Pure Pointer Programs and Tree Isomorphism[⋆]

Martin Hofmann, Ramyaa Ramyaa, and Ulrich Schöpp

Ludwig-Maximilians Universität München
Oettingenstraße 67, 80538 Munich, Germany
{mhofmann,ramyaa,schoepp}@tcs.ifi.lmu.de

Abstract. In a previous work, Hofmann and Schöpp have introduced the programming language PURPLE to formalise the common intuition of LOGSPACE-algorithms as pure pointer programs that take as input some structured data (e.g. a graph) and store in memory only a constant number of pointers to the input (e.g. to the graph nodes). It was shown that PURPLE is strictly contained in LOGSPACE, being unable to decide st-connectivity in undirected graphs.
In this paper we study the options of strengthening PURPLE as a manageable idealisation of computation with logarithmic space that may be used to give some evidence that PTIME-problems such as Horn satisfiability cannot be solved in logarithmic space.
We show that with counting, PURPLE captures all of LOGSPACE on locally ordered graphs. Our main result is that without a local ordering, even with counting and nondeterminism, PURPLE cannot solve tree isomorphism. This generalises the same result for Transitive Closure Logic with counting, to a formalism that can iterate over the input structure, furnishing a new proof as a by-product.

1 Introduction

In a previous work we have introduced the Pure Pointer Language PURPLE [9], which captures the intuitive idea of accessing read only graph-like input data via pointers as an abstract datatype. In addition to the usual control structures PURPLE provides a way of iterating over all nodes of the input graph in an unspecified order. PURPLE programs can be evaluated in LOGSPACE, but are strictly weaker, for example because counting cannot be defined [9]. One can thus hope to prove strict separations between PURPLE and PTIME, which could then be seen as additional evidence for the assumed inequality LOGSPACE $\neq$ PTIME. We have already shown that PURPLE strictly subsumes deterministic transitive closure (DTC) logic with local ordering and that it cannot in general decide connectivity of undirected (locally ordered) graphs [8]. As a consequence, the same follows for DTC with local ordering, which was open until then. This trivially shows that PURPLE programs cannot solve PTIME-complete problems, such as Horn satisfiability, because reachability is an instance thereof.

Unfortunately, such a result cannot be seen as evidence for LOGSPACE $\neq$ PTIME because reachability in undirected graphs is in LOGSPACE by Reingold's theorem. Thus, the way forward would consist of strengthening PURPLE so as to still remain within LOGSPACE or at least POLY-LOGSPACE, yet in such a way that reachability does become

⋆ This work was supported by Deutsche Forschungsgemeinschaft (DFG) under grant PURPLE.

definable, whereas Horn satisfiability demonstrably is not. While we are not yet in a position to offer such an extension, we report in this paper considerable progress with pinning down its possible form. A reasonable attempt borrowed from finite model theory consists of adding arithmetic variables ranging over the size of the input ("counting"). We show, however, that with a local-ordering (edges of input graph navigable bidirectionally) such an extension captures all of LOGSPACE. If the edges of the input graph can be followed only in one direction (an analogue of "1LO" in finite model theory) then counting, even in the presence of nondeterminism, does not capture LOGSPACE, because the tree isomorphism problem, which is in LOGSPACE, cannot be solved. This constitutes the main technical result of this paper. It implies a similar result for TC logic, which was known, but proved in a completely different fashion. Our method is based on bisimulation, while the proof for TC logic uses Ehrenfeucht-Fraïssé (EF) games [5].

The iteration construct of PURPLE subsumes first-order quantification but seems to be stronger. Despite some effort on our side it was not possible to extend the EF-method to PURPLE. The method of bisimulation, on the other hand, did do the trick.

While we believe that PURPLE with nondeterminism and counting is strictly stronger than TC logic with counting, it is difficult to come up with an explicit example, for this would have to be between LOGSPACE and TC logic with counting, but tree isomorphism is the only known such example.

We can, however, notice that PURPLE is able to traverse the nodes of a tree in some (non reproducible) breadth-first ordering whereas TC apparently can not. Whether this results in a clean separation we do not presently know, but we may note that, if we replace breadth-first with depth-first then, being able to traverse *is* strictly stronger than being able to quantify which means that PURPLE-style traversal is not a mere reformulation of quantification: If we assume that we can traverse the nodes of a tree in depth first order for some (not necessarily reproducible order between siblings) then we can implement Lindell's algorithm and thus solve tree isomorphism, and this is in fact what happens in [7]. On the other hand, TC logic is able to, for example, count the number of nodes that lie between two given nodes in *any* depth-first ordering.

Related work. The first attempt at a "relativised separation" by permitting access to input data only via an abstract interface are Cook & Rackoff's results on jumping automata on graphs (JAGs) [3], which show that no finite automaton being able to place pebbles on a graph and moving these along edges can solve undirected st-connectivity. At that time this problem was probably believed to be not in LOGSPACE unless NLOGSPACE $=$ LOGSPACE, so that such result might have been seen as evidence for NLOGSPACE $\neq$ LOGSPACE. Reingold's LOGSPACE algorithm [15] for undirected st-connectivity has changed this picture and consequently it has been shown that it suffices to extend JAGs with counting in order to encode this algorithm [14]. In our own work on PURPLE we extended Cook & Rackoff's result by allowing such JAGs to visit every node of the input albeit in an unspecified order [9]. We [8] were able to show that st-connectivity remains unsolvable in such a framework thus solving an open question about the strength of transitive closure logic on locally ordered structures [6]. The latter is an alternative to the automata-theoretic approach to restricting the access to the input. There, and more generally in finite model theory [4, 10], "programs" are formulas of some logic to be evaluated over the input represented as a logical structure.

When it comes to relativisation many people first think of oracles; however just as in the case of P vs. NP also the oracle-relativised versions of LOGSPACE VS. PTIME can go either way depending on the choice of oracle [12], which, as usual, shows that techniques such as diagonalization that work in the presence of oracles cannot possibly lead to a full separation of LOGSPACE and PTIME. Incidentally, we may remark here that the proof techniques we use for PURPLE, namely Cook and Rackoff's [3] radius bounds that we generalized in [8] and bisimulation as in this paper do *not* "relativize" in the classical sense. But, of course, we nevertheless do not claim that we achieve a full separation with them! Another approach from general complexity theory that should be mentioned is [2], which shows that the existence of P-complete sparse sets implies LOGSPACE $=$ PTIME.

Finite model theory has a related but slightly different goal to ours. One also seeks to access the input, e.g. a graph, in an abstract fashion; yet the ultimate goal is to capture complexity classes, i.e. to define a logic in which despite the limited access all and only the properties decidable in a given complexity class are definable. In general, this is possible if a total ordering on the elements (nodes) of the input is available or definable.

Once a complexity class is "captured" by a formalism it becomes next to useless for relativised separation, for then the relativised separation is as hard as full unrelativised separation. This also applies to several alternative characterisations of LOGSPACE by programming language methods, such as [1, 11]. As far as we know PURPLE is the first programming formalism for a proper, yet nontrivial subset of LOGSPACE which can therefore be used as a vehicle for relativised separation.

