What Can Be Verified Locally?

Alkida Balliu, Gianlorenzo D 'Angelo, Pierre Fraigniaud, Dennis Olivetti

What Can Be Verified Locally?*

Alkida Balliu[†1], Gianlorenzo D'Angelo[2], Pierre Fraigniaud[‡3], and Dennis Olivetti[§1]

[1]Department of Computer Science, Aalto University, Finland.
[2]Gran Sasso Science Institute, L'Aquila, Italy.
[3]CNRS and University Paris Diderot, France.

Abstract

In the framework of *distributed network computing*, it is known that not all Turing-decidable predicates on labeled networks can be decided *locally* whenever the computing entities are Turing machines (TM), and this holds even if nodes are running *non-deterministic* Turing machines (NTM). In contrast, we show that every Turing-decidable predicate on labeled networks can be decided locally if nodes are running *alternating* Turing machines (ATM). More specifically, we show that, for every such predicate, there is a local algorithm for ATMs, with at most two alternations, that decides whether the actual labeled network satisfies that predicate. To this aim, we define a hierarchy of classes of decision tasks, where the lowest level contains tasks solvable with TMs, the first level those solvable with NTMs, and the level $k > 1$ contains those tasks solvable with ATMs with $k - 1$ alternations. We characterize the entire hierarchy, and show that it collapses in the second level. In addition, we show separation results between the classes of network predicates that are locally decidable with TMs, NTMs, and ATMs, and we establish the existence of completeness results for each of these classes, using novel notions of *local reduction*. We complete these results by a study of the local decision hierarchy when certificates are bounded to be of logarithmic size.

*A preliminary version of this paper has appeared in the proceedings of the 34th International Symposium on Theoretical Aspects of Computer Science (STACS), Hannover, Germany, March 8-11, 2017. This work was partially done during the first and fourth authors visit at IRIF (CNRS and University Paris Diderot).

†Supported in part by Gran Sasso Science Institute, L'Aquila, Italy, and by the Academy of Finland, Grant 285721. Additional support from the French ANR project DESCARTES.

‡Additional support from the French ANR project DESCARTES, and the Inria project GANG.

§Supported in part by Gran Sasso Science Institute, L'Aquila, Italy, and by the Academy of Finland, Grant 285721. Additional support from the French ANR project DESCARTES.

1 Introduction

1.1 Context and objective

In the framework of network computing, *distributed decision* is the ability to check the legality of network configurations using a distributed algorithm. This concern is of the utmost importance in the context of fault-tolerant distributed computing, where it is highly desirable that the nodes are able to collectively check the legality of their current configuration, which could have been altered by the corruption of variables due to failures. In this paper, we are interested in *local* distributed decision. More specifically, we consider the standard LOCAL model of computation in networks [15]. Nodes are assumed to be given distinct identities, and each node executes the same algorithm, which proceeds in synchronous rounds where all nodes start at the same time. In each round, every node sends messages to its neighbors, receives messages from its neighbors, and performs some individual computation. The model does not limit the amount of data sent in the messages, neither does it limit the amount of computation that is performed by a node during a round. Indeed, the model places an emphasis on the number of rounds before every node can output, as a measure of locality. A *local algorithm* is a distributed algorithm $\mathcal{A}$ satisfying that there exists a constant $t \geq 0$ such that $\mathcal{A}$ terminates in at most t rounds in all networks, for all inputs. The parameter t is called the *radius* of $\mathcal{A}$. In other words, in every network G, and for all inputs to the nodes of G, every node executing $\mathcal{A}$ just needs to collect all information present in the t-ball around it in order to output, where the *t-ball* of u is the ball $B_G(u,t) = \{v \in V(G) : \mathrm{dist}(u,v) \leq t\}$, where $\mathrm{dist}(u,v)$ denotes the length (i.e., number of edges) of a shortest path between u and v.

The objective of the paper is to determine what network properties can be decided locally, as a function of the individual computing power of the nodes.

Following the guidelines of [7], we define a *configuration* as a pair (G,x) where $G = (V,E)$ is a connected simple undirected graph, and $x : V(G) \to \{0,1\}^*$ is a function assigning an input $x(u)$ to every node $u \in V$. A *distributed language* $\mathcal{L}$ is a set of configurations (we consider only Turing-decidable sets). A configuration $(G,x) \in \mathcal{L}$ is said to be *legal* w.r.t. $\mathcal{L}$. The membership of a configuration in a distributed language is independent of the identity that may be assigned to the nodes in the LOCAL model. For instance, the set $\{(G,x) : \exists v \in V(G), \mathrm{id}(v) = 1\}$ is not considered as a distributed language.

The class LD is the set of all distributed languages that are locally decidable. That is, LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying that, for every configuration (G,x),

$$(G,x) \in \mathcal{L} \iff \mathcal{A} \text{ accepts } (G,x)$$

where one says that $\mathcal{A}$ accepts if it accepts at *all* nodes. More formally, given a graph G, let $\mathrm{ID}(G)$ be the set of all injective functions from $V(G)$ to positive integers, i.e., $\mathrm{ID}(G)$ denote the set of all possible identity assignments to the nodes of G. Then LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G,x),

$$\begin{array}{rcl} (G,x) \in \mathcal{L} & \Rightarrow & \forall \mathrm{id} \in \mathrm{ID}(G), \forall u \in V(G), \mathcal{A}_{G,x,\mathrm{id}}(u) = \text{accept} \\ (G,x) \notin \mathcal{L} & \Rightarrow & \forall \mathrm{id} \in \mathrm{ID}(G), \exists u \in V(G), \mathcal{A}_{G,x,\mathrm{id}}(u) = \text{reject} \end{array}$$

where $\mathcal{A}_{G,x,\mathrm{id}}(u)$ is the output of Algorithm $\mathcal{A}$ running on the instance (G,x) with identity-assignment id, at node u. (Note that the two implications in the definition of LD cannot be

merged into one if-and-only-if statement because LD requires that both ways should hold *for any identity-assignment* to the nodes). For instance, the language PROP-COL, composed of all (connected) properly colored graphs, is in LD. Similarly, the class LCL of "locally checkable labelings", defined in [14], satisfies LCL $\subseteq$ LD. In fact, LCL is precisely LD restricted to configurations on graphs with constant maximum degree, and inputs of constant size.

