On the Parallel Complexity of Bisimulation on Finite Systems

Moses Ganardi[1], Stefan Göller[2], and Markus Lohrey[1]

1 **University of Siegen, Germany**
ganardi@eti.uni-siegen.de
2 **Laboratoire Specification et Verification (LSV), ENS de Cachan, France; and CNRS, France**
goeller@lsv.ens-cachan.fr
3 **University of Siegen, Germany**
lohrey@eti.uni-siegen.de

Abstract

In this paper the computational complexity of the (bi)simulation problem over restricted graph classes is studied. For trees given as pointer structures or terms the (bi)simulation problem is complete for logarithmic space or NC^1, respectively. This solves an open problem from Balcázar, Gabarró, and Sántha. We also show that the simulation problem is P-complete even for graphs of bounded path-width.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases bisimulation, computational complexity, tree width

Digital Object Identifier 10.4230/LIPIcs.CSL.2016.12

1 Introduction

Courcelle's theorem states that every problem definable in monadic second-order logic (MSO) is solvable in linear time on graphs of bounded tree-width. In recent works by Elberfeld, Jakoby, and Tantau, techniques have been developed to transfer this famous result to low space and circuit complexity classes [6, 7]. In particular, the following logspace (resp., NC^1) version of Courcelle's theorem was shown (see Section 2 for the necessary definitions):

▶ **Theorem 1** ([6, 7]). *For a fixed MSO-sentence ψ and a fixed constant k one can check in logspace whether a given structure $\mathcal{A}$ of tree-width at most k satisfies ψ. If a tree decomposition of $\mathcal{A}$ of width k is given in term representation, then one can check in* $\mathsf{DLOGTIME}$*-uniform* NC^1 *whether $\mathcal{A}$ satisfies ψ.*

This result is a very powerful metatheorem, which can be applied to many computational problems. On the other hand, there are important problems solvable in logspace on graphs of bounded tree-width that are not covered by Theorem 1. One example is the graph isomorphism problem. Graph isomorphism is not MSO-definable even over finite paths since two finite paths are isomorphic if they have the same length, but one cannot express in MSO that two finite sets have the same size. Lindell [17] has shown that isomorphism of trees is in logspace, and only very recently Elberfeld and Schweitzer [8] extended this result to graphs of bounded tree-width.

In this paper, we are concerned with the complexity of simulation and bisimulation, which are of fundamental importance in the theory of reactive systems, see e.g. [1] for more

25th EACSL Annual Conference on Computer Science Logic (CSL 2016).
Editors: Jean-Marc Talbot and Laurent Regnier; Article No. 12; pp. 12:1–12:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

background. It is known that on finite state systems simulation and bisimulation are both P-complete [2], and hence have no efficient parallel algorithm unless $\mathsf{P} = \mathsf{NC}$. Surprisingly, no results on the complexity of (bi)simulation on natural subclasses of finite state systems are known (whereas there exists an extensive literature on (bi)simulation problems for various classes of infinite state systems, like pushdown systems or Petri nets). The authors of [2] pose this open question and suggest to consider the bisimulation problem on trees. The above remark that tree isomorphism cannot be expressed in MSO applies to bisimulation on trees as well (two finite paths are bisimilar if and only if they are isomorphic). Moreover, it is not clear, whether there is a natural reduction of the bisimulation problem on trees to the logspace-solvable isomorphism problem for trees (or even bounded tree-width graphs).

In this paper, we determine the complexity of the bisimulation problem on trees. More precisely, we show the following results:

- On trees the (bi)simulation problem is complete for logarithmic space (resp., NC^1) if the trees are given as pointer structures (resp., in term representation).
- The simulation problem is P-hard (and hence P-complete) already for graphs of bounded path-width.

Whether the bisimulation problem on graphs of bounded tree-width is in NC remains open.

We prove our results for the bisimulation problem by a reduction to the evaluation problem for a new class of Boolean circuits that we call *tree-shaped circuits*. These are circuits that are composed in a tree-like fashion of smaller subcircuits. We define the width of such a circuit as the maximal number of different paths from the root to an input in one of the above mentioned subcircuits. The main technical contributions of this paper are logspace- and NC^1-evaluation algorithms (depending on the input representation) for tree-shaped circuits of bounded width. These circuits should not be confused with circuits of bounded tree-width, which are known to have logspace- and NC^1-evaluation algorithms (depending on the representation) by the above Theorem 1. We show how to partially unfold tree-shaped circuits of bounded width into circuits of bounded tree-width. This unfolding is possible in TC^0 (assuming the right representation of the circuit). Finally, the resulting bounded tree-width circuit can be evaluated using Theorem 1. For the above logspace result we actually prove a stronger statement: A given tree-shaped circuit of size N and width m can be evaluated in space $O(\log N \cdot \log m)$.

One should also mention the paper [10], where a logic LREG (which extends classical first-order logic) is introduced. It is shown that LREG captures logspace on directed trees. Hence, bisimulation for trees is expressible in LREG. Due to the very technical definition of LREG we think that it is not easier to express the bisimulation problem in LREG than giving a direct logspace algorithm. Moreover expressibility in LREG does not imply that bisimulation for trees in term representation is in NC^1.

2 Preliminaries

Graphs and trees. A *(directed) graph* $G = (V, E)$ consists of a set of nodes $V(G) = V$ and a set of edges $E(G) = E \subseteq V \times V$. If $(u, v) \in E$, we call v a *successor* of u. A *path* (of length n from v_0 to v_n) in G is a node sequence $v_0, v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < n$. A graph is *acyclic* if there is no path of length ≥ 1 from a node to itself. We say that two nodes $u, v \in V$ are *connected* if there exists a path from u to v in the underlying undirected graph $(V, E \cup \{(v, u) \mid (u, v) \in E\})$. A set of nodes $U \subseteq V$ is *connected* if any two nodes in U are connected. The *size* $|G|$ of a graph G is the number of nodes $|V(G)|$.

A graph T is a *(rooted) tree* if there exists a node $r \in V(T)$, called the *root* of T, such that for all $v \in V(T)$ there exists exactly one path from r to v. The *depth* of a node u,

denoted by $\mathrm{depth}(u)$, is the length of the unique path from the root to u. A node u is an *ancestor* (resp., *proper ancestor*) of v, briefly $u \preceq v$ (resp., $u \prec v$), if there exists a path (resp., a path of length ≥ 1) in T from u to v.

A *node-labelled graph* (V, E, β) is a graph (V, E) together with a labelling function $\beta : V \to A$ into a finite set A. An *edge-labelled graph* (V, E) consists of a set of nodes V and a labelled edge relation $E \subseteq V \times A \times V$. We also write $u \xrightarrow{a} v$ instead of $(u, a, v) \in E$. Unlabelled graphs are also regarded as labelled graphs over a singleton label set. An edge-labelled tree is an edge-labelled graph (V, E) where the sets $E_a = \{(u, v) \mid (u, a, v) \in E\}$ are pairwise disjoint for $a \in A$ and $(V, \bigcup_{a \in A} E_a)$ is a directed tree.

(Bi)simulation. A *bisimulation* on an edge-labelled graph (V, E) is a binary relation $R \subseteq V \times V$ such that for all $(u, v) \in R$ the following conditions hold:

1. For all $u \xrightarrow{a} u'$ there exists $v \xrightarrow{a} v'$ such that $(u', v') \in R$.
2. For all $v \xrightarrow{a} v'$ there exists $u \xrightarrow{a} u'$ such that $(u', v') \in R$.

A relation R that only satisfies condition 1 for all $(u, v) \in R$ is called a *simulation*. A (bi)simulation on two edge-labelled graphs is a (bi)simulation on their disjoint union. Two nodes u, v are called *bisimilar* if there exists a bisimulation R such that $(u, v) \in R$. We say that u *is simulated by* v if there exists a simulation R such that $(u, v) \in R$.

It is easy to see that the union of all bisimulations is a bisimulation again and an equivalence relation, called the *bisimulation equivalence*. On finite graphs the bisimulation equivalence can be computed in polynomial time by a partition refinement algorithm [15]. In fact, the *bisimulation problem*, i.e., deciding whether two nodes are bisimilar in a given finite graph, is P-complete [2].