More specific to the technical contribution of this paper as opposed to PURPLE as a whole, we mention that the addition of arithmetical variables ranging over the size of the input structure called "counting" is a popular device for strengthening such logics and it came as a surprise that despite the obvious power to simulate log-sized work-tapes, counting still does not render (deterministic) transitive closure logic equivalent to (N)LOGSPACE. This is because despite Lindell's intriguing LOGSPACE algorithm [13], isomorphism of unordered trees has been shown not to be definable in transitive closure logic with counting [5]. Recently, this has led to the introduction of a new recursion principle allowing the implementation of Lindell's algorithm in logic [7].

2 PURPLE and extensions

PURPLE programs are parameterised by a finite set L of labels and a finite set S of predicate symbols. Each predicate symbol p is assumed to have a finite arity $\mathit{ar}(p) \in \mathbb{N}$.

The input of a program is a pointer structure, which interprets the labels and predicates: A *pointer structure* on L and S ((L, S)-model, for short) specifies a finite set U as a universe, a function $[\![l]\!]\colon U \to U$ for each label $l \in L$ and a set $[\![p]\!] \subseteq U^{\mathit{ar}(p)}$ for each predicate symbol p.

Relational structures, such as graphs, are special cases of such pointer structures. For graphs one would use an empty set of labels and a single binary predicate $\mathit{edge}(x, y)$. In some cases, pointer structures are arguably more natural than relational structures, however. For example bounded degree graphs with a one-way local ordering (1LO), in which the edges emanating from each node are ordered, are represented naturally using one sort V and labels $\mathtt{succ}_i$, where i ranges from 1 to the degree of the graph. For a two-way local ordering (2LO), which also orders the incoming edges at each node, one may in addition use labels of the form $\mathtt{pred}_i$.

A program with labels L and predicate symbols S can access its input structure through the following terms for pointers to elements of the universe and for booleans.

$$\begin{aligned} t^U &::= x^U \mid t^U.l \text{ for any label } l \in L \\ t^{\mathtt{bool}} &::= x^{\mathtt{bool}} \mid \neg t^{\mathtt{bool}} \mid t_1^{\mathtt{bool}} \wedge t_2^{\mathtt{bool}} \mid t_1^U = t_2^U \mid p(x_1^U, \ldots, x_{ar(p)}^U) \text{ for any } p \in S \end{aligned}$$

We call x^U pointer variables. The intention is that $t^U.l$ is interpreted by $[\![l]\!](t^U)$.

The programs themselves are given by the grammar.

$$\begin{aligned} \mathit{Prg} \ ::= \ & \mathtt{skip} \mid \mathit{Prg}_1; \mathit{Prg}_2 \mid x^U := t^U \mid x^{\mathtt{bool}} := t^{\mathtt{bool}} \\ & \mid \mathtt{if}\ t^{\mathtt{bool}}\ \mathtt{then}\ \mathit{Prg}_1\ \mathtt{else}\ \mathit{Prg}_2 \mid \mathtt{forall}\ x^U\ \mathtt{do}\ \mathit{Prg} \end{aligned}$$

We do not include a `while`-loop, as it can be defined from the `forall`-loop, see [9]. We write $\mathtt{if}\ t^{\mathtt{bool}}\ \mathtt{then}\ \mathit{Prg}$ for $\mathtt{if}\ t^{\mathtt{bool}}\ \mathtt{then}\ \mathit{Prg}\ \mathtt{else}\ \mathtt{skip}$.

A *configuration* $\langle \rho, q \rangle$ consists of a *pebbling* ρ and a *state* q. The pebbling ρ maps pointer variables (which we also call pebbles) to elements of the universe U. The state q is a function from boolean variables to booleans. Given a configuration I, we define an interpretation of the terms $[\![t^{\mathtt{bool}}]\!]_I \in \{\mathtt{true}, \mathtt{false}\}$ and $[\![t^U]\!]_I \in U$ in the usual way.

A big-step reduction relation $\mathit{Prg} \vdash_M I \longrightarrow O$ between configurations I and O on some (L, S)-model M and a program Prg is defined inductively by the following clauses:

- $\mathtt{skip} \vdash_M I \longrightarrow I$.
- $\mathit{Prg}_1; \mathit{Prg}_2 \vdash_M I \longrightarrow O$ if $\mathit{Prg}_1 \vdash_M I \longrightarrow R$ and $\mathit{Prg}_2 \vdash_M R \longrightarrow O$ for some R.
- $x^U := t^U \vdash_M \langle \rho, q \rangle \longrightarrow \langle \rho[x \mapsto [\![t]\!]_{\langle \rho, q \rangle}], q \rangle$.
- $x^{\mathtt{bool}} := t^{\mathtt{bool}} \vdash_M \langle \rho, q \rangle \longrightarrow \langle \rho, q[x \mapsto [\![t]\!]_{\langle \rho, q \rangle}] \rangle$.
- $\mathtt{if}\ t\ \mathtt{then}\ \mathit{Prg}_1\ \mathtt{else}\ \mathit{Prg}_2 \vdash_M I \longrightarrow O$ if $[\![t]\!]_I = \mathtt{true}$ and $\mathit{Prg}_1 \vdash_M I \longrightarrow O$.
- $\mathtt{if}\ t\ \mathtt{then}\ \mathit{Prg}_1\ \mathtt{else}\ \mathit{Prg}_2 \vdash_M I \longrightarrow O$ if $[\![t]\!]_I = \mathtt{false}$ and $\mathit{Prg}_2 \vdash_M I \longrightarrow O$.
- $\mathtt{forall}\ x^U\ \mathtt{do}\ \mathit{Prg}_1 \vdash_M I \longrightarrow O$ if there exists an enumeration $u_1, u_2, \ldots, u_n$ of $[\![U]\!]$ and configurations $I = \langle \rho_1, q_1 \rangle, \langle \rho_2, q_2 \rangle, \ldots, \langle \rho_{n+1}, q_{n+1} \rangle = O$, such that $\mathit{Prg}_1 \vdash_M \langle \rho_k[x \mapsto u_k], q_k \rangle \longrightarrow \langle \rho_{k+1}, q_{k+1} \rangle$ holds for all $k \in \{1, \ldots, n\}$.

When the model M is clear from the context we may omit the subscript.

In order for a PURPLE program to accept (resp. reject) an input, it must do so no matter what enumerations are chosen for its `forall`-loops. This is defined formally in the next definition, in which we use a boolean variable *result* to indicate acceptance.

Definition 1. *A program Prg* accepts *(resp. rejects) an (L, S)-model M if $\mathit{Prg} \vdash_M \langle \rho, q \rangle \longrightarrow \langle \rho', q' \rangle$ implies $q'(\mathit{result}) = \mathtt{true}$ (resp. $q'(\mathit{result}) = \mathtt{false}$) for all ρ, ρ', q and q'.*

A program Prg *recognises* a set X of (L, S)-models if it accepts any model in X and rejects all others. Note that a program may neither accept nor reject its input, namely if for some runs it returns `true` and for others it returns `false`. To put it simply, a PURPLE program for some problem X should give the correct answer, be it `true` or `false` for any given input and independent of the run, i.e. the traversal sequences chosen.

We also note that predicate symbols, which were not part of the original definition of PURPLE [9], are there just for notational convenience and do not add expressive power. Unary predicates can be modelled with an extra pointer that points to designated nodes for "true" and "false". A binary relation can be modelled by introducing an extra node for each pair of related nodes with pointers *fst* and *snd* pointing to the latter two nodes. One uses a unary predicate to differentiate between actual nodes and these helper nodes.