The class NLD is the non-deterministic version of LD, i.e., the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ *verifying* $\mathcal{L}$, i.e., satisfying that, for every configuration (G, x),

$$(G, x) \in \mathcal{L} \iff \exists c, \mathcal{A} \text{ accepts } (G, x) \text{ with certificate } c.$$

More formally, NLD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G, x),

$$\begin{array}{rcl} (G, x) \in \mathcal{L} & \Rightarrow & \exists c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \forall u \in V(G), \mathcal{A}_{G,x,c,\mathrm{id}}(u) = \text{accepts} \\ (G, x) \notin \mathcal{L} & \Rightarrow & \forall c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \exists u \in V(G), \mathcal{A}_{G,x,c,\mathrm{id}}(u) = \text{rejects} \end{array}$$

where $\mathcal{C}(G)$ is the class of all functions $c : V(G) \to \{0, 1\}^*$, assigning the certificate $c(u)$ to each node u. Note that the certificates c may depend on the network and on the input to the nodes, but should be set independently of the actual identity assignment to the nodes of the network. If we were able to set certificates depending on the ID-assignment to the nodes, then every distributed language would be non-deterministically decidable [8, 9]. In this paper, we aim at a better understanding of the power given to the verification protocol by the ability to set up ID-dependent certificates. For this purpose, we follow the guidelines of [7] by considering ID-independent certificates, hence reducing the role if IDs to mere mechanisms enabling each node to solely distinguishing nodes in the network. (See [2] for a more detailed description of the differences between ID-dependent and ID-independent certificates). In the following, for the sake of simplifying the notations, we shall omit specifying the domain sets $\mathcal{C}(G)$ and $\mathrm{ID}(G)$ unless they are not clear from the context. It follows from the above that NLD is a class of distributed languages that can be locally *verified*, in the sense that, on legal instances, certificates can be assigned to nodes by a *prover* so that a *verifier* $\mathcal{A}$ accepts, and, on illegal instances, the verifier $\mathcal{A}$ rejects (i.e., at least one node rejects) systematically, and cannot be fooled by any fake certificate. For instance, the language

$$\text{TREE} = \{(G, x) : G \text{ is a tree}\}$$

is in NLD, by selecting a root r of the given tree, and assigning to each node u a counter $c(u)$ equal to its hop-distance to r (the hop-distance between two nodes u and v is the minimum number of edges of a path with extremities u and v). If the given (connected) graph contains a cycle, then no counters could be assigned to fool an algorithm checking that, at each node u with $c(u) \neq 0$, a unique neighbor v satisfies $c(v) < c(u)$, and all other neighbors $w \neq v$ satisfy $c(w) > c(u)$. In [6], NLD was proved to be exactly the class of distributed languages that are closed under lift, i.e., if (G, x) is in the language, any covering graph of (G, x) preserving the inputs is also in the language (see, e.g., Definition 1 in [6]).

Finally, [7] defined the randomized versions $\mathsf{BPLD}_{p,q}$ and $\mathsf{BPNLD}_{p,q}$, of the aforementioned classes LD and NLD, respectively, by replacing the use of a deterministic algorithm with the use of a randomized algorithm characterized by its probability $p > 0$ of acceptance for legal instances, and its probability $q > 0$ of rejection for illegal instances. By defining $\mathsf{BPNLD} = \cup_{p^2+q \geq 1} \mathsf{BPNLD}_{p,q}$, the landscape of local decision was pictured as follows:

$$\mathsf{LD} \subset \mathsf{NLD} \subset \mathsf{BPNLD} = \mathsf{All}$$

where all inclusions are strict, and All is the set of all distributed languages. That is, every distributed language can be locally verified with constant success probabilities p and q, for some p and q satisfying $p^2 + q \geq 1$. In other words, by combining non-determinism with randomization, one can decide any given distributed language.

1.2 Our contributions

Following up the approach recently applied to *distributed graph automata* in [16], and to the CONGEST model in [3], we observe that the class LD and NLD are in fact the basic levels of a "local hierarchy" defined as follows. Let $\Sigma_0^{\mathsf{local}} = \Pi_0^{\mathsf{local}} = \mathsf{LD}$, and, for $k \geq 1$, let $\Sigma_k^{\mathsf{local}}$ be the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying that, for every configuration (G, x),

$$(G,x) \in \mathcal{L} \iff \exists c_1, \forall c_2, \ldots, Q c_k, \mathcal{A} \text{ accepts } (G,x) \text{ with certificates } c_1, c_2, \ldots, c_k$$

where the quantifiers alternate, and Q is the universal quantifier if k is even, and the existential one if k is odd. The class Π_k^{local} is defined similarly, by starting with a universal quantifier, instead of an existential one. A local algorithm $\mathcal{A}$ insuring membership to a class $\mathcal{C} \in \{\Sigma_k^{\mathsf{local}}, k \geq 0\} \cup \{\Pi_k^{\mathsf{local}}, k \geq 0\}$ is called a $\mathcal{C}$-algorithm. Hence, $\mathsf{NLD} = \Sigma_1^{\mathsf{local}}$, and, for instance, Π_2^{local} is the class of all distributed languages $\mathcal{L}$ for which there exists a Π_2^{local}-algorithm, that is, a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G, x),

$$\begin{array}{rcl} (G,x) \in \mathcal{L} & \Rightarrow & \forall c_1, \exists c_2, \forall \mathrm{id}, \forall u \in V(G), \mathcal{A}_{G,x,c_1,c_2,\mathrm{id}}(u) = \text{accept}; \\ (G,x) \notin \mathcal{L} & \Rightarrow & \exists c_1, \forall c_2, \forall \mathrm{id}, \exists u \in V(G), \mathcal{A}_{G,x,c_1,c_2,\mathrm{id}}(u) = \text{reject}. \end{array} \quad (1)$$

Our main results are the following.

Theorem 1 $\mathsf{LD} \subset \Pi_1^{\mathsf{local}} \subset \mathsf{NLD} = \Sigma_2^{\mathsf{local}} \subset \Pi_2^{\mathsf{local}} = \mathsf{All}$*, where all inclusions are strict.*

That is, $\Pi_1^{\mathsf{local}} \supset \Pi_0^{\mathsf{local}}$, while $\Sigma_2^{\mathsf{local}} = \Sigma_1^{\mathsf{local}}$, and the whole local hierarchy collapses to the second level, at Π_2^{local}. In other words, while not every Turing-decidable network property can be decided locally if nodes are running *non-deterministic* Turing machines (NTM), Theorem 1 says that every Turing-decidable network property can be decided locally if nodes are running *alternating* Turing machines (ATM). More specifically, for every network property, there is a local algorithm for ATMs, with at most 2 alternations, that decides that property.

We complete our description of the local hierarchy by a collection of separation and completeness results regarding the different classes and co-classes in the hierarchy. In particular, we revisit the completeness results in [7], and show that the notion of reduction introduced in this latter paper is too strong, and may allow a language outside NLD to be reduced to a language in NLD. We introduce a more restricted form of local reduction, called *label-preserving*, which does not have this undesirable property, and we establish the following.

Theorem 2 NLD *and* Π_2^{local} *have complete distributed languages under local label-preserving reductions.*

Figure 1 summarizes all our separation results.

We complete these results by a study of the local decision hierarchy when certificates are bounded to be of logarithmic size. In this case, the hierarchy $(\Sigma_k^{\mathsf{log\text{-}local}}, \Pi_k^{\mathsf{log\text{-}local}})_{k \geq 0}$ may not collapse, and there are languages outside this hierarchy. We prove that, beyond the bottom levels, the ability to use the node identities to set the certificates does not help.