Tree-width and path-width. A *tree decomposition* (T, β) of a directed graph G is a node-labelled tree T, where $\beta : V(T) \to 2^{V(G)}$ assigns to each node of T a so called *bag* such that

- for all $v \in V(G)$ the set $\{t \in V(T) \mid v \in \beta(t)\}$ is non-empty and connected, and
- for all $(u, v) \in E(G)$ there exists $t \in V(T)$ such that $u, v \in \beta(t)$.

The *width* of (T, β) is $\max_{t \in V(T)} |\beta(t)| - 1$ and the *tree-width* of a graph G is the minimum width over all tree decompositions of G. A tree decomposition (T, β) is a *path decomposition* if T is a path. The *path-width* of a graph G is the minimum width of all path decompositions of G. Tree-width and path-width are also defined for node- and edge-labelled graphs via their underlying unlabelled graph. More background of tree-width can be found for instance in [5].

Circuits. A *(Boolean) circuit* $C = (G, \beta)$ is a node-labelled graph, where G is acyclic and $\beta : V(G) \to \{x_1, \ldots, x_n, 0, 1, \neg, \wedge, \vee\}$ for some n. Nodes of C are usually called *gates*. Gates labelled by 0, 1 (constant gates) or by a variable x_i (input gates) have no successors. Gates labelled by $\wedge$ or $\vee$ have at least one successor, and gates labelled by $\neg$ have exactly one successor. A *variable-free* circuit is a circuit without input gates. In a variable-free circuit every gate can be evaluated to either 0 or 1. The question, whether a given gate of a given variable-free circuit evaluates to 1 is known as the *circuit value problem*. It is one of the classical P-complete problems, see [9] for more details.

Complexity classes. In this paper we will use the complexity classes $\mathsf{AC}^0 \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{NC} \subseteq \mathsf{P}$. A function $f : \Sigma^* \to \Gamma^*$ is *logspace-computable* if it can be computed on a deterministic Turing-machine with a read-only input tape, a write-only output tape and a working tape whose length is bounded logarithmically in the input length. We denote by $\mathsf{L}$the

class of languages which can be decided in logspace, i.e. for which the characteristic function is logspace-computable. Throughout the paper we will use implicitly that compositions of logspace-computable functions are logspace-computable again. In the following we denote by NC^1 the class of all languages which are accepted by a $\mathsf{DLOGTIME}$-uniform family of bounded fan-in Boolean circuits of polynomial size and logarithmic depth [19]. We also use the obvious generalization of this definition to functions. If we allow unbounded fan-in circuits but require constant depth, we obtain the class AC^0. If we additionally allow threshold gates, we obtain the class TC^0. It can be seen as the extension of AC^0 by the ability of counting. Typical problems in TC^0 are the computation of the sum, product and integer quotient of two binary encoded integers, and the sum and product of an arbitrary number of binary encoded integers [12]. Note that AC^0, TC^0, and NC^1 always refer to their $\mathsf{DLOGTIME}$-uniform versions in this paper. For more details on space and circuit complexity we refer to [19].

Tree representations. The complexity of tree problems often depends on how the trees are represented. Firstly, trees can be given as *pointer structures* where the edge relation is given explicitly as a list of pairs consisting of two node names, which is the standard encoding of graphs in general. Secondly, trees can be given in *term representation* (or *bracket representation*): The string `()` represents a tree of size 1. If a tree T has a root and direct subtrees $T_1, \ldots, T_n$ which have term representations $r_1, \ldots, r_n$, then the string $(r_1 \cdots r_n)$ is a term representation of T. Notice that a tree can have multiple term representations. Thirdly, trees can be represented in *ancestor representation* where we specify a list of all pairs (u, v) where u is an ancestor of v. Elberfeld et al. showed that term and ancestor representations can be converted into each other in TC^0 [7]. This is useful since operating on ancestor representations of trees is technically easier than on term representations.

To represent node-labelled trees (e.g., tree decompositions), in the pointer and ancestor representation we append a list of pairs consisting of a node name and a node label. In the term representation node labelled trees can be encoded by introducing for each label a an opening bracket symbol $(_a$. For instance, $(_a(_b)(_a))$ encodes a tree with an a-labelled root and two children which are labelled by b and a. If the set of node labels is not fixed (this is the case for tree decompositions) we choose an arbitrary binary block code for the opening brackets $(_a$. To represent an edge-labelled tree we transform it into a node-labelled tree, e.g. by assigning the label of an edge (u, v) to its end point v and labelling the root by a special symbol. Let us finally mention that these coding details for labelled trees are only relevant for our NC^1 lower bound in Section 4.1, which refers to AC^0-reductions. The reason is that different codings of labelled trees in term representation can be transformed in TC^0, but not necessarily in AC^0, into each other.

3 Bisimulation on Trees

In this section we consider the bisimulation problem on edge-labelled trees, i.e. the question whether the roots of two given edge-labelled trees are bisimilar. We show that the bisimulation problem is L-complete if the trees are encoded as pointer structures and NC^1-complete if the trees are given in term representation. We remark that the same complexity bounds hold for the tree isomorphism problem [4, 13, 17].

3.1 Trees as pointer structures

We start with showing a logspace upper bound for the bisimulation problem for trees in pointer representation. Bisimilarity between two edge-labelled trees T_1 and T_2 can be expressed as a

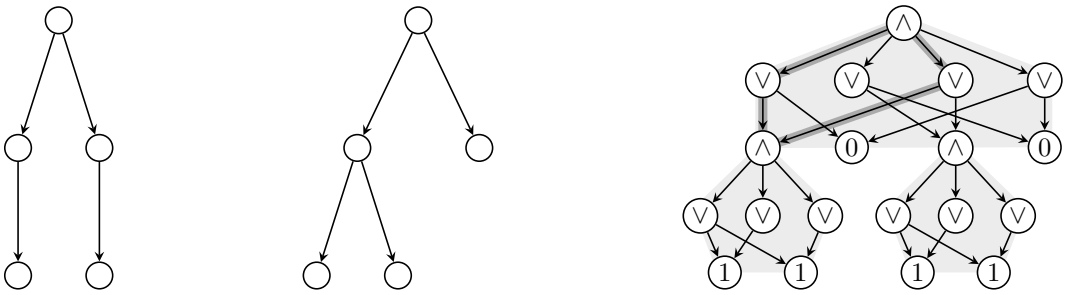


Figure 1 Two trees T_1, T_2 and the tree-shaped circuit $C(T_1, T_2)$ for bisimulation equivalence.

Boolean circuit $C(T_1, T_2)$: For all $u \in V(T_1), v \in V(T_2)$ such that $\mathrm{depth}(u) = \mathrm{depth}(v)$ the circuit contains a gate $x_{u,v}$, which evaluates to true if and only if u is bisimilar to v. We define

$$x_{u,v} = \bigwedge_{u \xrightarrow{a} u'} \bigvee_{v \xrightarrow{a} v'} x_{u',v'} \wedge \bigwedge_{v \xrightarrow{a} v'} \bigvee_{u \xrightarrow{a} u'} x_{u',v'}. \tag{1}$$

As usual we regard an empty conjunction (resp., disjunction) as 1 (resp., 0). In particular, if both u and v are leaves, then $x_{u,v} = 1$, and if exactly one of u and v is a leaf, then $x_{u,v} = 0$. An example circuit is shown in Figure 1, where T_1 and T_2 are unlabelled. Note that the circuit is composed in a tree-shaped form from smaller circuits. These smaller circuits correspond to the definition in (1) and have the crucial property that there exist exactly two paths from the root to an arbitrary leaf, which are highlighted in one subcircuit in Figure 1. This is the case because each gate variable $x_{u',v'}$ occurs once in the first conjunction and once in the second conjunction in (1). In fact, we will show that circuits with such a path property can be evaluated in logspace.