2.1 Counting

PURPLE cannot solve counting problems, as shown in [9]. So, one obvious addition to PURPLE is counting. Here we extend PURPLE by counting variables ("counters"), each of which can hold a number from 0 to the size of the input structure's universe, and we extend the terms with arithmetic operations:

$$t^{\texttt{bool}} ::= \cdots \mid \mathit{iszero}(t^{\texttt{count}}) \qquad t^{\texttt{count}} ::= x^{\texttt{count}} \mid \mathit{max} \mid \mathit{pred}(t^{\texttt{count}})$$

We extend the operational semantics such that the state q now not only maps boolean variables to booleans, but also counting variables to numbers.

PURPLE with counters (PURPLE$_c$) can do arithmetic: the complement of a counter can be computed using *max* and repeated decrement; the increment can be implemented using double complementation and decrement; The rest of the operations follow by repeated applications of these. PURPLE$_c$ can count the number of tuples of nodes satisfying any PURPLE-definable property, and so can simulate counting quantifiers used in (D)TC logics.

Lemma 1. PURPLE$_c$ *captures* LOGSPACE *on graphs with a two-way local ordering, represented as pointer structures as described above.*

Proof (Outline). PURPLE captures all of LOGSPACE on ordered graphs. So, it suffices to show that a total ordering can be defined on any input graph.

To this end we note that given any graph node n, PURPLE$_c$ can define a total ordering of the weakly connected component containing n. We use Reingold's algorithm for undirected st-connectivity, which checks for connectivity by enumerating all nodes in the weakly connected component of s and checking if t appears therein. This algorithm can be implemented by RAMJAGs [14], and so, by an easy translation, also in PURPLE$_c$. In this way, PURPLE$_c$ can order the nodes of the weakly connected component according to their order of their first appearance in the enumeration.

In their proof that TC-logic with counting captures NLOGSPACE on graphs with a two-way local ordering [5], Etessami and Immerman have shown how a total ordering can be defined using counting from such orderings of the weakly connected components. This argument can be adapted to PURPLE$_c$ to complete the proof.

2.2 Nondeterminism

Another way to extend the power of PURPLE is to add nondeterminism. This gives PURPLE the power to decide st-connectivity (reachability) even for directed graphs and in the presence of counting also the power of deciding non-reachability by Immerman-Szelepcsenyi. PURPLE-programs with non-determinism can be evaluated in NLOGSPACE but (probably) not LOGSPACE.

Adding nondeterminism is not completely straightforward, as we must separate nondeterministic choices from the choices made in the evaluation of `forall`-loops. We would like to allow programs to make nondeterministic choices, while still maintaining that their acceptance behaviour is independent of the enumerations chosen in the `forall`-loops.

PURPLE with nondeterminism (PURPLE$_{nd}$) has a command for nondeterministic choice:

$$\mathit{Prg} ::= \ldots \mid \texttt{choose}\ \mathit{Prg}_1\ \texttt{or}\ \mathit{Prg}_2$$

To define the semantics of PURPLE with nondeterminism, we amend the notion of configuration so that it now consists of a triple $\langle \rho, q, \sigma \rangle$, where ρ and q are a pebbling and a state as before and σ is an infinite list enumerations of the universe U. This new component σ specifies the runs of all future `forall` loops. Therefore, in the definition of $(\texttt{forall}\ x^U\ \texttt{do}\ \mathit{Prg}) \vdash \langle \rho, q, \sigma \rangle \longrightarrow \langle \rho', q', \sigma' \rangle$ we do not use an arbitrary enumeration of U, but we take the first enumeration $u_1, \ldots, u_n$ from σ. The subsequent computation uses the list $\mathit{tail}(\sigma)$ of the remaining enumerations from σ. That is, we require there to be configurations $\langle \rho, q, \mathit{tail}(\sigma) \rangle = \langle \rho_1, q_1, \sigma_1 \rangle, \ldots, \langle \rho_{n+1}, q_{n+1}, \sigma_{n+1} \rangle = \langle \rho', q', \sigma' \rangle$ such that $\mathit{Prg} \vdash \langle \rho_k[x \mapsto u_k], q_k, \sigma_k \rangle \longrightarrow \langle \rho_{k+1}, q_{k+1}, \sigma_{k+1} \rangle$ holds for all $k \in \{1, \ldots, n\}$. For the semantics of the new term, we stipulate $\texttt{choose}\ \mathit{Prg}_1\ \texttt{or}\ \mathit{Prg}_2 \vdash I \longrightarrow O$ if $\mathit{Prg}_1 \vdash I \longrightarrow O$ or $\mathit{Prg}_2 \vdash I \longrightarrow O$. In all other cases, σ is merely passed on. E.g. $x := t \vdash \langle \rho, q, \sigma \rangle \longrightarrow \langle \rho[x \mapsto [\![t]\!]_I], q, \sigma \rangle$.

With these provisos, we can make the role of the two kinds of choices precise and define when a nondeterministic program accepts an input:

Definition 2. *A nondeterministic program Prg* accepts *an (L, S)-model M if for all I there exists O with Prg $\vdash_M I \longrightarrow O$ and $O(\mathit{result}) =$* `true`. *It rejects M if for all I and for all O with Prg $\vdash_M I \longrightarrow O$ one has $O(\mathit{result}) =$* `false`.

Thus, in the positive case, M must find, for all traversals of the `forall`-loops, appropriate nondeterministic choices leading to result `true`. In the negative case, however, the program must yield result `false` no matter how the nondeterministic choices are made and how the `forall`-loops are being traversed.

Note that for programs without `choose`, this definition agrees with the one for PURPLE above. For programs without `forall`-loop it agrees with the standard definition of nondeterminism.

In [9] we have shown that PURPLE can evaluate the formulae of DTC-logic. One may expect that with nondeterminism, this result generalises to TC-logic. Indeed, the TC-operator itself can be evaluated by nondeterministic PURPLE much like the DTC-operator by deterministic PURPLE. However, with nondeterminism it becomes harder to evaluate negations of TC-formulae. To be able to do so, we add counting as well, so that we can implement Immerman-Szelepcsenyi's algorithm for complementation in NL. We refer to PURPLE with counting and nondeterminism by PURPLE$_{c,nd}$.

We obtain the following proposition. Recall that any relational structure M can be understood as a pointer structure.

Lemma 2. *For each closed TC-formula φ on a relational signature Σ, there exists a program P_φ in* PURPLE$_{c,nd}$ *such that, for any Σ-structure of Σ, $M \models \varphi$ holds if and only if P_φ recognises M.*

3 Preliminaries

We first define AB-trees, a family of leaf-coloured trees. (These are modified versions of the trees presented in [5]). AB-trees are defined without any ordering, in particular there is no ordering on the children of any node. For each k, there are exactly two AB-trees A_{2k} and B_{2k} of height $2k$, and exactly three AB-trees C_{2k+1}, D_{2k+1} and E_{2k+1} of height $2k+1$. We define these trees by induction on k.