Theorem 3 *For every $k \geq 3$, the class $\Sigma_k^{\mathsf{log\text{-}local}}$ does not depend on whether or not the certificates depend on the node identities. The same holds for $\Pi_k^{\mathsf{log\text{-}local}}$, for every $k \geq 2$.*

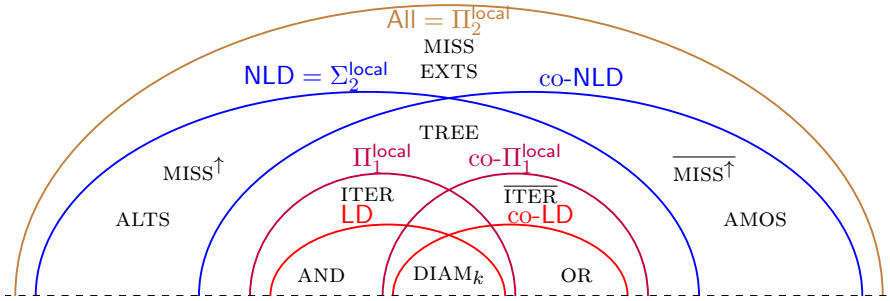

Figure 1: *Relations between the different decision classes of the local hierarchy (the definitions of the various languages can be found in the text).*

1.3 Related Work

Several forms of "local hierarchies" have been investigated in the literature, with the objective of understanding the power of local computation, and/or for the purpose of designing verification mechanisms for fault-tolerant computing. In particular, [16] has investigated the case of *distributed graph automata*, where the nodes are finite automata, and the network is anonymous (which are weaker assumptions than those in our setting), but also assuming an arbitrary global interpretation of the individual decisions of the nodes (which is a stronger assumption than those in our setting). It is shown that all levels Σ_k^{aut}, $k \geq 0$, of the resulting hierarchy are separated, and that the whole local hierarchy is exactly composed of the MSO (monadic second order) formulas on graphs.

In the framework of distributed computing, where the computing entities are Turing machines, *proof-labeling schemes* (PLS) [9], extended to *locally checkable proofs* (LCP) [8], give the ability to certify predicates using certificates that benefit of the node identities. That is, for the same network predicate, and the same legal network configuration, the distributed proof that this configuration is legal may be different if the node identities are different. In this context, the whole hierarchy collapses at the first level, with $\Sigma_1^{\mathsf{lcp}} = \mathsf{All}$. However, this holds only if the certificates can be as large as $\Omega(n^2)$ bits. In [3], the class LogLCP [8], which bounds the certificate to be of size $O(\log n)$ bits is extended to a hierarchy that fits to the CONGEST model. In particular, it is shown that MST stands at the second level $\Pi_2^{\mathsf{log\text{-}lcp}}$ of that hierarchy, while there are languages outside the hierarchy.

In [7], the authors introduced the model investigated in this paper. In particular, they defined and characterized the class NLD, which is nothing else than $\Sigma_1^{\mathsf{local}}$, that is, the class of languages that have a proof-labeling scheme in which the certificates are *not* depending on the node identities. It is proved that, while $\mathsf{NLD} \neq \mathsf{All}$, randomization helps a lot, as the randomized version BPNLD of NLD satisfies $\mathsf{BPNLD} = \mathsf{All}$. It is also proved that, with the oracle #nodes providing each node with the number of nodes in the network, we get $\mathsf{NLD}^{\#\mathrm{nodes}} = \mathsf{All}$. Interestingly, it was proved [6] that restricting the verification algorithms for NLD to be *identity-oblivious*, that is, enforcing that each node decides the same output for every identity-assignment to the nodes in the network, does not

reduce the ability to verify languages. This is summarized by the equality $\mathsf{NLDO} = \mathsf{NLD}$ where the "O" in NLDO stands for identity-oblivious. In contrast, it was recently proved that restricting the algorithms to be identity-oblivious reduces the ability to decide languages locally, i.e., $\mathsf{LDO} \subsetneq \mathsf{LD}$ (see [5]).

Finally, it is worth mentioning that the ability to decide a distributed language locally has impact on the ability to design *construction* algorithms [13] for that language (i.e., computing outputs x such that the configuration (G, x) is legal w.r.t. the specification of the task). For instance, it is known that if $\mathcal{L}$ is locally decidable, then any randomized local construction algorithm for $\mathcal{L}$ can be derandomized [14]. This result has been recently extended [1] to the case of languages that are locally decidable by a randomized algorithm (i.e., extended from LD to BPLD according to the notations in [7]). More generally, the reader is invited to consult [4, 10, 11, 12, 15, 17] for good introductions to local computing, and/or samples of significant results related to local computing.

2 All languages are Π_2^{local} decidable

In this section, we show the last equality of Theorem 1.

Proposition 4 $\Pi_2^{\mathsf{local}} = \mathsf{All}$.

Proof. Let $\mathcal{L}$ be a distributed language. We give an explicit Π_2^{local}-algorithm for $\mathcal{L}$, i.e., a local algorithm $\mathcal{A}$ such that, for every configuration (G, x), Eq. (1) is satisfied. For this purpose, we describe the distributed certificates c_1 and c_2. Intuitively, the certificate c_1 aims at convincing each node that $(G, x) \notin \mathcal{L}$, while c_2 aims at demonstrating the opposite. More precisely, at each node u in a configuration (G, x), the certificate $c_1(u)$ is interpreted as a triple $(M(u), \mathrm{data}(u), \mathrm{index}(u))$ where $M(u)$ is an $m \times m$ boolean matrix, $\mathrm{data}(u)$ is a linear array with m entries, and $\mathrm{index}(u) \in \{1, \ldots, m\}$. Informally, $c_1(u)$ aims at proving to node u that it is node labeled $\mathrm{index}(u)$ in the m-node graph with adjacency matrix $M(u)$, and that the whole input data is $\mathrm{data}(u)$. We denote by n the number of nodes of the actual graph G.

For a legal configuration $(G, x) \in \mathcal{L}$, given c_1, the certificate c_2 is then defined as follows. It is based on the identification of a few specific nodes, that we call *witnesses*. Intuitively, a witness is a node enabling to demonstrate that the structure of the configuration (G, x) does not fit with the given certificate c_1. Let $\mathrm{dist}(u, v)$ denote the distance between any two nodes u and v in the actual network G, that is, $\mathrm{dist}(u, v)$ equals the number of edges of a shortest path between u and v in G. A certificate $c_2(u)$ is of the form $(f(u), \sigma(u))$ where $f(u) \in \{0, \ldots, 4\}$ is a flag, and $\sigma(u) \in \{0, 1\}^*$ depends on the value of the flag.

Case 0. There are two adjacent nodes $v \neq v'$ such that $(M(v), \mathrm{data}(v)) \neq (M(v'), \mathrm{data}(v'))$, or there is at least one node v in which $c_1(v)$ cannot be read as a triple $(M(v), \mathrm{data}(v), \mathrm{index}(v))$. Then we set one of these nodes as witness w, and we set $c_2(u) = (0, \mathrm{dist}(u, w))$ at every node u.