We use a more syntactic definition of such circuits: A *tree-shaped circuit* is a sequence of Boolean equations $\mathcal{S} = (x_i = \varphi_i)_{1 \le i \le n}$ where $\varphi_1, \ldots, \varphi_n$ are Boolean formulas over the variables $x_1, \ldots, x_n$ such that the graph

$$T_{\mathcal{S}} = (\{x_1, \ldots, x_n\}, \{(x_i, x_j) \mid x_j \text{ occurs in } \varphi_i\}) \tag{2}$$

is a tree with root x_1. This implies that there are no cyclic definitions in $\mathcal{S}$ and that no variable x_k appears in two different formulas φ_i and φ_j $(i \neq j)$. The *size* of $\mathcal{S}$ is defined as the sum of the sizes of all formulas φ_i. The *width* of $\mathcal{S}$ is defined as the maximal number of occurrences of a variable x_j in a formula φ_i. An example of a tree-shaped circuit of width two is given by the formulas (1) for bisimulation. We can view $\mathcal{S}$ as an ordinary Boolean circuit by taking the disjoint union of the formula trees of the φ_i and then merging all x_i-labelled leaves with the root of the formula tree of φ_i. For example, the tree-shaped circuit $C(T_1, T_2)$ for bisimulation equivalence can be regarded as a tree-shaped circuit of width 2, which can be computed in logspace from T_1 and T_2. The main goal of this section is to show that the circuit value problem restricted to tree-shaped circuits of bounded width belongs to logspace.

It is interesting to compare tree-shaped circuits of constant width with a class of circuits that is presented in [10, page 5]. For the latter, the authors require that for every path in the circuit the product of the fan-outs (i.e., indegrees, since we direct circuits towards the input gates) of the gates on the path is bounded polynomially by the circuit size. It is shown that circuits with this property can be evaluated in logspace. Note that tree-shaped circuits of constant width do not have this path property from [10]. On the other hand, the circuits

from [10] can have nodes with large fan-out, which is not possible for tree-shaped circuits of constant width. Hence, the two circuit classes are incomparable.

Note that a variable-free Boolean circuit can be represented by a relational structure with a binary edge relation and unary relations for the labels 0, 1, $\neg$, $\wedge$, $\vee$. Moreover, there is a formula of monadic-second order logic (MSO) expressing that a variable-free Boolean circuit evaluates to 1. As a consequence, by Theorem 1, we can evaluate variable-free Boolean circuits of tree-width at most k in logspace for every fixed k. Moreover, the complexity can be improved to NC^1 if we also provide for the input circuit a bounded width tree decomposition in term representation. For this one stores a tree decomposition (T, β) by an expression for the node-labelled tree T, where every node $t \in V(T)$ is labelled by the bag $\beta(t)$. Note that $\beta(t)$ is a set of gates of the circuit, and these gates are stored by their addresses. To sum up, we have:

▶ **Theorem 2.** *For every fixed $k \in \mathbb{N}$, the circuit value problem restricted to circuits of tree-width at most k can be solved in logspace. If in addition to the input circuit C a width-k tree decomposition of C in term representation is given, then the circuit value problem can be solved in NC^1.*

However, we cannot directly apply Theorem 2 to tree-shaped circuits since neither their tree-width nor clique-width is bounded by the following lemma. Clique-width is another graph measure defined by so called k-expressions, see e.g. [14] for a survey. It is easy to see that Theorem 2 can be generalized to circuits of clique-width k, provided a k-expression for the circuit is part of the input.

▶ **Lemma 3.** *For $n \geq 1$, let T_n be the tree of size $n+1$ whose root has exactly n children. The set of circuits $\{C(T_n, T_n) \mid n \geq 1\}$ has unbounded tree-width and unbounded clique-width.*

Proof. The underlying undirected graph of $C(T_n, T_n)$ contains the complete bipartite graph $K_{n,n}$ as a (topological) minor. This can be seen by removing the output gate and dissolving all constant gates (*dissolving* a node of degree 2 means deleting it and connecting its two neighbors. Figure 2 shows an example for $n = 3$. Since the tree-width of $K_{n,n}$ is n and the tree-width of every minor of a graph G is bounded by the tree-width of G, it follows that the tree-width of $C(T_n, T_n)$ is at least n.

It is known that a set of graphs which has bounded clique-width and for which there exist only finitely many n such that the bipartite graph $K_{n,n}$ is contained as a subgraph (not only minor) also has bounded tree-width [11]. We claim that for each $n \geq 1$ the undirected graph of $C(T_n, T_n)$ does not contain $K_{3,3}$ as a subgraph, which implies that $\{C(T_n, T_n) \mid n \geq 1\}$ also has unbounded clique-width (note that $C(T_n, T_n)$ contains a $K_{2,2}$, i.e., a cycle on four nodes). Note that in $C(T_n, T_n)$ for every simple path of four nodes, one of the four nodes has degree 2 (these are the gates in the middle layer of Figure 2). But this is not possible in a $K_{3,3}$. ◀

Before we show how to evaluate tree-shaped circuits of bounded width in logspace, we introduce some notions. The size $|u|$ of a node u in a tree is the size of the subtree rooted in u. We say that a node u is *heavy* if for all siblings v of u we either have (i) $|u| > |v|$ or (ii) $|u| = |v|$ and $u < v$ (where $<$ denotes some fixed order on the nodes of the tree). Otherwise a node is called *light*. Notice that the root is heavy and that every inner node has exactly one heavy child. We can compute in logspace for each node its size and determine whether it is heavy. Note that every path in a tree contains at most $O(\log n)$ light nodes.

By the following result, tree-shaped circuits of bounded width can be evaluated in logspace (take $m \in O(1)$). Also note that a tree-shaped circuit of width $m = 1$ is a tree and therefore can be evaluated in logspace.

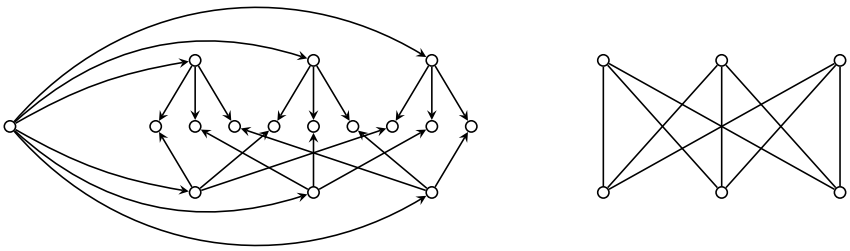


Figure 2 The circuit $C(T_3, T_3)$ for bisimulation equivalence. Removing the output gate on the left and dissolving the constant gates in the middle layer shows that $K_{3,3}$ is a minor of $C(T_3, T_3)$.

▶ **Theorem 4.** *A given tree-shaped circuit $\mathcal{S} = (x_i = \varphi_i)_{1 \le i \le n}$ of width $m \ge 2$ can be evaluated in space $O(\log s + \log n \cdot \log m)$, where $s = \max\{|\varphi_i| \mid 1 \le i \le n\}$ is the maximal size of one of the formulas φ_i. In particular, for every fixed $m \in \mathbb{N}$, there is a logspace algorithm which evaluates a given tree-shaped circuit of width at most m.*

Proof. Let $\mathcal{S} = (x_i = \varphi_i)_{1 \le i \le n}$ be a tree-shaped circuit of width m and let $s = \max\{|\varphi_i| \mid 1 \le i \le n\}$. Recall the definition of the tree $T_{\mathcal{S}}$ with node set $\{x_1, \ldots, x_n\}$ from (2).

First of all, we label every node x_i of $T_{\mathcal{S}}$ by (i) the size $|x_i|$ of the subtree rooted in x_i and (ii) a single bit indicating whether x_i is a heavy node of $T_{\mathcal{S}}$. This information can be computed in space $O(\log n)$ by traversing for each node x_i the subtree of $T_{\mathcal{S}}$ rooted in x_i in depth-first order and counting the number of nodes in binary representation.

The circuit $\mathcal{S}$ is evaluated in a recursive way using a pointer to one of the nodes $x_1, \ldots, x_n$ (which needs space $O(\log n)$) and a stack of height $O(\log n \cdot \log m)$ as follows. Initially the pointer is set to the root x_1. Assume that x_k is the heavy child of x_1 in $T_{\mathcal{S}}$. Then, the pointer is moved to x_k without writing anything on the stack. Next, the subcircuit rooted at x_k is evaluated recursively. By induction, space $O(\log s + \log n \cdot \log m)$ is used for this. Once the algorithm returns from the recursion the pointer is back on x_k and one can release the space. The value of x_k is stored on the stack, which now contains a single bit. Note that the algorithm knows (using the labeling computed before) that x_k is the heavy child of its parent node x_1. This information now triggers the evaluation of the Boolean formula φ_1. This is done by the standard evaluation algorithm that traverses the Boolean formula tree in depth-first order and stores (i) a pointer to the current node of φ_1 (this pointer needs space $O(\log |\varphi_1|)$ and (ii) a constant number of additional bits, indicating the current direction of the traversal (up or down), and the value of the current node in case we move upwards. Each time, this depth-first traversal of φ_1 arrives at a leaf node, the following is done:

- If the leaf is labelled with the variable x_k (the heavy child of x_1 in $T_{\mathcal{S}}$), then the value of x_k is retrieved from the stack and the algorithm continues the evaluation of φ_1.
- If the leaf is a light child labelled with the variable $x_i \ne x_k$, then let $1 \le c \le m$ such that the leaf corresponds to the c-th occurrence of x_i in φ_1. The algorithm stores c on the stack (which needs space $O(\log m)$) and continues recursively with the evaluation of the subcircuit rooted at x_i. Once it comes back from the recursion, the pointer is back on x_i. The algorithm sees that x_i is a light child of x_1. Using this information and the number c stored on the stack, it continues the evaluation of φ_1 at the right position in φ_1.

