to a single literal $f(x_1, \ldots, x_n) = x_i^b$ or to the parity $f(x_1, \ldots, x_n) = x_i \oplus x_j \oplus b$ of two literals. If the application is a single negative literal, then it is easy to encode the rank 1 decision trees T_1 and T_2 from the proof of Lemma 3.8 as $\mathcal{C}$-DLs

$$L_1 = \langle \overline{x_1}, 1\rangle, \ldots, \langle \overline{x_{u-1}}, 1\rangle, \langle \overline{x_u}, 0\rangle, \langle \overline{x_{u+1}}, 1\rangle, \ldots, \langle \overline{x_n}, 1\rangle, \langle 1, 0\rangle \text{ and}$$
$$L_2 = \langle \overline{x_n}, 1\rangle, \ldots, \langle \overline{x_{u+1}}, 1\rangle, \langle \overline{x_u}, 0\rangle, \langle \overline{x_{u-1}}, 1\rangle, \ldots, \langle \overline{x_1}, 1\rangle, \langle 1, 0\rangle,$$

resulting in a reduction from the L-hard problem DIPATHCENTER to $\mathcal{C}$-DL-ISO. If the application is a single variable, replacing the negative literals with the corresponding positive ones encodes the dual functions $f_i(\overline{x_1}, \ldots, \overline{x_n})$ as $\mathcal{C}$-DLs, implying L-hardness also for this case.

If the application is the parity of two variables, we can add an additional variable x_{n+1} and include it in each condition:

$$L'_1 = \langle x_1 \oplus x_{n+1}, 1\rangle, \ldots, \langle x_{u-1} \oplus x_{n+1}, 1\rangle, \langle x_u \oplus x_{n+1}, 0\rangle, \langle x_{u+1} \oplus x_{n+1}, 1\rangle, \ldots, \langle x_n \oplus x_{n+1}, 1\rangle, \langle 1, 0\rangle \text{ and}$$
$$L'_2 = \langle x_n \oplus x_{n+1}, 1\rangle, \ldots, \langle x_{u+1} \oplus x_{n+1}, 1\rangle, \langle x_u \oplus x_{n+1}, 0\rangle, \langle x_{u-1} \oplus x_{n+1}, 1\rangle, \ldots, \langle x_1 \oplus x_{n+1}, 1\rangle, \langle 1, 0\rangle,$$

If the application is the negation of the parity of two variables, we include $x_{n+1} \oplus 1$ instead of x_{n+1} in each condition. Either way, x_{n+1} is the only variable such that the two restrictions $f'_i[x_{n+1} \leftarrow 0]$ and $f'_i[x_{n+1} \leftarrow 1]$ are dual to each other, so any isomorphism from L'_1 to L'_2 must map x_{n+1} to itself. Thus we again obtain a reduction from DIPATHCENTER to $\mathcal{C}$-DL-ISO. ☐

We remark that isomorphism of $\{x_1\}$-CSP instances can be decided in TC^0 by comparing the number of variables that occur in constraint applications, so Theorem 5.15 does not hold for $\mathcal{C}$-CSP isomorphism.

5.2. Reducing isomorphism of Schaefer decision lists to graph isomorphism

In this section, we show that $\mathcal{C}$-DL isomorphism is reducible to graph isomorphism if $\overline{\mathcal{C}}$ is Schaefer, adapting the methods of [BHRV02] for CSP instances to the layered structure of decision lists. We give a reduction from $\mathcal{C}$-DL-Iso to the label-respecting isomorphism problem of labeled trees, which is equivalent to graph isomorphism [RZ00]. In this problem, we are given two rooted trees where each vertex has a label. We ask if there is an isomorphism between the trees which is label-respecting, i.e., two vertices in the first tree have the same label if and only if their images in the second tree have the same label. A generalized version of this problem that is also GI-complete is isomorphism of colored labeled trees, where each vertex additionally has a color and we ask for a color-preserving and label-respecting isomorphism.

Let L be a given $\mathcal{C}$-DL, where $\overline{\mathcal{C}}$ is Schaefer. The first step is to find a normal form. Since we will use the same normal form representation again in the next subsection, we will describe it for arbitrary $\mathcal{C}$-DLs. Let $\mathcal{C}(n)$ denote the set of all n-ary applications of the constraints in $\mathcal{C}$. Its cardinality is bounded by $|\mathcal{C}| \cdot n^r$, where r is the maximum arity of a constraint in $\mathcal{C}$. We partition $\mathcal{C}(n)$ into the sets $C_1(L), \dots, C_m(L), C_{m+1}(L)$ such that each $C_k(L)$ contains all $f \in \mathcal{C}(n) \setminus \bigcup_{i<k} C_i(L)$ that satisfy