Otherwise, i.e., if the pair $(M(u), \mathrm{data}(u))$ is identical to some pair (M, data) at every node u:

Case 1. (G, x) is isomorphic to (M, data), preserving the inputs, denoted by $(G, x) \sim (M, \mathrm{data})$, and index() represents the isomorphism. Then we set $c_2(u) = (1)$ at every node u.

Case 2. index() is not injective. Then we set $c_2(u) = (2, i, d(u, w), d(u, w'))$ where $i \in \{1, \ldots, m\}$, and $w \neq w'$ are two distinct nodes such that $\text{index}(w) = \text{index}(w') = i$. These two nodes w and w' are both witnesses.

Case 3. $n < m$ and index() is injective. Then we set $c_2(u) = (3, i)$ where $i \in \{1, \ldots, m\}$ is such that $\text{index}(v) \neq i$ for every node v.

Case 4. $n = m$ and index() is injective, but (G, x) is not isomorphic to (M, data). Then we set as witness a node w whose neighborhood in (G, x) does not fit with what it should be according to (M, data), and we set $c_2(u) = (4, d(u, w))$ for every node u.

The local verification algorithm $\mathcal{A}$ then proceeds as follows. First, every node u checks whether its flag $f(u)$ in $c_2(u)$ is identical to all the ones of its neighbors, and between 0 and 4. If not, then u rejects. Otherwise, u carries on executing the verification procedure. Its behavior depends on the value of its flag.

- If $f(u) = 0$, and if the given distance is greater than zero, then u rejects whenever none of its neighbors have a distance to the witness that is smaller than its own, otherwise it accepts. If the given distance is zero, then u accepts whenever there is indeed an inconsistency with its certificate c_1 (i.e., its certificate c_1 cannot be read as a pair matrix-data, or its certificate c_1 is distinct from the one of its neighbors), otherwise it rejects.
- If $f(u) = 1$, then u accepts or rejects according to whether $(M(u), \text{data}(u)) \in \mathcal{L}$ (recall that, by definition, we consider only distributed languages $\mathcal{L}$ that are Turing-decidable).
- If $f(u) = 2$, then u checks that it has the same index i in its certificate c_2 as all its neighbors. If that is not the case, then it rejects. Otherwise, it checks each of the two distances in its certificate c_2 separately, each one as in the case where $f(u) = 0$. A node with one of the two distances equal to 0 also checks that its c_1 index is equal to the index i in c_2. If that is not the case, or if its two distances are equal to 0, then it rejects. If all the test are passed, then u accepts.
- If $f(u) = 3$, then u accepts if and only if it has the same index i in its c_2 certificate as all its neighbors, and $\text{index}(u) \neq i$.
- If $f(u) = 4$, then u checks the distances as in the case where $f(u) = 0$. A node with distance 0 also checks that its neighborhood in the actual configuration (G, x) is not what it should be according to (M, data). It accepts or rejects accordingly.

To prove the correctness of this Algorithm $\mathcal{A}$, let us first consider a legal configuration $(G, x) \in \mathcal{L}$. We show that the way c_2 is defined guarantees that all nodes accept, because c_2 correctly pinpoints inconsistencies in c_1, witnessing any attempt of c_1 to certify that the actual configuration is illegal. Indeed, in Case 0, by the setting of c_2, all nodes but the witness accept. Also, the witness itself accepts because it does witness the inconsistency of the c_1 certificate. In Case 1, all nodes accept because $(G, x) \sim (M, \text{data})$ and $(G, x) \in \mathcal{L}$. In Case 2, by the setting of c_2, all nodes but the witnesses accept, and the witnesses accept too because each one checks that it is the vertex with index i in M. In Case 3, all nodes accept by construction of the certificate c_2. Finally, in Case 4, by the setting of c_2, all nodes but the witness accept. Also, the witness itself accepts because, as

in Case 0, it does witness the inconsistency of the c_1 certificate. So, in all cases, all nodes accept, as desired.

We are now left with the case of illegal configurations. Let $(G, x) \notin \mathcal{L}$ be such an illegal configuration. We set $c_1(u) = (M, \text{data}, \text{index}(u))$ where $(M, \text{data}) \sim (G, x)$ and $\text{index}(u)$ is the index of node u in the adjacency matrix M and the array data. We show that, for any certificate c_2, at least one node rejects. Indeed, for all nodes to accept, they need to have the same flag in c_2. This flag cannot be 1 because, if $f(u) = 1$ then u checks the legality of (M, data). In all other cases, the distance checking should be passed at all nodes for them to accept. Thus, the flag is distinct from 0 and 4 because every radius-1 ball in (G, x) fits with its description in (M, data). Also, the flag is distinct from 2 because there are no two distinct nodes with the same index i in the c_1 certificate. Finally, also the flag is distinct from 3, because, by the setting of c_1, every index in $\{1, \ldots, n\}$ appears at some node, and this node would reject. Hence, all cases lead to contradiction, that is, not all nodes can accept, as desired. □

To conclude the section, let us define a simple decision task in $\Pi_2^{\mathsf{local}} \setminus \mathsf{NLD}$. Let EXTS, which stands for "exactly two selected" be the following language. We set

$$(G, x) \in \text{EXTS} \iff (\forall u \in V(G),\, x(u) \in \{\bot, \top\}) \text{ and } (|\{u \in V(G) : x(u) = \top\}| = 2).$$

Proving that EXTS $\notin$ NLD is easy using the following characterization of NLD. (See Figure 2 for an example). Let $t \geq 1$. A configuration (G', x') is a t-lift of a configuration (G, x) if there exists a mapping $\phi : V(G') \to V(G)$ such that, for every $u \in V(G')$, ϕ induces an isomorphism between $B_G(\phi(u), t)$ and $B_{G'}(u, t)$ preserving inputs (i.e., $x(\phi(u)) = x'(u)$ for all $u \in V(G')$). A distributed language $\mathcal{L}$ is closed under lift if there exists $t \geq 1$ such that, for every (G, x), we have $(G, x) \in \mathcal{L}$ implies $(G', x') \in \mathcal{L}$ for every (G', x') that is a t-lift of (G, x).