Note that in the second case, we have $|x_i| \le |x_1|/2$. This implies that the number of bits stored on the stack is bounded by $O(\log n \cdot \log m)$. The total space consumption is therefore $O(\log n + \log s + \log n \cdot \log m) = O(\log s + \log n \cdot \log m)$ (since we assume $m \ge 2$). ◀

Note that we do not assume $m \in O(1)$ in Theorem 4. In particular, since k, n, and m are all bounded by the size of the tree-shaped circuit (which is the sum of the sizes of the formulas φ_i), it follows that a tree-shaped circuit of size N can be evaluated in space $O(\log^2 N)$. For the special case $m \in O(1)$ (which is used for the bisimulation problem) we give an alternative proof below. This proof prepares our handling of trees in term representation in the next section (proof of Theorem 6).

Alternative proof of Theorem 4 for constant width. Let N be the size of $\mathcal{S}$ and $m \in O(1)$ its width. We will construct from $\mathcal{S}$ in logspace an equivalent polynomially sized circuit with constant tree-width, which can be seen as a partial tree unfolding of the circuit corresponding to $\mathcal{S}$. By Theorem 2 the resulting circuit can be evaluated in logspace.

As before, we view the Boolean formulas φ_i as labelled trees. For simplicity we assume that all φ_i have at least size two, which can be ensured by replacing φ_i by $\varphi_i \wedge 1$, so that $\mathcal{S}$ contains no "chain rules". Moreover, we assume that the trees φ_i have disjoint node sets. Let x_i be an inner node of $T_{\mathcal{S}}$ whose heavy child is x_k. Inductively we define a circuit C_i as follows: We take the formula φ_i, viewed as a tree, and merge all x_k-labelled leaves into a single node. Note that x_j-labelled leaves for $j \neq k$ are not merged. Then we insert into each leaf labelled by some variable x_j a copy of the circuit C_j. Finally let C be C_1, which clearly evaluates to the same truth value as $\mathcal{S}$. Figure 3 shows the circuit resulting from the circuit $C(T_1, T_2)$ on the right in Figure 1.

Note that the number of copies of C_i in C is bounded by m^{ℓ_i} where ℓ_i is the number of light nodes on the path from x_1 to x_i in $T_{\mathcal{S}}$. Since $\ell_i \leq \log n$, C has size at most $m^{\log n} \cdot N$, which is bounded by $n^{O(1)} \cdot N$ since m is a constant.

Furthermore C can be computed in logspace from $\mathcal{S}$. To make this explicit, we introduce a naming scheme for the gates in C. The set $\mathrm{Addr}(x_i)$ (addresses for the copies of x_i) contains finite words over the alphabet $\{1, \ldots, m\}$ defined inductively: We set $\mathrm{Addr}(x_1) = \{\varepsilon\}$ for the root x_1. If x_j is the heavy child of x_i in $T_{\mathcal{S}}$, we set $\mathrm{Addr}(x_j) = \mathrm{Addr}(x_i)$. If x_j is a light child of x_i in $T_{\mathcal{S}}$, we set $\mathrm{Addr}(x_j) = \mathrm{Addr}(x_i) \cdot \{1, \ldots, k\}$ where $k \leq m$ is the number of occurrences of x_j in φ_i. The sets $\mathrm{Addr}(x_i)$ contain words of length $O(\log n)$ over the constant sized alphabet $\{1, \ldots, m\}$. Moreover, given a word of length $O(\log n)$ over $\{1, \ldots, m\}$ we can easily check in logspace whether it belongs to $\mathrm{Addr}(x_i)$ by traversing the path from x_i to the root of $T_{\mathcal{S}}$. Now we can define the circuit C over the gate set

$$V(C) = \bigcup_{i=1}^{n} \{(u, a) \mid a \in \mathrm{Addr}(x_i),\ u \text{ is a non-input node of the tree } \varphi_i\}.$$

For every edge (u, u') of φ_i $(1 \leq i \leq n)$ and $a \in \mathrm{Addr}(x_i)$, we add the following edges to $E(C)$:

1. If u' is a non-input gate, add the edge $((u, a), (u', a))$.
2. If u' is labelled by x_j, let v be the root of φ_j and add the edge
   a. $((u, a), (v, a \cdot d))$ if x_j is a light child of x_i and u' is its d-th occurrence of x_j in φ_i,
   b. $((u, a), (v, a))$ if x_j is the heavy child of x_i.

The labels of the gates in C are inherited from the formula trees φ_i. Note that a pair (u, a) from $V(C)$ can be stored in logspace. Moreover, whether a pair belongs to $V(C)$ and whether a pair of nodes from $V(C)$ belongs to $E(C)$ can be checked in logspace. Hence, the circuit C can be constructed in logspace.

Finally we show that the tree-width of C is at most 2. Consider the subgraph T of C where edges of type 2b in the above definition of C are removed if u' is the d-th occurrence of x_j in φ_i for some $d > 1$. In Figure 3 such edges are drawn as dotted lines. The resulting

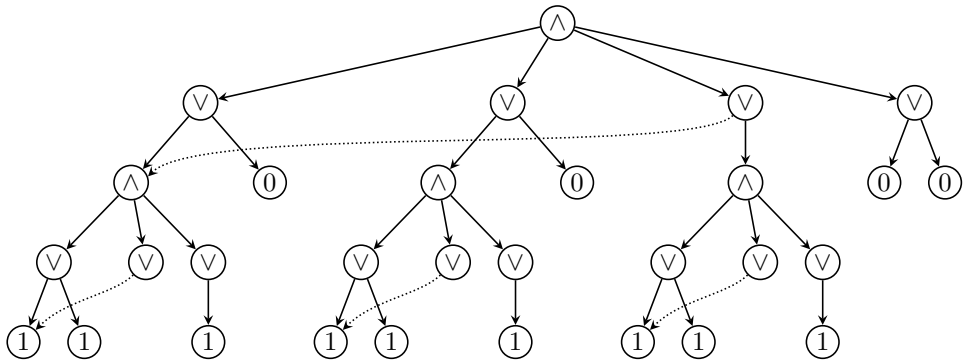


Figure 3 The partial unfolding of the tree-shaped circuit $C(T_1, T_2)$ from Figure 1.

subgraph T is indeed a tree, on which we define a tree decomposition (T, β) of C. Let u be a non-input node of φ_i and $a \in \mathrm{Addr}(x_i)$. The bag $\beta(u, a)$ contains the gate (u, a), its parent node in the tree T (if existent) and the unique node (v, a) where v is the root of φ_j and x_j is the heavy child of x_i (if existent). One can verify that (T, β) is indeed a valid tree decomposition. ◀

We can apply Theorem 4 to the tree-shaped circuit defined by (1) to solve the tree bisimulation problem in logspace:

▶ **Corollary 5.** *The bisimulation problem for trees given as pointer structures is in* L.

3.2 Trees in term representation

Next we will show that the tree bisimulation problem belongs to NC^1 if the trees are given as terms. For that we prove an NC^1-version of Theorem 4 (for $m \in O(1)$), which uses the NC^1-part of Theorem 2, where a tree decomposition in term representation is part of the input. Here we require that the tree-shaped circuit $\mathcal{S} = (x_i = \varphi_i)_{1 \le i \le n}$ must be given together with the underlying tree $T_{\mathcal{S}}$ in term representation. We also assume that the Boolean formulas φ_i are given in term representation. Recall the ancestor representation of a tree from Section 2 and that it can be transformed into the term representation of a tree in TC^0 and vice versa.