$$\forall a \in \{0,1\}^n : \left(f(a) \wedge \bigwedge_{g \in \bigcup_{i<k} C_i(L)} \neg g(a) \right) = 1 \Rightarrow L(a) = k \bmod 2, \tag{1}$$

where $C_1(L)$ might be empty. Given a constraint application $f \in \mathcal{C}(n)$, its *level* $\mathrm{lv}_L(f)$ is the index k for which $f \in C_k(L)$. The normal form of L is defined as the decision list

$$N_L = \big\langle C_1(L), 1 \big\rangle, \big\langle C_2(L), 0 \big\rangle, \big\langle C_3(L), 1 \big\rangle, \dots, \big\langle C_m(L), m \bmod 2 \big\rangle, \big\langle 1, m+1 \bmod 2 \big\rangle,$$

where for each k, the set $C_k(L)$ represents the function $\bigvee_{f \in C_k(L)} f$.

Theorem 5.16. *Let $\mathcal{C}$ be a class of functions, each depending on at most r variables, such that $\overline{\mathcal{C}}$ is Schaefer. Then the $\mathcal{C}$-DL isomorphism problem is polynomial-time reducible to graph isomorphism.*

Proof. Let $L = \langle f_1, c_1 \rangle, \dots, \langle f_\ell, c_\ell \rangle$ be a given $\mathcal{C}$-DL, where $\overline{\mathcal{C}}$ is Schaefer. We can assume that in each pair $\langle f_i, c_i \rangle$ in L, f_i is a conjunction (or parity) of literals, i.e., the outer disjunctions are resolved by splitting them into several pairs. Similarly, in the normal form $N_L = \big\langle C_1(L), 1 \big\rangle, \dots, \big\langle C_m(L), m \bmod 2 \big\rangle, \big\langle 1, m + 1 \bmod 2 \big\rangle$ of L we can assume that each set $C_k(L)$ consists only of such functions. In other words, it suffices to restrict $\mathcal{C}(n)$ to conjunctions (or parities) of literals of arity at most r over $\{x_1, \dots, x_n\}$.

We first compute the normal form N_L of L. By Theorem 5.2 we can check property (1) above in polynomial time, since a condition $f \in \mathcal{C}(n)$ does not fulfill this property if and only if for some $j \in \{1, \dots, \ell\}$ with $c_j \neq k \bmod 2$, the function

$$f \wedge \bigwedge_{g \in \bigcup_{i<k} C_i(L)} \neg g \wedge \bigwedge_{i<j} \neg f_i \wedge f_j$$

is satisfiable. This function can be encoded as $\mathcal{C}'$-CSP instance for some Schaefer class $\mathcal{C}' \supseteq \overline{\mathcal{C}}$, as f and f_j are parities if $\overline{\mathcal{C}}$-CSP is affine, and conjunctions of literals otherwise.

The next step is to encode N_L as a labeled tree T_L (in the sense of [RZ00]) such that two normal forms N_{L_1} and N_{L_2} are syntactically isomorphic if and only if there is a label-respecting tree isomorphism from T_{L_1} to T_{L_2}. As the type of the constraints in $\overline{\mathcal{C}}$ is fixed (i.e., either Horn, anti-Horn, bijunctive or affine), each constraint application $f \in \mathcal{C}(n)$ is already uniquely determined by a set L_f of literals and a bit b_f. For example, when $\overline{\mathcal{C}}$ is Horn, the set $L_f = \{x_{i_1}, \dots, x_{i_{k-1}}, \overline{x_{i_k}}\}$ uniquely identifies the corresponding function $f = x_{i_1} \wedge \dots \wedge x_{i_{k-1}} \wedge \overline{x_{i_k}}$, and if $\overline{\mathcal{C}}$ is affine, the variable set $L_f = \{x_{i_1}, \dots, x_{i_k}\}$ and the bit b_f uniquely identify the function $f = x_{i_1} \oplus \dots \oplus x_{i_k} \oplus b_f$. As b_f is only needed in the affine case, we fix it to 0 in the other cases.

We now outline the encoding algorithm which computes a labeled colored tree T_L on input N_L. We create a root node with m children corresponding to $C_1(L), \ldots, C_m(L)$, where the node for $C_k(L)$ is colored k. In the subtree rooted at the node corresponding to $C_k(L)$ we create a child c_f with color b_f for each function $f \in C_k(L)$. The node c_f will have $|L_f|$ children which are leaves $v_{k,f,i,b}$ corresponding to the literals x_i^b in L_f, where $v_{k,f,i,b}$ is labeled by the variable index i and colored by the bit b. This completes the construction of the tree T_L.