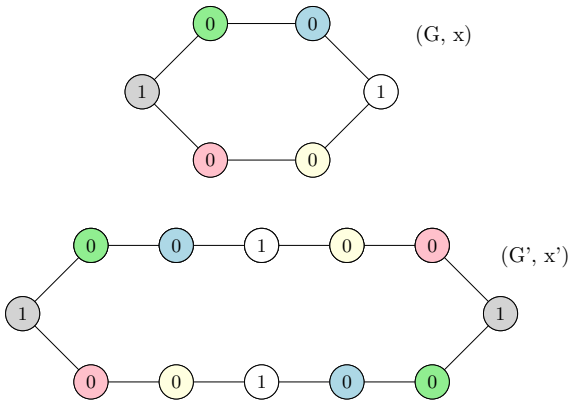

Figure 2: *An example illustrating why* EXTS $\notin$ NLD. *The coloring denotes a mapping, i.e., each node of G' colored c is mapped on the node of G having the same color c. The nodes marked 1 are the selected nodes. In (G, x), every node should accept while, in (G', x'), at least one node should reject. However, the nodes in G' have the same view at distance 2 as their image in G. Therefore, there cannot be a 2-round verification procedure for* EXTS. *This generalizes trivially to t rounds.*

Lemma 1 ([6]) NLD *is the class of distributed languages closed under lift.*

Since EXTS is not closed under lift, it results from Lemma 1 that EXTS $\notin$ NLD.

3 On the impact of the last universal quantifier

In this section, we prove the part of Theorem 1 related to the two classes Π_1^{local} and $\Sigma_2^{\mathsf{local}}$. These two classes have in common that the universal quantifier is positioned last. It results that these two classes seem to be limited, as witnessed by the following two propositions.

Proposition 5 $\Sigma_2^{\mathsf{local}} = \mathsf{NLD}$.

Proof. By definition, $\mathsf{NLD} = \Sigma_1^{\mathsf{local}} \subseteq \Sigma_2^{\mathsf{local}}$. Hence, by Lemma 1, it is sufficient to prove that, for any $\mathcal{L} \in \Sigma_2^{\mathsf{local}}$, $\mathcal{L}$ is closed under lift. If $\mathcal{L} \in \Sigma_2^{\mathsf{local}}$ then let $\mathcal{A}$ be a local algorithm establishing the membership of $\mathcal{L}$ in $\Sigma_2^{\mathsf{local}}$. $\mathcal{A}$ is satisfying the following. For every $(G, x) \in \mathcal{L}$,

$$\exists c_1, \forall c_2, \forall \mathrm{id}, \forall u \in V(G), \mathcal{A}_{G,x,c_1,c_2,\mathrm{id}}(u) = \text{accept}.$$

Let t be the radius of $\mathcal{A}$, and assume, for the purpose of contradiction, that $\mathcal{L}$ is not closed under lift. There exists $(G, x) \in \mathcal{L}$, and a t-lift (G', x') of (G, x) with $(G', x') \notin \mathcal{L}$. Let $\phi : V(G') \to V(G)$ be this t-lift. Note that the t-balls in (G, x) are identical to the t-balls in (G', x') by definition of a t-lift. Let c_1 be the distributed certificate that makes $\mathcal{A}$ accept (G, x) at all nodes, for all certificates c_2. Let c_1' be the distributed certificate for (G', x') defined by $c_1'(u) = c_1(\phi(u))$. Since, with certificate c_1, $\mathcal{A}$ accepts at all nodes of G, for every certificate c_2, and for every identity assignment, it follows that, with certificate c_1', $\mathcal{A}$ accepts at all nodes of G', for every certificate c_2', and for every identity assignment, contradicting the correctness of $\mathcal{A}$. Therefore, $\mathcal{L}$ is closed under lift. Thus, $\Sigma_2^{\mathsf{local}} \subseteq \mathsf{NLD} = \Sigma_1^{\mathsf{local}} \subseteq \Sigma_2^{\mathsf{local}}$, and the result follows. □

To show that $\Pi_1^{\mathsf{local}} \neq \mathsf{NLD}$, we consider the language ALTS, which stands for "at least two selected". (Note that ALTS is the complement of the language AMOS introduced in [7], where AMOS stands for "at most one selected"). We set

$$(G, x) \in \text{ALTS} \iff (\forall u \in V(G),\, x(u) \in \{\bot, \top\}) \text{ and } (|\{u \in V(G) : x(u) = \top\}| \geq 2).$$

To separate NLD and Π_1^{local}, we show that $\text{ALTS} \in \mathsf{NLD} \setminus \Pi_1^{\mathsf{local}}$.

Proposition 6 $\Pi_1^{\mathsf{local}} \subset \mathsf{NLD}$ (the inclusion is strict).

Proof. By Lemma 1, to establish $\Pi_1^{\mathsf{local}} \subseteq \mathsf{NLD}$, it is sufficient to prove that, for any $\mathcal{L} \in \Pi_1^{\mathsf{local}}$, $\mathcal{L}$ is closed under lift. The arguments are exactly similar to the ones used in the proof of Proposition 5 without even the need to lift a first set of certificates. To show that $\Pi_1^{\mathsf{local}} \neq \mathsf{NLD}$, we consider the language ALTS. We have $\text{ALTS} \in \mathsf{NLD}$ because ALTS is closed under lift. However, $\text{ALTS} \notin \Pi_1^{\mathsf{local}}$. Indeed, assume that there exists a local algorithm $\mathcal{A}$ for $\text{ALTS} \in \Pi_1^{\mathsf{local}}$. Then, consider a cycle C_n where all inputs are $\bot$. By assumption, there exists a certificate c_1 such that at least one node v rejects. Now consider the same input instance modified such that two arbitrary nodes not in the ball of v have $\top$ as input. In this case, every node must accept given any certificate, but v with c_1 will reject since it has the same local view and the same certificate as before. □

While Π_1^{local} is in NLD, the universal quantifier adds some power compared to LD. We show that $\mathsf{LD} \neq \Pi_1^{\mathsf{local}}$ by exhibiting a language in $\Pi_1^{\mathsf{local}} \setminus \mathsf{LD}$. Note that the existence of this language is not straightforward as it must involve Turing-computability issues. Indeed, if one does not insist on the fact that the local algorithm must be a Turing-computable function, then the two classes LD and Π_1^{local} would be identical. For instance, given a t-round algorithm $\mathcal{A}$ deciding a language $\mathcal{L}$

in Π_1^{local}, one could define the following mechanism for deciding the same language in LD. Given a t-ball B centered at u, node u accepts if and only if there are no certificate assignments to the nodes of B that could lead $\mathcal{A}$ to reject at u. However, this mechanism is not a Turing-computable function. Interestingly, NLD would still not collapse to LD even if using non Turing-computable decision mechanisms. To see why, assume that we are given the ability to try all possible certificates of an NLD algorithm $\mathcal{A}$. The simple decision mechanism at every node u consisting in rejecting at u as long as $\mathcal{A}$ rejects one of the certificates at u, which works fine for Π_1^{local}, does not work for NLD. Indeed, a node that rejects a configuration for some certificate cannot safely reject because it might be a legal configuration with an incorrect certificate. We show that, in fact, $\Pi_1^{\mathsf{local}} \setminus \mathsf{LD} \neq \emptyset$.