▶ **Theorem 6.** *For every fixed $m \in \mathbb{N}$, one can evaluate in NC^1 a given tree-shaped circuit $\mathcal{S} = (x_i = \varphi_i)_{1 \le i \le n}$ (with all φ_i given in term representation) of width at most m that is given together with the term representation of the tree $T_{\mathcal{S}}$.*

Before we prove Theorem 6, we apply it to the tree bisimulation problem.

▶ **Theorem 7.** *The bisimulation problem for trees given in term representations is in* NC^1.

Proof. First we convert the term representations of T_1 and T_2 into ancestor representations in TC^0. We can compute the tree-shaped circuit $\mathcal{S}$ corresponding to the circuit $C(T_1, T_2)$ and an ancestor representation of $T_{\mathcal{S}}$ in TC^0. The set of variables $\{x_{u,v} \mid u \in V(T_1),\ v \in V(T_2),\ \mathrm{depth}(u) = \mathrm{depth}(v)\}$ can be clearly computed in TC^0 since for a given node of a tree in ancestor representation one can compute its depth by counting ancestors. The term representations of the formulas $\varphi_{u,v}$ in (1) can then be computed in AC^0. The ancestor representation of $T_{\mathcal{S}}$ is also AC^0-computable, since $x_{u,v}$ is an ancestor of $x_{u',v'}$ if and only if u is an ancestor of u' and v is an ancestor of v'. By Theorem 6 we can evaluate $\mathcal{S}$ in NC^1. ◀

Proof of Theorem 6. Let $\mathcal{S} = (x_i = \varphi_i)_{1 \le i \le n}$ be a tree-shaped circuit of width at most m, where every φ_i is given in term representation. Moreover, we assume to have the term (or ancestor) representation of the tree $T_{\mathcal{S}}$. We will show how to compute in TC^0 the partial unfolding C of $\mathcal{S}$ and the tree decomposition of C in ancestor representation from the alternative proof of Theorem 4 on page 12:8.

Let N be the size of $\mathcal{S}$. For each node x_i of $T_{\mathcal{S}}$ we can compute in TC^0 its depth (by counting ancestors) and the size of the subtree below x_i (by counting descendants). Hence, we can also compute the heavy child of every inner node x_i in $T_{\mathcal{S}}$. Additionally, if x_i has a parent node x_j, we can compute the number $d(x_i) \in \{1, \dots, m\}$ of occurrences of x_i in φ_j. We transform all formulas φ_i of $\mathcal{S}$ into ancestor representation in TC^0 where we can assume that the encodings of all nodes have length $O(\log N)$. Also we can ensure that all formulas have at least size two.

Recall that every node u of a formula φ_i has multiple copies (u, a) in C where $a \in \mathrm{Addr}(x_i)$ is an address of length $O(\log n)$ defined on page 12:8. Given a string $a \in \{1, \dots, m\}^*$ of length $O(\log n)$ and a node x_i we can verify in TC^0 whether $a \in \mathrm{Addr}(x_i)$: From the ancestor representation of $T_{\mathcal{S}}$ we can compute the sequence of all light nodes $x_{i_1} \prec x_{i_2} \prec \cdots \prec x_{i_k}$ in $T_{\mathcal{S}}$ on the path from the root to x_i. This can be done by sorting all light ancestors of x_i by their depth in ascending order. It is known that sorting n numbers with n bits each is in TC^0 [19]. Then we have $a_1 \cdots a_k \in \mathrm{Addr}(x_i)$ if and only if $a_j \le d(x_{i_j})$ for all $j \in \{1, \dots, k\}$. Hence, we can also encode all gates of the partial unfolding C by strings of length $O(\log N)$ and can compute $V(C)$ in TC^0. With the previous preparation the edge relation $E(C)$ can be computed in AC^0 using the definition on page 12:8.

It remains to show that we can compute in TC^0 the ancestor representation of the width-2 tree decomposition (T, β) of C from page 12:9. We set $V(T) = V(C)$. Let $(u, a), (u', a') \in V(T)$ be nodes where u (resp., u') belongs to φ_i (resp., φ_j). Then (u, a) is an ancestor of (u', a') in T if and only if $x_i \preceq x_j$ in $T_{\mathcal{S}}$, a is a prefix of a' and the following holds:

1. If $x_i = x_j$, then $a = a'$ and $u \preceq u'$ in φ_i.
2. If $x_i \prec x_j$ in $T_{\mathcal{S}}$, let x_k be the unique child of x_i which is an ancestor of x_j.
   a. If x_k is a light node, let $d \in \{1, \dots, m\}$ be the number in a' at position $|a| + 1$. Then the d-th occurrence of x_k in φ_i is a descendant of u.
   b. If x_k is a heavy node, then the first occurrence of x_k in φ_i is a descendant of u.

The last condition forbids those edges that were deleted when constructing the tree T from the partial unfolding C (the dotted edges in Figure 3). The bag-function β can be computed straightforward in TC^0 using its definition on page 12:9. By Theorem 2 we can evaluate C in NC^1, which concludes the proof. ◀

Equality of hereditarily finite sets. The bisimulation problem on trees in term representation arises in a very natural way. A *hereditarily finite set* is either the empty set $\{\}$ or a set $\{a_1, \dots, a_n\}$ containing finitely many hereditarily finite sets $a_1, \dots, a_n$. Hereditarily finite sets have a natural string representation over the bracket symbols { and }. By counting brackets, one can check in TC^0, whether a string over { and } is well-bracketed [3]. As before, such a well-bracketed string corresponds to a tree. By induction over the height of trees, one can easily show that two well-bracketed strings over { and } represent the same set if and only if the corresponding trees are bisimilar. Hence, the tree bisimulation problem for (unlabelled) trees in term representation is equivalent to the *set equality problem*, which asks whether two such string representations represent the same set. For example $\{\{\}\{\}\}$ and $\{\{\}\}$ represent the same set. From Theorem 7 we obtain the NC^1 upper bound in the following result. The NC^1 lower bound follows from the NC^1-hardness of the bisimulation problem for unlabelled trees in term representation, which is shown in the next section (Theorem 9).

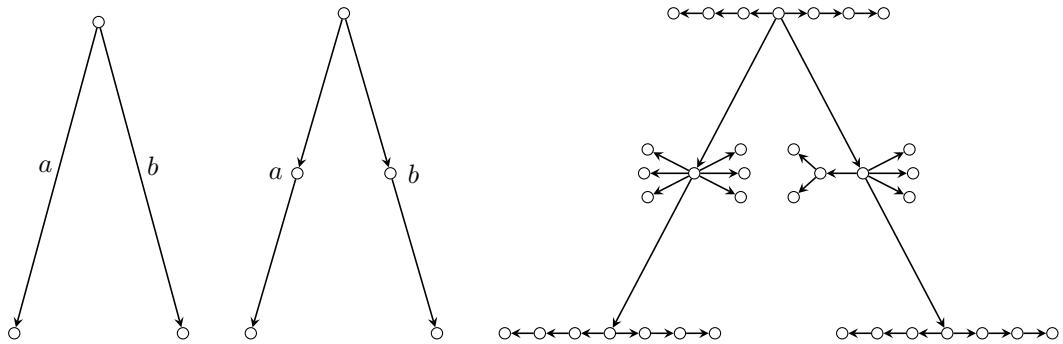


Figure 4 From labelled to unlabelled trees.

▶ **Corollary 8.** *The set equality problem is* NC^1*-complete with respect to* AC^0*-reductions.*

4 Lower Bounds

In this section we prove several lower bounds. In Section 4.1 we prove matching lower bounds for the upper bounds from Corollary 5 and Theorem 7. Finally, in Section 4.2 we show that the simulation problem becomes already P-complete for graphs of bounded path-width.

4.1 Bisimulation on Trees

▶ **Theorem 9.** *The bisimulation problem for unlabelled trees is* L*-hard if the trees are given as pointer structures and* NC^1*-hard if they are given in term representation (both with respect to* AC^0*-reductions).*

Before we prove Theorem 9 let us first show the following lemma:

▶ **Lemma 10.** *The bisimulation problem for edge-labelled trees in term representation (resp., pointer representation) is* AC^0*-reducible to the bisimulation problem for unlabelled trees in term representation (resp., pointer representation).*

Proof. We only show the lemma for the term representation; the same construction also works for the pointer representation. In [18] Srba presents a reduction from the bisimulation problem for edge-labelled graphs to the bisimulation problem for unlabelled graphs. In fact, this construction transforms trees into trees. We slightly modify the reduction to ensure AC^0-computability and assume that there are only two labels, say a and b (which is the case for the trees constructed in the proof of Theorem 9).