– The tree A_0 is a leaf with colour A and B_0 is a leaf with colour B.
– For any k, the tree C_{2k+1} (resp. E_{2k+1}) has six immediate subtrees: four of A_{2k} (resp. B_{2k}) and two of B_{2k} (resp. A_{2k}).
– For any k, the tree D_{2k+1} has six immediate subtrees: three each of A_{2k} and B_{2k}.
– For $k > 0$, the tree A_{2k} has eight immediate subtrees: three each of E_{2k-1} and C_{2k-1}, and two of D_{2k-1}.
– For $k > 0$, the tree B_{2k} has eight immediate subtrees: two each of E_{2k-1} and C_{2k-1}, and four of D_{2k-1}.

While an AB-tree only comes with a colouring of its leaves, we can consider it as a coloured graph in which each node is assigned a colour from the set $\{A, B, C, D, E\}$. Write $ht(n)$ for the height of a node n, measured from leaf nodes, which have height 0. A node n has colour A, B, C, D or E respectively if the subtree rooted at this node is $A_{ht(n)}$, $B_{ht(n)}$, $C_{ht(n)}$, $D_{ht(n)}$ or $E_{ht(n)}$ respectively. With this definition, each node n in an AB-tree has a uniquely defined colour; we denote it by $colour(n)$.

A *structural isomorphism* between two AB-trees is a bijection between the nodes of the two trees that preserves the tree structure, but that need not preserve colours. In contrast, a *colour isomorphism* must preserve both structure and colours. For any $X, Y \in \{A, B, C, D, E\}$ and any k the trees X_k and Y_k are structurally isomorphic. They are (colour) isomorphic iff $X = Y$. AB-trees have many isomorphic subtrees, and in particular, have the following properties.

Lemma 3. *For all nodes n, m with children $n_1, \ldots, n_t$ and $m_1, \ldots, m_t$:*
1. For any $i, j, k : \{1, \ldots, t\}$ with $i \neq j$ there is a bijection $f : \{1, \ldots, t\} \to \{1, \ldots, t\}$ with $f(i) = k$ and $colour(n_j) = colour(m_{f(j)})$.
2. For any $i, j : \{1, \ldots, t\}$, there is a bijection $f : \{1, \ldots, t\} \to \{1, \ldots, t\}$ with $colour(n_i) = colour(m_{f(i)})$ and $colour(n_j) = colour(m_{f(j)})$.
3. If n, m are of even height (i.e., of colour A or B), there is a colour preserving bijection from the grandchildren of n to the grandchildren of m.

An AB-tree T is presented to a PURPLE programs as a pointer structure over the set of nodes of T with the label $parent$ (which maps $root(T)$ to itself and other nodes to their parents), and the predicate A_0 (true only for leaf nodes with colour A). The colours of internal nodes are not available to PURPLE programs. PURPLE can solve isomorphism of AB-trees if and only if there is a program that can determine, for inputs of any height, the colour of the root. The following example shows that PURPLE can determine the colour of internal nodes up to fixed height. W.l.o.g., the initial configuration of any program places its pebbles on the root of its input, and initializes all boolean variables to `false`.

Example: The following PURPLE program accepts tree C_1 but rejects D_1 and E_1. It uses boolean variables $\mathit{seen1}$, $\mathit{seen2}$, $\mathit{seen3}$ and result and pointers rt and chld.

```
forall $chld$ do
    if $(chld.parent = rt) \wedge A_0(chld)$ then
        if $seen3$ then $result :=$ true;
        if $seen2$ then $seen3 :=$ true;
        if $\neg seen1$ then $seen1 :=$ true else $seen2 :=$ true
```

Using this repeatedly, for any height h, there is a PURPLE program Prg_h that determines the colour of nodes of height up to h. However, the number of variables in Prg_h increases with h. Our main result asserts that no program can do this for all heights.

Definition 3. *A PURPLE (PURPLE$_{c,nd}$) program Prg is* simple *if it does not have any Boolean variables and only contains compositions of the form $\mathit{Prg}_1; \mathit{Prg}_2$ where Prg_1 does not contain loops, and in every* `forall` *p* `do` *Prg_1, Prg_1 does not modify the variable p. Output is represented by the equality of dedicated pointers* res_1 *and* res_2.

Lemma 4. *For every PURPLE (PURPLE$_{c,nd}$) program Prg, there is a simple PURPLE (PURPLE$_{c,nd}$) program Prg′ such that an input with more than two nodes is accepted (resp. rejected) by Prg′ iff it is accepted (resp. rejected) by Prg.*

A pebbling I *fixes* a node n, if I places a pebble p on a node of the subtree rooted at n. Fixed nodes must be treated like pebbled ones, as PURPLE programs can navigate the path from pebbled nodes to the root.

Pebblings I, J with pebbles P *distance-match* ($I \sim J$) if the relative placement of pebbles is the same, i.e. $\forall p, q \in P.\ \mathit{ht}(I(p)) = \mathit{ht}(J(p))$ and $\mathit{dist}(I(p), I(q)) = \mathit{dist}(J(p), J(q))$, where, $\mathit{dist}(n, m) = \langle h - \mathit{ht}(n), h - \mathit{ht}(m)\rangle$ where h is the height of the closest common ancestor of n and m. If $I \sim J$ we define $\mathit{match}_{(I,J)}$ as a function from nodes fixed by I to nodes fixed by J such that $\forall p \in P.\ \mathit{match}_{(I,J)}(I(p)) = J(p)$ and $\forall n.\ (\mathit{match}_{(I,J)}(n) = m) \ \rightarrow\ (\mathit{match}_{(I,J)}(\mathit{parent}(n)) = \mathit{parent}(m))$. The pebblings I, J *colour-match till* h, written $I \sim_h J$, if $I \sim J$ and $\mathit{colour}(n) = \mathit{colour}(\mathit{match}_{(I,J)}(n))$ for any node n with $\mathit{ht}(n) \leq h$. Given pebblings I_1 and I_2 with pebbles P, we write $I_1 I_2$ for their disjoint combination, defined to be the pebbling with pebbles $\{1,2\} \times P$, such that $I_1 I_2(\langle i, p\rangle) = I_i(p)$ for any $\langle i, p\rangle \in \{1,2\} \times P$.

Lemma 5. *Given pebblings I_1, I_2, J_1, and J_2 with pebbles P, $I_1 I_2 \sim_h J_1 J_2$ iff $I_1 \sim_h J_1$ and $I_2 \sim_h J_2$ and $\forall p, q \in P.\ \mathit{dist}(I_1(p), I_2(q)) = \mathit{dist}(J_1(p), J_2(q))$.*

A structural isomorphism F *witnesses* $I \sim J$ if $J(r) = F(I(r))$ for every pebble r. We say that J is induced by F and I. Note that for any node n fixed by I, $\mathit{match}_{(I,J)}(n) = F(n)$. Given a node n of a tree T, pebblings I, J and a structural isomorphism F, we denote by $T|_n$ the subtree rooted at the node n; we denote by $I|_n$ and $F|_n$ the restrictions of I and F to $T|_n$ respectively. Likewise, given a height h, we write $I|^h$ and $F|^h$ for the restrictions of I and F to nodes above h respectively. Finally, given a height h, we write $F\|_h$ to mean that for any node n with height $\leq h$ and $colour(n) = colour(F(n))$, $F|_n$ is a colour isomorphism; analogously, we write $I\|_h J$ if for any nodes n, m with height $\leq h$ and $match_{(I,J)}(n) = m$, and $colour(m) = colour(n)$, we have $I|_n \sim_h J|_m$.