Proposition 7 $\mathsf{LD} \subset \Pi_1^{\mathsf{local}}$ *where the inclusion is strict.*

Proof. We describe the distributed language ITER, which stands for "iteration". Let M be a Turing machine, and let us enumerate lexicographically all the states of the system tape-machine where M starts its execution on the blank tape, with the head at the beginning of the tape. We define the function $f_M : \mathbb{N} \to \mathbb{N}$ by $f_M(0) = 0$, $f_M(1) = 1$, and, for $i > 1$, $f_M(i)$ equal to the index of the system-state after one step of M from system-state i. In other words, f_M is defined so that 1 denotes the rejecting state for any tape content, and any head position, while 0 denotes the accepting state for any tape content, and any head position. All other configurations uniquely identify the entire tape content, the head position, and the current non halting state. In essence, when the machine switches from some configuration $i > 1$ to another configuration $j > 1$, we keep track of the tape content, and of the head position. If the machine halts, then we discard the tape content as well as the head position, and we simply set $f_M^{(i)}$ equal to 0 or 1 accordingly, where $g^{(i)}$ denotes the ith iterated of a function g.

We define ITER as the collection of configurations (G, x) representing two sequences of iterations of a function f_M on different inputs a and b (see Figure 3).

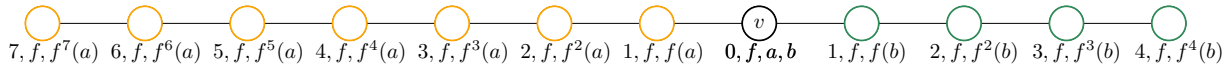

Figure 3: *An illustration of the distributed language* ITER. *The subpaths L and R are depicted in orange and green, respectively.*

More precisely, let M be a Turing machine, and let a and b be two non-negative integers. We define the following family of configurations (see Figure 3). A configuration in ITER mainly consists of a path P with a special node v, called the *pivot*, identified in this path. So $P = LvR$ where L and R are subpaths, respectively called left path and right path. All nodes u of the path are given in input the machine M and the distance $d(u) = \mathrm{dist}(u, v)$ from the pivot ($d(v) = 0$). The pivot v is also given a and b as inputs. Also, the node of the left path (resp., right path) at distance i from v is given a value $x(u) = f_{i,L}$ (resp., $f_{i,R}$) as input. To be in the language, it is required that, for every i, $f_{i,L} = f_M^{(i)}(a)$ and $f_{i,R} = f_M^{(i)}(b)$. The configuration is in the language if and only if the f-values at both extremities of the path P are 0 or 1, and at least one of them is equal to 0. That is, the configuration is in the language if and only if:

$$(f_{|L|,L} \in \{0, 1\} \text{ and } f_{|R|,R} \in \{0, 1\}) \text{ and } (f_{|L|,L} = 0 \text{ or } f_{|R|,R} = 0). \tag{2}$$

A configuration $(G, x) \in$ ITER if and only if (G, x) satisfies all the above conditions with respect to

the given machine M. Hence, a configuration is in the language if the machine terminates on both inputs a and b, and accepts at least one of these two inputs.

Let us consider a weaker version of ITER, denoted by ITER$^-$ where the condition of Eq. (2) is replaced by just:

$$f_{|L|,L} \in \{0,1\} \text{ and } f_{|R|,R} \in \{0,1\}.$$

We show that ITER$^-$ $\in$ LD. First, each node u checks that it has the same machine M as its neighbors, and that it has at most two neighbors. Then, each node checks the consistency of $d(u)$, that is:

- if $d(u) = 0$, then u must have exactly two neighbors w_1, w_2 such that $d(w_1) = d(w_2) = 1$;
- if u has exactly two neighbors, then it must hold that one neighbor, w_1, satisfies $d(w_1) = d(u) - 1$. and another neighbor, w_2, satisfies $d(w_2) = d(u) + 1$;
- if u has exactly one neighbor w, then it must hold that $d(w) = d(u) + 1$;

The pivot checks that one of its neighbors, w_1, satisfies $x(w_1) = f_M(a)$, and that its other neighbor, w_2, satisfies $x(w_2) = f_M(b)$. Also, every node u, but the pivot and its neighbors, checks that $x(u) = f_M(x(w))$, where w is the node satisfying $d(w) = d(u) - 1$. Finally, every node u having just one neighbor should satisfy $x(u) \in \{0,1\}$. A node rejects if at least one of these tests is not passed, otherwise it accepts.

Moreover, ITER $\in \Pi_1^{\mathsf{local}}$. To see why, we describe a local algorithm $\mathcal{A}$ using certificates. The algorithm first checks whether $(G,x) \in$ ITER$^-$. All nodes but the pivot v decide according to this checking. If the pivot rejected $(G,x) \in$ ITER$^-$, then it rejects in $\mathcal{A}$ as well. Otherwise, it carries on its decision process by interpreting its certificate as a non-negative integer k, and accepts in $\mathcal{A}$ unless $f_M^{(k)}(a) = 1$ and $f_M^{(k)}(b) = 1$.

To show the correctness of $\mathcal{A}$, let $(G,x) \in$ ITER. We have $f_{|L|,L} = 0$ or $f_{|R|,R} = 0$, i.e., $f_M^{(|L|)}(a) = 0$ or $f_M^{(|R|)}(b) = 0$. W.l.o.g., assume $f_M^{(|L|)}(a) = 0$. If $k \geq |L|$ then $f_M^{(k)}(a) = 0$ since $f_M(0) = 0$, and thus v accepts. If $k < |L|$ then $f_M^{(k)}(a) \neq 1$ since $f_M(1) = 1$, and thus v accepts. Therefore, all certificates lead to acceptance. Let us now consider $(G,x) \notin$ ITER. If $(G,x) \notin$ ITER$^-$ then at least one node rejects, independently of the certificate. So, we assume that $(G,x) \in$ ITER$^-$ $\setminus$ ITER. Thus, $f_M^{(|L|)}(a) = 1$ and $f_M^{(|R|)}(b) = 1$. The certificate is set to $k = \max\{|L|, |R|\}$. Let us assume, w.l.o.g., that $k = |L| \geq |R|$. By this setting, we have $f_M^{(k)}(a) = 1$. Moreover, since $k \geq |R|$, and since $f_M(1) = 1$, we get that $f_M^{(k)}(b) = 1$. Therefore, $\mathcal{A}$ rejects, as desired. Thus, ITER $\in \Pi_1^{\mathsf{local}}$.