Consider a tree T with edge labels a and b. First every labelled edge of T is subdivided into two edges. In Figure 4 (middle tree), the new node added for an x-labelled edge ($x \in \{a, b\}$) is labelled with x. To distinguish the original nodes from the new nodes, we attach to each original node two paths of length 3. To each new node we attach one of two small trees depending on the label of the original edge that is represented by the new node, see the right tree in Figure 4. Let us denote the resulting unlabelled tree with $\mathsf{ul}(T)$. It is not hard to prove that two labelled trees T_1 and T_2 are bisimilar if and only if $\mathsf{ul}(T_1)$ and $\mathsf{ul}(T_2)$ are bisimilar. The proof is basically given in [18].

It remains to prove that the term representation of $\mathsf{ul}(T)$ can be computed in AC^0 from the term representation of T. Consider the term representation t of T. Recall that we identify the opening brackets in t with the nodes of T. We assume that the term representation t

contains the following three opening bracket types: $\texttt{(}_a$ and $\texttt{(}_b$, which represent nodes with an incoming a-labelled (resp. b-labelled) edge, and $\texttt{(}$ for the root. For example, $t = \texttt{(}\texttt{(}_a\texttt{)}\texttt{(}_b\texttt{))}$ is a term representation for the left tree in Figure 4. The transformation $T \mapsto \mathsf{ul}(T)$ can be described by two isometric homomorphisms. A homomorphism $h : \Sigma^* \to \Gamma^*$ is *isometric* if there is an $\ell \geq 1$ such that $|h(c)| = \ell$ for all $c \in \Sigma$. In [16] it is shown that for a given isometric homomorphism $h : \Sigma^* \to \Gamma^*$ and a word $w \in \Sigma^*$ one can compute $h(w)$ in AC^0.[1]

We will proceed in two steps. Define the isometric homomorphism $h_1 : \{\texttt{(}_a, \texttt{(}_b, \texttt{)}\}^* \to \{\texttt{(}_a, \texttt{(}_b, \texttt{(}, \texttt{)}, \texttt{]}\}^*$ by:

$$\texttt{(}_a \mapsto \texttt{(}_a\texttt{(} \qquad \texttt{(}_b \mapsto \texttt{(}_b\texttt{(} \qquad \texttt{)} \mapsto \texttt{)]}$$

Let $u \in \{\texttt{(}_a, \texttt{(}_b, \texttt{)}\}^*$ be the word such that $t = \texttt{(}u\texttt{)}$ and consider the string $\texttt{(}h_1(u)\texttt{)}$. Formally, it is not a term representation (since we have two types of closing brackets). Nevertheless, it describes the tree obtained from T by subdividing every edge and labelling each new node with the former edge label. For example $t = \texttt{(}\texttt{(}_a\texttt{)}\texttt{(}_b\texttt{))}$ is transformed into $\texttt{(}h_1(u)\texttt{)} = \texttt{(}\texttt{(}_a\texttt{()]}\texttt{(}_b\texttt{()]))}$, which describes the node-labelled tree in Figure 4. The second isometric homomorphism $h_2 : \{\texttt{(}_a, \texttt{(}_b, \texttt{(}, \texttt{)}, \texttt{]}\}^* \to \{\texttt{(}, \texttt{)}\}^*$ is defined by:

- $\texttt{(} \mapsto \texttt{((((()))}$ (an opening bracket followed by a path of length 3)
- $\texttt{)} \mapsto \texttt{((()))))}$ (a path of length 3 followed by a closing bracket)
- $\texttt{]} \mapsto \texttt{()()()))}$ (3 leaves followed by a closing bracket)
- $\texttt{(}_a \mapsto \texttt{(()()()}$ (an opening bracket followed by 3 leaves)
- $\texttt{(}_b \mapsto \texttt{((()()))}$ (an opening bracket followed by the tree )

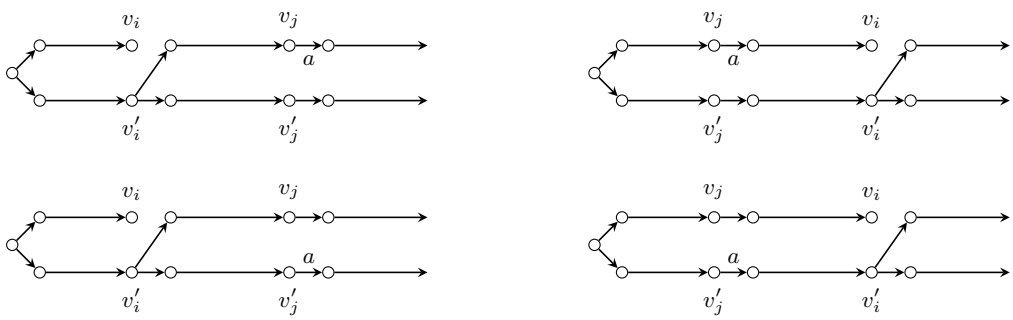
Then, the string $h_2(\texttt{(}h_1(u)\texttt{))}$ is indeed a term representation for the desired unlabelled tree $\mathsf{ul}(T)$. ◀

Proof of Theorem 9. By Lemma 10 it suffices to show the lower bounds for edge-labelled trees. We reuse the proofs from [13], where it is shown that the tree isomorphism problem is L-hard (NC^1-hard, respectively) with respect to AC^0-reductions if the the trees are given as pointer structures (in term representation, respectively). Let us start with the bisimulation problem for trees given in pointer representation. Here, Jenner et al. reduce from the L-complete reachability problem on paths, i.e., the question whether for a given directed path graph G and two nodes $v_i, v_j \in V(G)$, there is a path from v_i to v_j. Without loss of generality, v_i and v_j are distinct and have successors v_{i+1} and v_{j+1}, respectively.

Consider the tree with a root node which has two copies of G as direct subtrees. We refer to nodes of the two copies by $v_1, \ldots, v_n$ and $v'_1, \ldots, v'_n$, respectively. Additionally we replace the edge (v_i, v_{i+1}) by the new edge (v'_i, v_{i+1}). Now let T_1 (resp., T_2) be the tree where the edge (v_j, v_{j+1}) (resp., (v'_j, v'_{j+1})) is labelled by a symbol a (all unlabelled edges are assumed to be labelled with a symbol $b \neq a$). Clearly, T_1 and T_2 can be computed in AC^0 from G. There is a path from v_i to v_j in G if and only if T_1 and T_2 are bisimilar. See Figure 5 for an illustration of the reduction.

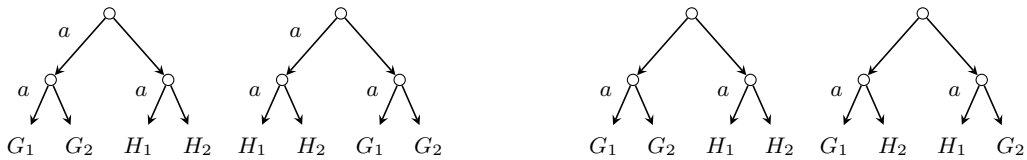
Secondly, Jenner et al. present in [13] an AC^0-reduction from the NC^1-complete evaluation problem of balanced Boolean expressions to the isomorphism problem for trees in term representation. They use the AND-gadget $T_\wedge(G_1, G_2, H_1, H_2)$ and the OR-gadget $T_\vee(G_1, G_2, H_1, H_2)$ which are depicted in Figure 6. Notice that for all trees G_1, G_2, H_1, H_2 the following holds, where $\sim$ can both mean bisimilarity and isomorphism:

[1] If the homomorphism is fixed, this is even possible in NC^0. Moreover, if the homomorphism is not isometric then the problem is TC^0-complete [16].



(a) T_1 and T_2 are bisimilar. **(b)** T_1 and T_2 are not bisimilar.

Figure 5 The two possible forms of T_1, T_2 depending on whether v_j is reachable from v_i or not.



(a) AND-gadget $T_\wedge$ **(b)** OR-gadget $T_\vee$

Figure 6 The trees for the NC^1 lower bound.