Lemma 6. *For all pebblings I, J with $I \sim_h J$, there is a structural isomorphism F with $F\|_h$ that witnesses it.*

4 Main Result

In this section we state and prove our main result – the impossibility of distinguishing between different tree colours at all heights. For simplicity and space reasons, we only give the proof for plain PURPLE as defined in Sec. 2; the extension with nondeterminism and counting is not given for space reasons; its structure and main invariants are identical to the proof presented in the main part.

Configurations of simple PURPLE program are just pebblings. We show that every simple PURPLE program that halts on input A_H in some pebbling O_A, will halt on input B_H (for some traversal) in some pebbling O_B such that $O_A(res_1) = O_A(res_2)$ iff $O_B(res_1) = O_B(res_2)$.

Theorem 1. *For any simple PURPLE program Prg there exists a height h with the following property: Whenever $h' \geq h$ and $H \geq h'$ (both h' and H even), then for any pebbling I_1, there exists a pebbling O_1 with Prg $\vdash_{A_H} I_1 \longrightarrow O_1$ such that whenever $I_1 \sim_{h'} I_2$ for some I_2, then we have*

- *INV1(h) : there is a pebbling O_2 with $I_1O_1 \sim I_2O_2$ and Prg $\vdash_{B_H} I_2 \longrightarrow O_2$;*
- *INV2(h) : for any pebbling O_2 with $I_1O_1 \sim_{h'} I_2O_2$ we have Prg $\vdash_{B_H} I_2 \longrightarrow O_2$.*

We take the properties *INV1*(h) and *INV2*(h) as defined by the statement of the theorem and denote their conjunction by $INV(h)$. The proof of the theorem is by induction on *Prg* and broken down into lemmas, one for each constructor, the most interesting being of course the forall loop. Before doing so, we note that this (together with Lemma 4) implies that PURPLE cannot solve tree isomorphism.

Corollary 1. *No PURPLE program can recognize the set $\{A_{2n} \mid n \in \mathbb{N}\}$.*

Proof. Assume, for a contradiction, that *Prg* recognizes $\{A_{2n} \mid n \in \mathbb{N}\}$. By Theorem 1 there exists an h such that *Prg* satisfies *INV*(h). Choose even $h' \geq h$ and $H > h'$ and choose I_1 and I_2 to be the configurations on A_H and B_H respectively that place all pebbles of *Prg* on the roots. Clearly, we have $I_1 \sim_{h'} I_2$. Theorem 1 gives us O_1 with *Prg* $\vdash_{A_H} I_1 \longrightarrow O_1$. Since *Prg* accepts A_H, configuration O_1 must be accepting. Property *INV1*(h) then furnishes O_2 with $I_1O_1 \sim I_2O_2$ and *Prg* $\vdash_{B_H} I_2 \longrightarrow O_2$. By $I_1O_1 \sim I_2O_2$, configuration O_2 must also be accepting. Hence, *Prg* does not reject B_H in the sense of Def. 1. Therefore it cannot recognize $\{A_{2n} \mid n \in \mathbb{N}\}$.

We now come to the inductive cases.

Lemma 7. *Let Prg be* `skip` *or an assignment. Then Prg satisfies $INV(0)$.*

Lemma 8. *Let Prg be* `if` c `then` Prg_1 `else` Prg_2 *where Prg_i satisfies $INV(h_i)$ for $i = 1, 2$. Then Prg satisfies $INV(\max(h_1, h_2))$.*

Proof. On pebblings I and J with $I \sim_h J$ ($h \geq 0$), c evaluates identically since atomic conditions do. As this involves no pebble movements, the result follows.

Lemma 9. *Let Prg be $Prg_1; Prg_2$ where Prg_1 contains no loops and Prg_2 satisfies $INV(h)$. Then Prg satisfies $INV(h)$.*

Proof. On pebblings I_1, I_2 with $I_1 \sim_{h'} I_2$ ($h' \geq h$), conditions evaluate identically (as in Lemma 8). Since Prg_1 can only move pebbles to already pebbled nodes, it halts with final pebblings J_1 and J_2 with $J_1 \sim_{h'} J_2$. The result follows since Prg_2 satisfies $INV(h)$.

Let *Prg* be `forall` r `do` Prg_1. Let $X = X_1, \ldots, X_{|A_H|}$ be the list of pebblings generated at the beginning of each iteration of *Prg*. Thus, $X_i(r) = E[i]$ for some enumeration E of A_H. Then, $\forall i < |A_H|.\ Prg_1 \vdash X_i \longrightarrow X_{i+1}[r \mapsto E[i]]$ and $Prg \vdash X_1 \longrightarrow X_{|A_H|+1}$ where $Prg_1 \vdash X_{|A_H|} \longrightarrow X_{|A_H|+1}$. Consider a list $Y = Y_1, \ldots, Y_{|B_H|+1}$ of pebblings of B_H such that $\forall i < |A_H|, X_i X_{i+1} \sim_h Y_i Y_{i+1}$. Then, assuming $INV2$ for Prg_1, $\forall i < |A_H| + 1.\ Prg_1 \vdash Y_i \longrightarrow Y_{i+1}$. Further, if $Y_i(r)$ gives an enumeration of B_H, then $Prg \vdash Y_1 \longrightarrow Y_{|B_H|+1}$. So, to prove the invariant for *Prg*, we need to construct such a list Y for some list X as above.

We first construct this matching list for a list X in which $X[i](r)$ is not necessarily an enumeration of the tree. The construction appears in Lemma 11 below. Its proof (not given for space reasons) relies on the following lemma, which shows that to colour match up to some height, it suffices to match what we call single children.

Definition 4. *A (non-root) node n is a* single child at pebbling I, *written in short as $singlechild(n, I)$, if I fixes n, but does not fix any of the siblings of n.*

Lemma 10. *For all pebblings I and J of trees of height H with pebbles P, if $(I \|_{H-|P|} J)$ and $\forall n.singlechild(n, I) \to colour(n) = colour(match_{(I,J)}(n))$, then $(I \sim_{H-|P|} J)$.*

Proof. Towards a contradiction, assume I fixes a node n of height $h \leq H - |P|$ and $colour(n) \neq colour(match_{I,J}(n))$. Since $(I \|_{H-|P|} J)$, for every ancestor m of n, $colour(m) \neq colour(match_{I,J}(m))$. So, I fixes some sibling of every ancestor of n (including n) i.e., I pebbles more than $|P|$ pairwise disjoint trees. □

The following lemma states that given a list of pebblings of A_H, it is possible to construct a list of pebblings over B_H such that successive pebbling pairs will colour match till desired height. The required list of pebblings is constructed recursively to colour match single children.