It remains to show that ITER $\notin$ LD. Let us assume, for the purpose of contradiction, that there exists a t-round algorithm $\mathcal{A}$ deciding ITER. Since ITER$^-$ $\in$ LD, this algorithm is able to distinguish an instance with $f_M^{(|L|)}(a) = 1$ and $f_M^{(|R|)}(b) = 1$ from instances in which $f_M^{(|L|)}(a) \neq 1$ or $f_M^{(|R|)}(b) \neq 1$. Observe that a node at distance greater than t from the pivot can gather information related to only one of the two inputs a and b. Therefore, the distinction between the case $f_M^{(|L|)}(a) = 1$ and $f_M^{(|R|)}(b) = 1$ and the case $f_M^{(|L|)}(a) \neq 1$ or $f_M^{(|R|)}(b) \neq 1$ can only be made by a node at distance at most t from the pivot. Therefore, by simulating $\mathcal{A}$ at all nodes in the ball of radius t around v, with identities between 1 and the size of the ball of radius $2t$ around the pivot, a sequential algorithm can determine, given a Turing machine M, and given a and b, whether there exist ℓ and r such that $f_M^{(\ell)}(a) = f_M^{(r)}(b) = 1$ or not, which is actually Turing undecidable. This contradiction implies that, indeed, ITER $\notin$ LD. □

4 Complementary classes

Given a class $\mathcal{C}$ of distributed languages, the class co-$\mathcal{C}$ is composed of all distributed languages $\mathcal{L}$ such that $\bar{\mathcal{L}} \in \mathcal{C}$, where $\bar{\mathcal{L}} = \{(G,x) \notin \mathcal{L}\}$. For instance, since Π_1^{local} is the class of languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ such that, for every configuration (G,x),

$$\begin{array}{rcl} (G,x) \in \mathcal{L} & \Rightarrow & \forall c, \forall \mathrm{id}, \forall u \in V(G), \mathcal{A}_{G,x,c,\mathrm{id}}(u) \text{ accepts;} \\ (G,x) \notin \mathcal{L} & \Rightarrow & \exists c, \forall \mathrm{id}, \exists u \in V(G), \mathcal{A}_{G,x,c,\mathrm{id}}(u) \text{ rejects;} \end{array}$$

we get that co-Π_1^{local} is the class of languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ such that, for every configuration (G,x),

$$\begin{array}{rcl} (G,x) \in \mathcal{L} & \Rightarrow & \exists c, \forall \mathrm{id}, \exists u \in V(G), \mathcal{A}_{G,x,c,\mathrm{id}}(u) = \text{accepts;} \\ (G,x) \notin \mathcal{L} & \Rightarrow & \forall c, \forall \mathrm{id}, \forall u \in V(G), \mathcal{A}_{G,x,c,\mathrm{id}}(u) = \text{rejects.} \end{array}$$

Note in particular, that the rejection must now be unanimous, while the acceptance requires only one node to accept. Let us define the following two languages: each input to every node belongs to $\{\text{true}, \text{false}\} = \{1, 0\}$, and a configuration is in AND (resp., in OR) if and only if the logical conjunction (resp., disjunction) of the inputs is true. That is,

$$\begin{array}{rcl} \textsc{and} & = & \{(G,x) : \forall u \in V(G), x(u) \in \{\text{true}, \text{false}\}, \text{ and } \bigwedge_{u \in V(G)} x(u) = \text{true}\}; \\ \textsc{or} & = & \{(G,x) : \forall u \in V(G), x(u) \in \{\text{true}, \text{false}\}, \text{ and } \bigvee_{u \in V(G)} x(u) = \text{true}\}. \end{array}$$

These two languages enable to separate LD from its co-class. Indeed, OR $\notin \mathsf{LD}$ as every node that sees only zeros must accept because there might exist far away nodes with input 1. Hence, an all-0 instance would be accepted, which is incorrect. Instead, AND $\in \mathsf{LD}$: every node accepts if and only if its input is 1. The class $\mathsf{LD} \cap$ co-LD is quite restricted. Nevertheless, it contains distributed languages such as DIAM$_k$, the class of graphs with diameter at most k, for any fixed k. We have the following separation.

Proposition 8 OR $\in$ co-$\mathsf{LD} \setminus \Pi_1^{\mathsf{local}}$, and AND $\in \mathsf{LD} \setminus$ co-Π_1^{local}.

Similarly, the languages ALTS and AMOS introduced in the proof of Proposition 6 enable to separate NLD from its co-class. Indeed, ALTS $= \overline{\text{AMOS}}$, ALTS is closed under lift, and AMOS is not closed under lift. Moreover, consider the language EXTS defined at the end of Section 2. Both EXTS and $\overline{\text{EXTS}}$ are not closed under lift. So, overall, by Lemma 1, we get:

Proposition 9 ALTS $\in \mathsf{NLD} \setminus$ co-NLD, AMOS $\in$ co-$\mathsf{NLD} \setminus \mathsf{NLD}$, and EXTS $\notin \mathsf{NLD} \cup$ co-NLD.

More interesting is the position of Π_1^{local} w.r.t. NLD and co-NLD. First, we point out that there are problems in $\mathsf{NLD} \cap$ co-NLD. For this purpose, we consider the aforementioned language

$$\text{TREE} = \{(G,x) : G \text{ is a tree}\}.$$

On the one hand, TREE $\in \mathsf{NLD}$, simply because a tree cannot be lifted. Indeed, notice that to be lifted, or to be the result of a lift (i.e., to be a covering), a graph must contain cycles — recall that we are considering solely connected simple undirected graphs, and thus only connected lifts. On the other hand, TREE $\in$ co-NLD since a tree cannot result from a lift. (By Lemma 1, co-NLD is the class of languages $\mathcal{L}$ closed down under lift, i.e., if $(G,x) \in \mathcal{L}$ is the lift of a configuration (G',x'), then we have $(G',x') \in \mathcal{L}$).

Proposition 10 $\Pi_1^{\mathsf{local}} \cup \text{co-}\Pi_1^{\mathsf{local}} \subset \mathsf{NLD} \cap \text{co-}\mathsf{NLD}$, where the inclusion is strict.

Proof. From Proposition 6, we know that $\Pi_1^{\mathsf{local}} \subset \mathsf{NLD}$. We prove that $\text{co-}\Pi_1^{\mathsf{local}} \subset \mathsf{NLD}$. Let $\mathcal{L} \in \text{co-}\Pi_1^{\mathsf{local}}$, and let $\mathcal{A}$ be a t-round algorithm deciding $\mathcal{L}$. Let $(G, x) \in \mathcal{L}$, and let c be a certificate such that $\mathcal{A}$ accepts at some node. Let (G', x') be a t-lift of (G, x), and lift c into c' accordingly. Then $\mathcal{A}$ also accepts (G', x'), which implies that $\mathcal{L}$ is closed under t-lift, and thus, by Lemma 1, $\mathcal{L} \in \mathsf{NLD}$. Therefore $\Pi_1^{\mathsf{local}} \cup \text{co-}\Pi_1^{\mathsf{local}} \subseteq \mathsf{NLD} \cap \text{co-}\mathsf{NLD}$.