- $G_1 \sim H_1$ and $G_2 \sim H_2 \iff T_\wedge(G_1, G_2, H_1, H_2) \sim T_\wedge(H_1, H_2, G_1, G_2)$, and
- $G_1 \sim H_1$ or $G_2 \sim H_2 \iff T_\vee(G_1, G_2, H_1, H_2) \sim T_\vee(G_1, H_2, H_1, G_2)$.

Using the fact that the AND-gadget and the OR-gadget have the same tree structure (if we ignore labels), one can show that the term representations for the resulting trees can be computed in AC^0 from the balanced Boolean expression; see the arguments in [13]. This yields an AC^0-reduction from the evaluation problem of balanced Boolean expressions to the bisimulation problem for trees in term representation. ◀

4.2 Simulation on Graphs of Bounded Path-Width

The results from Corollary 5 and Theorem 7 also hold for the simulation problem. The proof is in fact much easier, since the simulation problem for trees reduces to the evaluation of the Boolean circuit obtained from (1) by removing the second conjunction over all edges $v \xrightarrow{a} v'$; in fact, this circuit is a tree. In this section we show that the simulation problem is P-complete on graphs of bounded path-width, and hence also on graphs of bounded tree-width. It remains open, whether the bisimulation problem for graphs of bounded tree-width belongs to NC or remains P-complete. For integers i, j we use the abbreviation $[i, j] = \{k \in \mathbb{N} \mid i \leq k \leq j\}$.

▶ **Theorem 11.** *There is a number k such that the following problem is P-complete: Given a graph G of path-width at most k and two nodes $u, v \in V(G)$, does u simulate v?*

Proof. Fix a P-complete language $L \subseteq \{0,1\}^*$ and a deterministic polynomial time bounded Turing machine $M = (Q, \Gamma, \{0,1\}, q_0, q_f, \delta)$ that accepts L. Here Q is the set of states, $\Gamma \supseteq \{0, 1, \Box\}$ is the tape alphabet ($\Box$ is the blank symbol), q_0 is the initial state, q_f is the final state, and $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\rightarrow, \leftarrow\}$ is the transition function, where $\rightarrow$ and $\leftarrow$ indicate the head direction. The machine has a single tape, whose cells are indexed with

integers. Initially, the input x is written in cells $0, \ldots, |x|-1$ and the tape head scans cell 0. We can assume that there is a polynomial $p(n)$ such that for every input $x \in \{0,1\}^*$ we have: $x \in L$ if and only if after $p(|x|)$ many transitions the machine is in state q_f, cell 0 contains $\Box$, and the tape head scans cell 0.

We can view configurations of M as words from $\Gamma^*(Q \times \Gamma)\Gamma^*$. Let $\Omega = \Gamma \cup (Q \times \Gamma)$. We define a partial mapping $\Delta : \Omega^3 \to \Omega$ as follows, where $a, a', b, c \in \Gamma$, $p, q \in Q$.

- $\Delta(a, b, c) = b$
- $\Delta(b, c, (q, a)) = (p, c)$ if $\delta(q, a) = (p, a', \leftarrow)$
- $\Delta(b, c, (q, a)) = c$ if $\delta(q, a) = (p, a', \rightarrow)$
- $\Delta(b, (q, a), c) = a'$ if $\delta(q, a) = (p, a', d)$ for some $d \in \{\rightarrow, \leftarrow\}$
- $\Delta((q, a), b, c) = b$ if $\delta(q, a) = (p, a', \leftarrow)$
- $\Delta((q, a), b, c) = (p, b)$ if $\delta(q, a) = (p, a', \rightarrow)$

In all other cases, Δ is undefined. The mapping Δ computes from the three symbols at positions $i-1$, i, $i+1$ in a configuration the symbol at position i in the successor configuration.

Let us fix an input $x = a_0 a_1 \cdots a_{n-1}$ of length $n > 0$ for the machine M and let $N = p(n) + 1$. Then there exists a unique computation of M on input x. We denote with C the corresponding computation table. Formally, it is a mapping $C : [-N, N] \times [0, N-1] \to \Omega$, where $C(i, t)$ is the symbol at cell i in the t-th configuration. It can be defined by the following properties:

- $C(0, 0) = (a_0, q_0)$, $C(i, 0) = a_i$ for $i \in [1, n-1]$, $C(i, 0) = \Box$ for $i \in [-N, N] \setminus [0, n-1]$,
- $C(-N, t) = C(N, t) = \Box$ for all $t \in [0, N-1]$
- $C(i, t) = \Delta(C(i-1, t-1), C(i, t-1), C(i+1, t-1))$ for all $t \in [1, N-1]$, $i \in [-N+1, N-1]$.

Let us fix the set of edge labels $A = \{-1, 0, 1, \alpha\} \uplus \Omega$. We define two edge-labelled graphs P (for position) and T (for time) with edge labels from A and the node sets

$$V(P) = [-N, N] \times \{0, 1\}, \quad V(T) = [0, N-1] \ \cup \ [0, N-1] \times \Omega \ \cup \ [0, N-1] \times \Omega^3.$$

For better readability, we write edges of P (resp., T) as $x \xrightarrow{a}_P y$ (resp., $x \xrightarrow{a}_T y$). Then, P and T contain the following edges:

$$(i, 0) \xrightarrow{\alpha}_P (i, 1) \text{ for all } i \in [-N, N]$$

$$(i, 1) \xrightarrow{\delta}_P (i + \delta, 0) \text{ for all } i \in [-N, N], \delta \in \{-1, 0, 1\} \text{ with } i + \delta \in [-N, N]$$

$$(i, 0) \xrightarrow{C(i,0)}_P (i, 0) \text{ for all } i \in [-N, N]$$

$$(t, a) \xrightarrow{\alpha}_T (t-1, b, c, d) \text{ for all } a \in \Omega, t \in [1, N-1], (b, c, d) \in \Delta^{-1}(a)$$

$$(t, a_{-1}, a_0, a_1) \xrightarrow{\delta}_T (t, a_\delta) \text{ for all } t \in [0, N-2], a_{-1}, a_0, a_1 \in \Omega, \delta \in \{-1, 0, 1\}$$

$$(t, a) \xrightarrow{b}_T t \text{ for all } a, b \in \Omega, t \in [1, N-1]$$

$$(0, a) \xrightarrow{b}_T 0 \text{ for all } a \in \Omega, b \in \{a, \alpha\}$$

$$t \xrightarrow{a}_T t \text{ for all } t \in [0, N-1], a \in A$$

An example of the construction is shown in Figure 7, where we assume $N = 3$ for simplicity. It is easy to see that both P and T (and hence also the disjoint union of P and T) have bounded path-width. More precisely, P has path-width 3 (the bags are the sets $\{(i, 0), (i, 1), (i+1, 0), (i+1, 1)\}$ for $-N \leq i \leq N-1$), whereas the path-width of T is bounded by $|\Omega|^3 + |\Omega|$ (the bags are the set $\{t, (t, a), (t-1, b, c, d) \mid a, b, c, d \in \Omega\}$ for $1 \leq t \leq N-1$ and $\{(t, a), (t, b, c, d) \mid a, b, c, d \in \Omega\}$ for $0 \leq t \leq N-2$). Recall that $|\Omega|$ is a fixed constant since the machine M is fixed.