Lemma 11. *Let $X = X_0, \ldots, X_k$ be a list of pebblings of A_H with pebbles P, and Y_0 and Y_k be pebblings of B_H with $X_0 X_k \sim_h Y_0 Y_k$ witnessed by F. There exists a list of pebblings $Y_1, \ldots, Y_{k-1}$ such that for $0 \leq i < k$, $X_i X_{i+1} \sim_{h-|P|} Y_i Y_{i+1}$, and $Y_i|^h = F \circ X_i|^h$ and $Y_i|_n = F \circ X_i|_n$ for every node n at height h with $colour(n) = colour(F(n))$.*

To use these results for a program Prg defined as `forall` r `do` Prg_1, we need to extend the proof for a list of pebblings X where the list of nodes $X[i](r)$ is some enumeration of the A_H and construct a list Y where the list of nodes $Y[i](r)$ is some enumeration of the B_H. Since PURPLE programs behave the same for any enumeration, we choose for $X[i](r)$ an enumeration of A_H called "special run" (SR) such that there is an enumeration "match run" (MR) of B_H with $ht(SR[i]) = ht(MR[i])$ and $dist(SR[i], SR[i{+}1]) = dist(MR[i], MR[i{+}1])$ and $colour(SR[i]) = colour(MR[i])$ if $ht(SR[i]) \leq h$ (determined by Prg_1).

We define the enumeration of the tree A_H called the *special run* $SR_{H,h}$ parameterized by an even height h. We fix a function gc that maps the nodes of A_H to an enumeration of their grandchildren. Int is a list of sets of nodes of A_H such that

$$Int[0] = \{m \mid ht(m) \geq h-1\},$$
$$Int[k] = \{m \mid m \in A_H|_{gc(n)[k]} \text{ for some } n \text{ with } ht(n) = h\} \text{ for } 1 \leq k \leq 48.$$

Define *interval boundary* $IB_{H,h}(j) = \sum_{i=0}^{j-1} |Int[i]|$ for $0 \leq j \leq 49$. Then, for $0 \leq i \leq 48$ the sublist of $SR_{H,h}$ from $IB_{H,h}(i) + 1$ till $IB_{H,h}(i+1)$ is called the i^{th} interval of $SR_{H,h}$ and is defined to be an arbitrary enumeration of $Int[i]$.

Lemma 12. *For all even heights H and h, the j^{th} interval of $SR_{H,h}$ ($1 \leq j < 49$) contains, for every node n with $ht(n) = h$, the subtree rooted at exactly one grandchild of n; the 0^{th} interval contains nodes with height $\geq h-1$.*

Given a structural isomorphism $F : A_H \to B_H$, we define the enumeration *match run* $MR_{H,h,F}$ of B_H. For each n of A_H of even height, we fix some colour preserving bijection f_n from the grandchildren of n to the grandchildren of $F(n)$ and let GC_n be the union of some colour isomorphisms from $A_H|_{n'}$ to $B_H|_{f_n(n')}$ for each grandchild n' of n. (GC_n may not preserve distances.) With these we define the enumeration *match run* $MR_{H,h,F}$ of B_H by: if $ht(SR_{H,h}[i]) \geq h-1$, then $MR_{H,h,F}[i] = F(SR_{H,h}[i])$; otherwise let n be the ancestor of $SR_{H,h}[i]$ at height h. If $colour(n) = colour(F(n))$, then, $MR_{H,h,F}[i] = F(SR_{H,h}[i])$; else $MR_{H,h,F}[i] = GC_n(SR_{H,h}[i])$.

Lemma 13. *Let H and h be even heights, and $F : A_H \to B_H$ be a structural isomorphism with $F\|h$. Let IB, SR and MR be $IB_{H,h}$, $SR_{H,h}$ and $MR_{H,h,F}$ respectively.*

1. *for all $i < |A_H|$, we have $ht(SR[i]) = ht(MR[i])$ and $ht(SR[i]) \leq h-2 \to colour(SR[i]) = colour(MR[i])$;*
2. *if $0 \leq j \leq 49$ and $IB[j] \leq i < IB[j+1]$, then $dist(SR[i],\ SR[i+1]) = dist(MR[i], MR[i+1])$;*
3. *if n is the ancestor of $SR[IB[j]]$ at height h and $colour(n) = colour(F(n))$, then we have $dist(SR[IB[j]-1],\ SR[IB[j]]) = dist(MR[IB[j]-1], MR[IB[j]])$ for $0 < j \leq 49$.*

Since GC_n may not preserve distances, we needed the extra conditions at the interval boundaries to ensure MR is defined using F. Lemma 11 shows that given a list X of pebblings and an initial structural isomorphism F, we can construct a list Y whose corresponding elements colour-match with X, such that under some conditions, $Y[i](r) = F \circ X[i](r)$. When we add the constraint that $X[i] = SR(i)$, if we have $F(SR_{H,h}(i)) = MR_{H,h,G}(i)$ for some G, and the conditions required to ensure $Y[i](r) = F \circ X[i](r)$, the result follows. However, no structural (let alone colour) isomorphism satisfies $Y[i](r) = F \circ X[i](r)$. So, we divide X into 49 sublists, one for each interval of $SR_{H,h}$, such that there are 49 structural isomorphisms with the property that successive ones agree on the nodes fixed at the boundaries of the intervals, and $MR_{H,h,F_0}[i] = F_j(SR[i])$ for if i is the in j^{th} interval of SR.

Lemma 14. *Let H, h be even heights, P be a set of pebbles, $h' = h - 50|P|$, $IB = IB_{H,h'}$, $SR = SR_{H,h'}$, $|A_H| = N$. Let $X = X_1, \ldots, X_N$ be a list of pebblings with $X_i(r) = SR[i]$. Let Y_1, Y_N be pebblings of B_H with $X_1 X_{A_H} \sim_h Y_1 Y_N$. Then, there are structural isomorphisms F_j ($0 \leq j \leq 48$) with*

1. *$MR_{H,h',F_0}[i] = F_j(SR[i])$ for $IB[j] \leq i \leq IB[j+1]$;*
2. *$F_0(n) = F_j(n)$ on all nodes n fixed by $X_{IB[j]}$ and $X_{IB[j+1]}$;*
3. *for all i, either $ht(SR[i]) > h' - 2$ or the ancestor n of $SR[i]$ at height $h' - 2$ satisfies $colour(n) = colour(F_j(n))$ where $IB[j] \leq i \leq IB[j+1]$.*

Proof. We first want to define an F_0 such that $F_0\|_{h'}$ and $colour(n) = colour(F_0(n))$ for any node n at height h' fixed by $X_{IB[j]}$ for some j ($0 \leq j \leq 48$). Define F_0 as follows: Let $S = \{IB[i] \mid 0 \leq i \leq 49\}$, and R_1, R', R_N be pebbling with pebbles $S \times P$, defined as $R_1(\langle i, p\rangle) = X_1(p)$, $R_N(\langle i, p\rangle) = X_N(p)$and $R'(\langle i, p\rangle) = X_i(p)$ for each $i \in S, p \in P$. Using Lemma 11 applied to the list $R_1 : R' : R_2$, Y_1 and Y_N with any structural isomorphism witnessing $X_1 X_{A_H} \sim_h Y_1 Y_{B_H}$ we can construct a Y' with $R_0 R' \sim_{h'} Y_0 Y'$. Then, there is a structural isomorphism F_0 witnessing this with the required properties.