It is easy to verify that if N_{L_1} and N_{L_2} are syntactically isomorphic via a permutation π, then π maps the collection $\{L_f \mid f \in C_k(L)\}$ of literal sets to the collection $\{L_f \mid f \in C_k(L')\}$, and thus induces a label-respecting and color-preserving isomorphism from T_{L_1} to T_{L_2}. Conversely, if there is a label-respecting and color-preserving isomorphism ψ from T_{L_1} to T_{L_2}, then ψ induces a permutation π on the leaf labels, which provides a syntactic isomorphism from N_{L_1} to N_{L_2}. □

5.3. An upper bound for isomorphism of general decision lists

Let $\mathcal{C}$ be any finite class of constraints. Böhler et al. have shown that the isomorphism problem for $\mathcal{C}$-CSP instances is in $\mathsf{P}_{\|}^{\mathsf{NP}}$ [BHRV02, Corollary 23], where the oracle queries are parallel and do not depend on the answers to previous oracle queries. In this section, we extend this result to the more general isomorphism problem for $\mathcal{C}$-DLs.

The following lemma is similar to Lemma 5.8 and allows us to compute the normal form defined in the preceding subsection in polynomial time by asking parallel queries to an NP oracle. Recall that for a $\mathcal{C}$-DL $L = \langle f_1, c_1\rangle, \ldots, \langle f_\ell, c_\ell\rangle$ and a function g, we have defined $L[g] = \langle f_1, c_1\rangle, \ldots, \langle f_{i-1}, c_{i-1}\rangle\langle 1, c_i\rangle$, where i is the smallest index for which $g \wedge \bigwedge_{j<i} \neg f_j$ implies f_i.

Lemma 5.17. *Let $\mathcal{C}$ be any finite class of constraints that is closed under negation, let L be a $\mathcal{C}$-DL on n variables and let $f, g \in \mathcal{C}(n)$. Then $\mathrm{lv}_L(f) \le \min\{\mathrm{lv}_L(g), \mathrm{lv}_L(\neg g)\}$ if and only if the two lists $L[f,g]$ and $L[f,\neg g]$ are equivalent.*

Proof. Let N_L be the normal form of L as defined above. The decision list $L[f]$ is equivalent to $N_L[f] = \langle C_1(L), 1\rangle, \ldots, \langle C_{i-1}(L), i-1 \bmod 2\rangle, \langle 1, i \bmod 2\rangle$, where $i = \mathrm{lv}_L(f)$. If $\mathrm{lv}_L(f) \le \min\{\mathrm{lv}_L(g), \mathrm{lv}_L(\neg g)\}$, the constraint applications g and $\neg g$ do not occur in $N_L[f]$. By construction of N_L, this implies $N_L[f,g] = N_L[f,\neg g] = N_L[f]$. Thus $L[f,g]$ and $L[f,\neg g]$ are equivalent.

In case $\mathrm{lv}_L(f) > \min\{\mathrm{lv}_L(g), \mathrm{lv}_L(\neg g)\} = j$, let a be an assignment with $C_{j+1}(L)(a) = 1$ and $C_k(L)(a) = 0$ for all $k \le j$. As $\max\{\mathrm{lv}_L(g), \mathrm{lv}_L(\neg g)\} = \ell + 1$, we have $N_L[f,g](a) \ne N_L[f,\neg g](a)$, so $L[f,g]$ and $L[f,\neg g]$ are not equivalent. □

Theorem 5.18. *$\mathcal{C}$-DL-Iso is in $\mathsf{P}_{\|}^{\mathsf{NP}}$ for any finite class $\mathcal{C}$ of constraints.*

Proof. To decide $\mathcal{C}$-DL-Iso we can use a constant number of rounds of parallel NP queries as Buss and Hay have shown that is equivalent to using one round of parallel NP queries [BH91].

Let L and L' be the given $\mathcal{C}$-DLs for n-ary boolean functions f and f', respectively. Our algorithm first computes the normal forms $N_L = \big\langle C_1(L), 1\big\rangle, \ldots, \big\langle C_m(L), m \bmod 2\big\rangle, \big\langle 1, m+1 \bmod 2\big\rangle$ and $N_{L'} = \big\langle C_1(L'), 1\big\rangle, \ldots, \big\langle C_m(L'), m \bmod 2\big\rangle, \big\langle 1, m+1 \bmod 2\big\rangle$ using two rounds of parallel queries; the algorithm is described below. To decide whether the functions represented by them are isomorphic, it suffices to check whether N_L and $N_{L'}$ are syntactically isomorphic, i.e., whether there is a permutation $\pi \in S_n$ such that $C_k(L)^\pi = C_k(L')$ for all levels $k \in \{1, \ldots, m\}$. This can be done using one more NP query.

To compute the normal form N_L of $L = \langle f_1, c_1\rangle, \ldots, \langle f_m, c_\ell\rangle$, we use the first round of parallel NP queries to compute the decision lists $L[f,g]$ for all $f, g \in \mathcal{C}(n)$. This is possible as $f \wedge g \wedge \bigwedge_{j<i} \neg f_j$ implies f_i if and only if $f \wedge g \wedge \bigwedge_{j\le i} \neg f_j$ is not satisfiable.