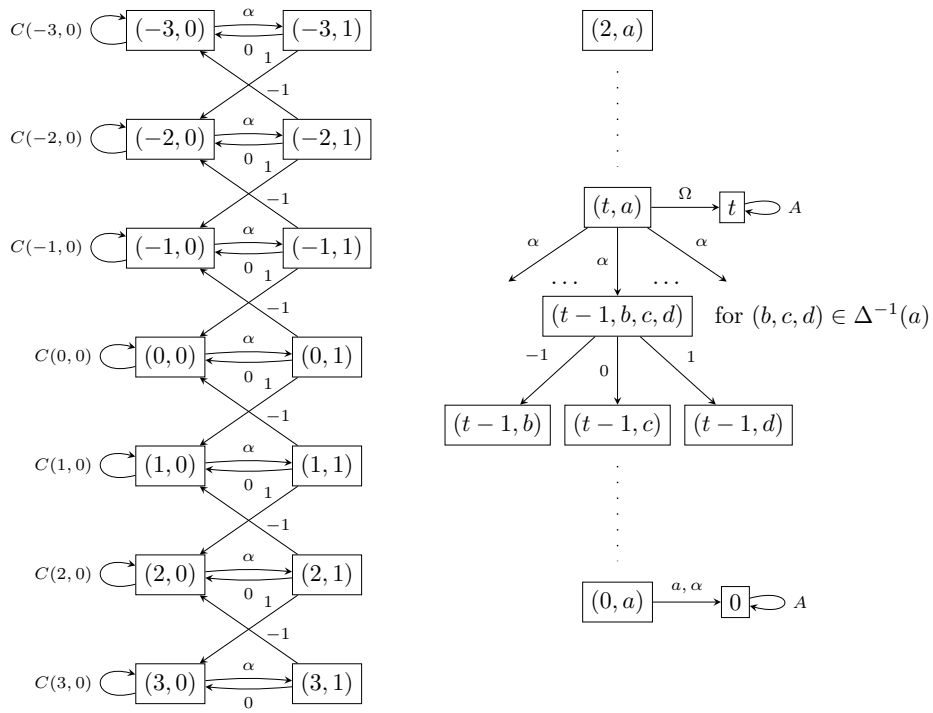


Figure 7 The edge-labelled graphs P and T, where we assume $N = 3$ for simplicity.

For nodes $u \in V(P)$, $v \in V(T)$ we write $u \sqsubseteq v$ if u is simulated by v in the disjoint union of P and T. The following claim proves the theorem, since M accepts x if and only if $C(0, N-1) = (q_f, \Box)$ by our assumptions on M.

▶ **Claim.** *For all $t \in [0, N-1]$, $i \in [-N, N]$ and $a \in \Omega$ such that $i + t \leq N$ and $i - t \geq -N$ we have: $C(i,t) = a$ if and only if $(i,0) \sqsubseteq (t,a)$.*

Proof of Claim. We prove the claim by induction on t. First, note that for every $v \in V(P)$ and every $t \in [0, N-1] \subseteq V(T)$ we have $v \sqsubseteq t$, since at t we can loop with every $a \in A$. For $t = 0$ note that indeed $(i,0) \sqsubseteq (0, C(i,0))$: The only outgoing edges for $(i,0)$ are labelled with $C(i,0)$ and α. From $(0, C(i,0))$ these labels lead to node 0, which simulates every node of P. On the other hand, if $a \neq C(i,0)$, then $(i,0)$ has a $C(i,0)$-labelled outgoing edge, whereas $(0,a)$ has no such outgoing edge. This implies $(i,0) \not\sqsubseteq (0,a)$.

Now assume that $t \in [1, N-1]$, $i \in [-N, N]$, $i + t \leq N$, and $i - t \geq -N$ and that the claim holds for $t - 1$. First assume that $C(i,t) = a$. Since $t \geq 1$, we have $i + 1 \leq N$ and $i - 1 \geq -N$, i.e., $i \in [-N+1, N-1]$. We have to show that $(i,0) \sqsubseteq (t,a)$. The edge $(i,0) \xrightarrow{C(i,0)}_P (i,0)$ can be simulated by the edge $(t,a) \xrightarrow{C(i,0)}_T t$ (recall that $(i,0) \sqsubseteq t$). Now consider the other possible edge $(i,0) \xrightarrow{\alpha}_P (i,1)$. Since $C(i,t) = a$, there must exist $(b,c,d) \in \Delta^{-1}(a)$ such that $b = C(i-1, t-1)$, $c = C(i, t-1)$, and $d = C(i+1, t-1)$. Also note that $i + \delta + (t-1) \leq N$ and $i + \delta - (t-1) \geq -N$ for all $\delta \in \{-1, 0, 1\}$. Hence, by induction $(i-1,0) \sqsubseteq (t-1,b)$, $(i,0) \sqsubseteq (t-1,c)$, and $(i+1,0) \sqsubseteq (t-1,d)$. But this implies that $(i,1) \sqsubseteq (t-1,b,c,d)$. Hence, we can choose the edge $(t,a) \xrightarrow{\alpha}_T (t-1,b,c,d)$ in order to simulate the edge $(i,0) \xrightarrow{\alpha}_P (i,1)$.

Finally, assume that $C(i,t) \neq a$. We have to show that $(i,0) \not\sqsubseteq (t,a)$. Let us choose the edge $(i,0) \xrightarrow{\alpha}_P (i,1)$. We have to show that for every $(b,c,d) \in \Delta^{-1}(a)$, $(i,1) \not\sqsubseteq (t,b,c,d)$.

Let us fix a triple $(b, c, d) \in \Delta^{-1}(a)$. Since $C(i, t) \neq a$, one of the following three statements holds: $C(i-1, t-1) \neq b$, $C(i, t-1) \neq c$, $C(i+1, t-1) \neq d$. Hence, by induction, $(i-1, 0) \not\sqsubseteq (t-1, b)$ or $(i, 0) \not\sqsubseteq (t-1, c)$ or $(i+1, 0) \not\sqsubseteq (t-1, d)$. This implies that, indeed, $(i, 1) \not\sqsubseteq (t, b, c, d)$. ◀

◀

It seems to be difficult to modify the above proof so that it shows P-hardness for bisimulation on graphs of bounded path-width or bounded tree-width. One might try to restrict the choices of the players in the bisimulation game (see e.g. [1]) so that they are forced to play as in the simulation game. There is a technique to achieve this (defenders forcing) but the problem is that it yields grid-like graph structures and hence graphs of unbounded tree-width.

5 Conclusion

We proved the following results:

- The bisimulation problem for trees that are given by pointer structures (resp., in term representation) is complete for deterministic logspace (resp. NC^1). These results also hold for the simulation problem for trees.
- Already for graphs of bounded path-width (a subclass of the graphs of bounded tree-width), the simulation problem becomes P-complete.

As an application of the first result, we showed that equality of hereditarily finite sets is NC^1-complete. For the proofs we introduced the new class of tree-shaped circuits and proved that the circuit evaluation problem for tree-shaped circuits of bounded width is in logspace or NC^1, depending on the representation of the circuit. It would be nice to find further applications of these circuits. The main open problem that remains is whether the bisimulation problem for graphs of bounded tree-width is in NC or P-complete.

References

1. L. Aceto and A. Ingólfsdóttir. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press, 2007.
2. J. L. Balcázar, J. Gabarró, and M. Santha. Deciding bisimilarity is P-complete. *Form. Asp. Comput.*, 4(6A):638–648, 1992.
3. D. A. M. Barrington and J. Corbet. On the relative complexity of some languages in NC^1. *Inform. Process. Lett.*, 32:251–256, 1989.
4. S. R. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In *Kurt Gödel Colloquium 97*, pages 18–33, 1997.
5. R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.
6. M. Elberfeld, A. Jakoby, and T. Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of FOCS 2010*, pages 143–152. IEEE, 2010.
7. M. Elberfeld, A. Jakoby, and T. Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *Proceedings of STACS 2012*, volume 14 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012.
8. M. Elberfeld and P. Schweitzer. Canonizing graphs of bounded tree width in logspace. In *Proceedings of STACS 2016*, volume 47 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.
9. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.

10 M. Grohe, B. Grußien, A. Hernich, and B. Laubner. L-recursion and a new logic for logarithmic space. *Log. Meth. Comput. Sci.*, 9(1), 2012.

11 F. Gurski and E. Wanke. The tree-width of clique-width bounded graphs without $K_{n,n}$. In *Proceedings of WG 2000*, volume 1928 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000.

12 W. Hesse, E. Allender, and D. A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65:695–716, 2002.

13 B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.

14 M. Kaminski, V. V. Lozin, and M. Milanic. Recent developments on graphs of bounded clique-width. *Discrete Appl. Math.*, 157(12):2747–2761, 2009.

15 P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inform. Comput.*, 86(1):43–68, 1990.

16 K.-J. Lange and P. McKenzie. On the Complexity of Free Monoid Morphisms. In *Proceedings of ISAAC'98*, volume 1533 of *Lecture Notes in Computer Science*, pages 247–256. Springer, 1998.

17 S. Lindell. A logspace algorithm for tree canonization (extended abstract). In *Proceedings of STOC 1992*, pages 400–404. ACM, 1992.

18 J. Srba. On the power of labels in transition systems. In *Proceedings of CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 277–291. Springer, 2001.

19 H. Vollmer. *Introduction to Circuit Complexity*. Springer, 1999.