Define F_j as follows: Take $F_j|^{h'} = F_0|^{h'}$, and $F_j|_n = F_0|_n$ for any node n of height h', with $colour(n) = colour(F_0(n))$ (this satisfies (1) for nodes below such n; Since F_0 is such that all nodes fixed by $X_{IB[j]}$ for $0 \leq j \leq 48$, (2) is satisfied). To satisfy (1) below any node n with height h' and $colour(n) \neq colour(F_0(n))$, define $F_j|_n$ as follows: By Lemma 12, there is at most one grandchild of n in j^{th} interval of $IB[j], \ldots, IB[j+1]$ of SR - say n'. So the only nodes that (1) enforces are in $A_H|_{n'}$. Set $F_j(n'') = MR[l'']$ for each $n'' \in A_H|_{n'}$ (including n')) where $SR[l''] = n''$. To preserve distances, set $F_j(parent(n')) = parent(F_j(n'))$. Elsewhere in $A_H|_n$, F_j is any structural isomorphism. Thus, $MR[i] = F_j(X_i(r))$ for $IB[j] < i \leq IB[j+1]$. (3) follows from the construction. □

The following lemma shows that given a list of pebblings of A_H, with pebble i enumerating it, these structural isomorphisms can be used to construct a list of pebblings B_H such that the successive pairs of pebblings colour match to the desired heights, and pebble i enumerates B_H.

Lemma 15. *Let H, h be even heights, P be a set of pebbles, $h' = h - 51|P|$, $IB = IB_{H,h'}$, $SR = SR_{H,h'}$, $|A_H| = N$. Let $X = X_1, \ldots, X_N$ be a list of pebblings with $X_i(r) = SR[i]$. Let Y_1, Y_N be pebblings of B_H such that $X_1 X_N \sim_h Y_1 Y_N$. Then there is a list of pebblings $Y = Y_1, \ldots, Y_N$ such that $Y_i \sim_{h'-|P|-2} X_i$ and a structural isomorphism F_0 such that $Y_i(r) = MR_{H,h',F_0}[i]$.*

Proof. By Lemma 14, for all $0 \leq j \leq 48$ there exists F_j with properties 1, 2 and 3 as in Lemma 14. Define $Y_{IB[j]} = F_0 \circ X_{IB[j]}$ for $0 \leq j \leq 49$. By Lemma 14(2), F_j agrees with F_0 on nodes fixed by pebblings $X_{IB[j]}$ and $X_{IB[j+1]}$, i.e., F_j witnesses $X_{IB[j]}X_{IB[j+1]} \sim_{h'} Y_{IB[j]}Y_{IB[j+1]}$. In order to make use of Lemma 14(3), we weaken this to $X_{IB[j]}X_{IB[j+1]} \sim_{h'-2} Y_{IB[j]}Y_{IB[j+1]}$. By Lemma 11, applied to $X_{IB[j]}, \ldots, X_{IB[j+1]}$, $Y_{IB[j]}$ and $Y_{IB[j+1]}$ with F_j witnessing $X_{IB[j]}X_{IB[j+1]} \sim_{h'-2} Y_{IB[j]}Y_{IB[j+1]}$, construct $Y_{IB[j]+1}, \ldots, Y_{IB[j+1]-1}$ such that for (a) $IB[j] \leq i \leq IB[j+1]$, $X_iX_{i+1} \sim_{h'-2-|P|} Y_iY_{i+1}$ and (b) $Y_i|^{h'} = F_j \circ X_i|^{h'}$ and (c) $Y_i|_n = F_j \circ X_i|_n$ for every node n at height h with $colour(n) = colour(F_j(n))$. By (b), (c) and 14 (3), $Y_i(r) = F_j \circ X_i(r)$ since $X_i(r) = SR[i]$. By Lemma 14 (1) we have $Y_i(r) = MR_{H,h',F_0}[i]$. □

Using this, the INV for the forall loop can be proved directly.

Lemma 16. *If Prg* $=$ `forall` r `do` *Prg_1 and Prg_1 satisfies $INV(h_1)$ then Prg satisfies $INV(h)$ where $h = h_1 + 52|P| + 2$ and P is set of pointers in Prg.*

Proof. Consider an even height $h' > h$. Let $gph = h' - 51|P|$, $gch = gph - 2$, $h_1' = gch - |P|$, and $H \geq h'$ be even, $IB = IB_{H,gph}$, $SR = SR_{H,gph}$, and $|A_H| = N$.

INV1: Let pebblings I_1 of A_H and I_2 of B_H satisfy $I_1 \sim_{h'} I_2$. Let $X = X_1, \ldots, X_N$ be a list of pebblings with $X_i(r) = SR[i]$, $X_1 = I_1[r \mapsto SR[1]]$, and for all $1 \leq i < N$, $Prg_1 \vdash X_i \longrightarrow X_{i+1}[r \mapsto SR[i]]$ such that for any pebblings J_1 and J_2 with $X_iX_{i+1}[r \mapsto SR[i]] \sim_{h_1'} J_1J_2$ we have $Prg_1 \vdash J_1 \longrightarrow J_2$. (Since Prg_1 does not modify r, by IH ($INV2$), such an X exists). By Lemma 15 with $Y_1 = I_2$ and $Y_N = F \circ (X_N)$ for some F that witnesses $I_1 \sim_{h'} I_2$, we have a list Y with $Prg_1 \vdash Y_i \longrightarrow Y_{i+1}[r \mapsto MR[i]]$ for all $i \leq N$. Applying IH again to get $Prg_1 \vdash X_N \longrightarrow O_1$ and $Prg_1 \vdash Y_N \longrightarrow O_2$, the result follows.

INV2: Let pebblings I_1, O_1 of A_H and I_2, O_2 of B_H satisfy $I_1O_1 \sim_{h'} I_2O_2$. Let X be as above. By Lemma 11 applied to $I_1 : X_N : O_1$ and I_2 and O_2, construct Y_N.

By Lemma 15 construct list Y with $Prg_1 \vdash Y_i \longrightarrow Y_{i+1}[r \mapsto MR[i]]$ for all $i \leq N$. So the result follows. □

The proof of Theorem 1 is now a straightforward induction on program structure.

The key to generalizing our main result for nondeterminism and counting lies in the correct formulation of the invariants, in particular the quantification for runs and nondeterministic choices:

Theorem 2. *For any simple* PURPLE$_{c,nd}$ *program Prg there exists a height h with the following property: Whenever $h' \geq h$ and $H \geq h'$ (both h' and H even), then for any pebblings $\rho_{i1} \sim_{h'} \rho_{i2}$ and any state q, there exists σ_1 (pre-selected traversal sequences) such that $Prg \vdash_{A_H} \langle \rho_{i1}, q, \sigma_1 \rangle \longrightarrow \langle \rho_{o1}, q', \sigma_1' \rangle$ implies*

- *$INV1(h)$: there is a pebbling ρ_{o2} with $\rho_{i1}\rho_{o1} \sim \rho_{i2}\rho_{o2}$ and σ_2 and σ_2' such that $Prg \vdash_{B_H} \langle \rho_{i2}, q, \sigma_2 \rangle \longrightarrow \langle \rho_{o2}, q', \sigma_2' \rangle$;*
- *$INV2(h)$: for any pebbling ρ_{o2} with $\rho_{i1}\rho_{o1} \sim_{h'} \rho_{i2}\rho_{o2}$ there exist σ_2 and σ_2' with $Prg \vdash_{B_H} \langle \rho_{i2}, q, \sigma_2 \rangle \longrightarrow \langle \rho_{o2}, q', \sigma_2' \rangle$.*

The proof follows the proof for PURPLE almost verbatim.