We may assume w.l.o.g. that $\mathcal{C}$ is closed under negation. In the second round of parallel queries, we ask for all $f, g \in \mathcal{C}(n)$, whether $L[f,g]$ and $L[f,\neg g]$ are equivalent. By Lemma 5.17, this allows to partition $\mathcal{C}(n)$ by level. It remains to check whether $C_1(L)$ is empty, which is true if for some $f \in \mathcal{C}(n)$ of minimum level, the decision list $L[f]$ is a tautology (this can be asked for all $f \in \mathcal{C}$ during the second round of parallel queries). □

An interesting question is whether it is also possible to compute a canonical representation for $\mathcal{C}$-DLs in the class $\mathsf{FP}_{\parallel}^{\mathsf{NP}}$. However, this seems unlikely as it would imply that graphs can be canonized in $\mathsf{FP}_{\parallel}^{\mathsf{NP}}$.

Acknowledgements. We thank Heribert Vollmer for bringing the question addressed in Section 5.3 to our attention, and the anonymous referees for their detailed and helpful remarks.

References

[ADKK12] V. Arvind, Bireswar Das, Johannes Köbler, and Sebastian Kuhnert, *The isomorphism problem for k-trees is complete for logspace*, Inf. Comput. **217** (2012), 1–11.

[AT00] Manindra Agrawal and Thomas Thierauf, *The formula isomorphism problem*, SIAM J. Computing **30** (2000), no. 3, 990–1009.

[BC08] László Babai and Paolo Codenotti, *Isomorhism of hypergraphs of low rank in moderately exponential time*, FOCS, 2008, pp. 667–676.

[BdW02] Harry Buhrman and Ronald de Wolf, *Complexity measures and decision tree complexity: a survey*, Theor. Comput. Sci. **288** (2002), no. 1, 21–43.

[BH91] Samuel R. Buss and Louise Hay, *On truth-table reducibility to SAT*, Information and Computation **91** (1991), no. 1, 86–102.

[BHRV02] Elmar Böhler, Edith Hemaspaandra, Steffen Reith, and Heribert Vollmer, *Equivalence and isomorphism for Boolean constraint satisfaction*, CSL, 2002, pp. 881–920.

[BHRV04] ———, *The complexity of Boolean constraint isomorphism*, STACS, 2004, pp. 164–175.

[BL83] László Babai and Eugene M. Luks, *Canonical labeling of graphs*, STOC, 1983, pp. 171–183.

[Blu92] Avrim Blum, *Rank-r decision trees are a subclass of r-decision lists*, Inf. Process. Lett. **42** (1992), no. 4, 183–185.

[DTW12] Bireswar Das, Jacobo Torán, and Fabian Wagner, *Restricted space algorithms for isomorphism on bounded treewidth graphs*, Information and Computation **217** (2012), 71–83.

[EH89] Andrzej Ehrenfeucht and David Haussler, *Learning decision trees from random examples*, Inf. Comput. **82** (1989), no. 3, 231–246.

[Ete97] Kousha Etessami, *Counting quantifiers, successor relations, and logarithmic space*, J. Comp. Syst. Sci. **54** (1997), no. 3, 400–411.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane, *Which problems have strongly exponential complexity?*, J. Comput. Syst. Sci. **63** (2001), no. 4, 512–530.

[JKMT03] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán, *Completeness results for graph isomorphism*, J. Computer and System Sciences **66** (2003), no. 3, 549–566.

[KM93] Eyal Kushilevitz and Yishay Mansour, *Learning decision trees using the Fourier spectrum*, SIAM J. Comput. **22** (1993), no. 6, 1331–1348.

[LMN93] Nathan Linial, Yishay Mansour, and Noam Nisan, *Constant depth circuits, Fourier transform, and learnability*, J. ACM **40** (1993), no. 3, 607–620.

[Luk99] Eugene M. Luks, *Hypergraph isomorphism and structural equivalence of Boolean functions*, STOC, 1999, pp. 652–658.

[Rei08] Omer Reingold, *Undirected connectivity in log-space*, J. ACM **55** (2008), no. 4, 17:1–17:24.

[Riv87] Ronald L. Rivest, *Learning decision lists*, Machine Learning **2** (1987), no. 3, 229–246.

[RZ00] Sarnath Ramnath and Peiyi Zhao, *On the isomorphism of expressions*, Inf. Process. Lett. **74** (2000), no. 3-4, 97–102.

[Sch78] Thomas J. Schaefer, *The complexity of satisfiability problems*, STOC, 1978, pp. 216–226.

[Thi00] Thomas Thierauf, *The computational complexity of equivalence and isomorphism problems*, LNCS, vol. 1852, Springer, 2000.