Corollary 2. *No* PURPLE$_{c,nd}$ *program can recognize the set* $\{A_{2n} \mid n \in \mathbb{N}\}$.

Proof. Assume, for a contradiction, that *Prg* recognizes $\{A_{2n} \mid n \in \mathbb{N}\}$. Choose h such that *Prg* satisfies *INV*(h). Let $H > h$ be even and choose ρ_{i1} and ρ_{i2} to be the pebblings on A_H and B_H respectively that place all pebbles of *Prg* on the roots. Let q be the state that maps all counters to 0. Clearly, we have $\rho_{i1} \sim_{h'} \rho_{i2}$. Let σ_1 be the pre-selected traversal sequences, as provided by Theorem 2. Since *Prg* accepts A_H, we must have *Prg* $\vdash_{A_H} \langle \rho_{i1}, q, \sigma_1 \rangle \longrightarrow \langle \rho_{o1}, q', \sigma_1' \rangle$ for some accepting configuration $\langle \rho_{o1}, q', \sigma_1' \rangle$. *INV1*$(h)$: then furnishes ρ_{o2} and σ_2 and σ_2' with $\rho_{i1}\rho_{o1} \sim \rho_{i2}\rho_{o2}$ and *Prg* $\vdash_{B_H}$ $\langle \rho_{i2}, q, \sigma_2 \rangle \longrightarrow \langle \rho_{o2}, q', \sigma_2' \rangle$. As $\langle \rho_{o2}, q', \sigma_2' \rangle$ must be accepting by $\rho_{i1}\rho_{o1} \sim \rho_{i2}\rho_{o2}$, program *Prg* does not reject B_H. Therefore *Prg* cannot recognize $\{A_{2n} \mid n \in \mathbb{N}\}$.

5 Horn satisfiability

For any AB-tree T, PURPLE can construct a Horn satisfiability problem S_T that is unsatisfiable iff T is A_H or C_H: The problem S_T has one variable X_n for each node n in T. The goal clause is $\neg X_{root(T)}$. For each node n with $A_0(n)$ there is a clause X_n making X_n a fact. Finally, for each node n with 8 (resp. 6) children and any choice of $t = 3$ (resp. $t = 4$) pairwise distinct children $n_1, \ldots, n_t$ of n, there is a clause $X_n, \neg X_{n_1}, \ldots, \neg X_{n_t}$. As a result, S_T is satisfiable iff T is an B, D or an E tree. So, PURPLE cannot solve Horn satisfiability problems presented as sets of clauses.

6 Conclusion and future work

We have introduced an extension of PURPLE (imperative language with statically allocated pointers and iteration) with non-determinism and counting. Our main result shows that even in this extension isomorphism of unordered trees cannot be decided even though this problem is in LOGSPACE. This generalises an analogous result for transitive closure logic with counting [5]. A completely new proof method based on bisimulation was necessary to prove this result. This then furnishes a new proof of the result in [5], but more importantly sheds new light on the strength and weakness of extensions to PURPLE.

This whole line of work is motivated by the quest for a meaningful rigorization of the intuition that a constant number of pointers cannot be sufficient to check satisfiability of a set of Horn clauses. It seems so obvious that a non-constant number of "already proven" facts must be kept in memory no matter which algorithm is used. On the other hand, since Horn satisfiability is complete for PTIME and only a "constant number of pointers" can be done in LOGSPACE a comprehensive proof of such a statement would separate LOGSPACE from PTIME. Thus, until the techniques become strong enough to prove such a result we seek meaningful relativisations of such a separation. To this end we introduced the programming language PURPLE that provides the ability to iterate over all input elements in an unspecified order. We now seek extensions of this formalism that while remaining within LOGSPACE enhance the power so as to get more expressive relativised separations.

The work reported here narrows down the design space for such an extension considerably. We have proved that PURPLE extended with nondeterminism and counting cannot decide isomorphism of trees where the children of a node are unordered.

For the above research programme this has the important implication that an encoding of Horn satisfiability where clauses and variables are unordered, i.e., we merely have a relation that tells whether a variable appears as a premise or conclusion of a Horn clause, is not appropriate for a meaningful relativised separation because tree isomorphism can be—within PURPLE—reduced to this version of Horn satisfiability. So, Horn satisfiability cannot be decided in PURPLE augmented with nondeterminism and counting either, but for the very reason that a particular LOGSPACE problem, namely tree isomorphism cannot.

This narrowing of the design space leaves the following two options to be investigated: Further extend PURPLE, for instance with the LREC-recursor by Grohe et al. [7] and try to prove that unordered Horn satisfiability cannot be decided in that system. While this would be a very interesting result, we feel that this path does not really get to the essence of the issue because it is too dependent on the precise presentation of the input structure. Our preferred solution is to remove counting, but merely provide nondeterminism and possible restricted patterns of recursion. In view of Prop. 1 this would then allow us to present instances of Horn satisfiability with as much local ordering as desired. E.g. by ordering the premises of any clause and giving for each variable a list of clauses which conclude with that variable. Unordered tree isomorphism can then no longer be encoded as Horn satisfiability.

Thus, in summary, the results of this paper give us robust evidence that in order to stay clear of full LOGSPACE while at the same time not being trivially weak one should stick to a constant number of variables be they pointer or boolean variables.

References

1. Guillaume Bonfante. Some programming languages for LOGSPACE and PTIME. AMAST'06, pages 66–80, Berlin, Heidelberg, 2006. Springer-Verlag.
2. Jin-Yi Cai and D. Sivakumar. Sparse hard sets for P: Resolution of a conjecture of Hartmanis. *J. Comput. Syst. Sci.*, 58(2):280–296, 1999.
3. Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM J. Comput.*, 9(3):636–652, 1980.
4. Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer, 1995.
5. Kousha Etessami and Neil Immerman. Tree canonization and transitive closure. In *IEEE Symp. Logic In Comput. Sci*, pages 331–341, 1995.
6. Erich Grädel and Gregory L. McColm. On the power of deterministic transitive closures. *Inf. Comput.*, 119(1):129–135, 1995.
7. Martin Grohe, Berit Grußien, André Hernich, and Bastian Laubner. L-recursion and a new logic for logarithmic space. In *CSL*, pages 277–291, 2011.
8. Martin Hofmann and Ulrich Schöpp. Pointer programs and undirected reachability. In *LICS*, pages 133–142, 2009.
9. Martin Hofmann and Ulrich Schöpp. Pure pointer programs with iteration. *ACM Trans. Comput. Log.*, 11(4), 2010.
10. Neil Immerman. Progress in descriptive complexity. In *Curr. Trends in Th. Comp. Sci.*, pages 71–82. 2001.

11. Neil D. Jones. LOGSPACE and PTIME characterized by programming languages. *Theor. Comput. Sci.*, 228(1-2):151–174, October 1999.
12. Richard E. Ladner and Nancy A. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.
13. Steven Lindell. A logspace algorithm for tree canonization (extended abstract). STOC '92, pages 400–404, New York, NY, USA, 1992. ACM.
14. Pinyan Lu, Jialin Zhang, Chung Poon, and Jin-Yi Cai. Simulating undirected *st*-connectivity algorithms on uniform JAGs and NNJAGs. In *Algo. and Comp.*, volume 3827 of *LNCS*. 2005.
15. Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.