To prove that the inclusion is strict, we consider the language TREE that we know to be in $\mathsf{NLD} \cap \text{co-}\mathsf{NLD}$. To see why $\text{TREE} \notin \Pi_1^{\mathsf{local}}$, consider a path and a cycle. If TREE could be decided in Π_1^{local}, then the center nodes of the path must accept for all certificates and for any identity-assignment. Hence, all degree-2 nodes that see only degree-2 nodes in their neighborhoods accept, for all certificates. As a consequence, the cycle will be incorrectly accepted for all certificates. Somewhat similarly, if TREE could be decided in $\text{co-}\Pi_1^{\mathsf{local}}$, say in t-rounds, then it would mean that, in a path, the node(s) that accept(s) (with the appropriate certificate) can only be at distance at most t from an extremity of the path. Indeed, otherwise, one could close the path and create a cycle that will still be accepted. So, by gluing two paths P and P' of length at least $2t$ to two antipodal nodes of a cycle C, and by giving to the nodes of P and P' the certificates that lead each of them to be accepted, this graph would be incorrectly accepted. □

5 Complete problems

In this section, we prove Theorem 2. Let G be a connected graph, and U be a set (typically, $U = \{0,1\}^*$). Let $e : V(G) \to U$, and let $\mathcal{S} : V(G) \to 2^{2^U}$. That is, e assigns an element $e(u) \in U$ to every node $u \in V(G)$, and $\mathcal{S}$ assigns a collection of sets $\mathcal{S}(u) = \{S_1(u), \ldots, S_{k_u}(u)\}$ to every node $u \in V(G)$, with $k_u \geq 1$ and $S_i : V(G) \to 2^U$ for every $i \geq 1$. We say that $\mathcal{S}$ covers e if and only if there exists $u \in V(G)$, and there exists $i \in \{1, \ldots, k_u\}$, such that $S_i(u) = \{e(v) \mid v \in V(G)\}$. In [7], the authors defined the language

$$\text{COVER} = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{S}(u), e(u)) \text{ such that } \mathcal{S} \text{ covers } e\}$$

and proved that COVER is the "most difficult decision task", in the sense that every distributed language can be locally reduced to COVER. However COVER is closed under lift as lifting does not create new elements and preserves the sets. Therefore, by Lemma 1, $\text{COVER} \in \mathsf{NLD}$.[1] This is in contradiction with the claim in [7] regarding the hardness of COVER. The reason for this contradiction is that the local reduction used in [7] for reducing any language to COVER is too strong. Indeed, it transforms a configuration (G, x) into a configuration (G, x') where the certificates used for proving x' may depend on the identities of the nodes in G. This is in contradiction with the definitions of the classes $\Sigma_k^{\mathsf{local}}$ and Π_k^{local}, $k \geq 0$, for which the certificates must be independent of the identity assignment. In this section, we show that completeness results can be obtained using a more constrained notion of reduction which preserves the membership to the classes.

Recall from [7] that a local reduction of $\mathcal{L}$ to $\mathcal{L}'$ is a local algorithm $\mathcal{R}$ which maps any configuration (G, x) to a configuration (G, y), where $y = R(G, x, \text{id})$ may depend on the identity assignment id, such that: $(G, x) \in \mathcal{L}$ if and only if, for every identity assignment id to the nodes

[1] In fact, one can show that there exists a local verification algorithm for COVER using certificates of size quasi linear in n whenever the ground set U is of polynomial size (see Proposition 14 in Appendix A).

of G, $(G, y) \in \mathcal{L}'$ where $y = \mathcal{R}(G, x, \mathrm{id})$. Ideally, we would like $\mathcal{R}$ to be *identity-oblivious*, that is, such that the output of each node does not depend on the identity assignment, but this appears to be too restrictive. So, instead, we use a concept somewhat intermediate between identity-oblivious reduction and the unconstraint reduction in [7].

Definition 1 Let $\mathcal{C}$ be a class of distributed languages, and let $\mathcal{L}$ and $\mathcal{L}'$ be two distributed languages. Let $\mathcal{A}$ be a $\mathcal{C}$-algorithm deciding $\mathcal{L}'$, and let $\mathcal{R}$ be a local reduction of $\mathcal{L}$ to $\mathcal{L}'$. We say that $(\mathcal{R}, \mathcal{A})$ is *label-preserving* for $(\mathcal{L}, \mathcal{L}')$ if and only if, for any configuration (G, x), the existential certificates used by the prover in $\mathcal{A}$ for (G, y) where $y = \mathcal{R}(G, x, \mathrm{id})$ are the same for all identity assignments id to G.

The following result shows that the notion of reduction in Definition 1 preserves the classes of distributed languages.

Lemma 2 *Let $\mathcal{C}$ be a class of distributed languages. Let $\mathcal{L}$ and $\mathcal{L}'$ be two distributed languages with $\mathcal{L}' \in \mathcal{C}$, and let $(\mathcal{R}, \mathcal{A})$ be a label-preserving local reduction for $(\mathcal{L}, \mathcal{L}')$. Then $\mathcal{L} \in \mathcal{C}$.*

Proof. We describe a local algorithm $\mathcal{B}$ for deciding $\mathcal{L}$ in $\mathcal{C}$. In essence, $\mathcal{B} = \mathcal{A} \circ R$. More precisely, let (G, x) be a configuration, with an arbitrary identity assignment id, and let $y = R(G, x, \mathrm{id})$. Let c be a certificate assigned by the prover in $\mathcal{A}$ for configuration (G, y). (Note that this certificate may depend on some previously set certificates, as in, e.g., Π_1^{local}). The certificate assigned by the prover in $\mathcal{B}$ for configuration (G, x) is c. The algorithm $\mathcal{B}$ then proceeds as follows. Given (G, x), it computes (G, y) using R, and then applies $\mathcal{A}$ on (G, y) using the certificates constructed by the prover in $\mathcal{B}$. Algorithm $\mathcal{B}$ then outputs the decision taken by $\mathcal{A}$. Since R preserves the membership to the languages, and since the certificates assigned by the prover in $\mathcal{A}$ for configurations resulting from the application of R are independent of the identity assignment, the certificates chosen under the identity assignment id are also appropriate for any other identity assignment id$'$. This guarantees the correctness of $\mathcal{B}$, and thus $\mathcal{L} \in \mathcal{C}$. □

We now exhibit a language that is among the hardest decision tasks, under local label-preserving reductions. In the following decision task, every node u of a configuration (G, x) is given a family $\mathcal{F}(u)$ of configurations, each described by an adjacency matrix representing a graph, and a 1-dimensional array representing the inputs to the nodes of that graph. In addition, every node u has an input string $x'(u) \in \{0, 1\}^*$. Hence, (G, x') is also a configuration. The actual configuration (G, x) is legal if (G, x') is missing in all families $\mathcal{F}(u)$ for every $u \in V(G)$, i.e., $(G, x') \notin \mathcal{F}$ where $\mathcal{F} = \cup_{u \in V(G)} \mathcal{F}(u)$. In short, we consider the language

$$\textsc{miss} = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{F}(u), x'(u)) \text{ and } (G, x') \notin \mathcal{F}\}$$

We show that MISS is among the hardest decision tasks, under local label-preserving reductions. Note that MISS $\notin \mathsf{NLD}$ (it is not closed under lift: it may be the case that $(G, x') \notin \mathcal{F}$ but a lift of (G, x') is in $\mathcal{F}$).

Proposition 11 MISS is Π_2^{local}-complete under local label-preserving reductions.

Proof. Let $\mathcal{L}$ be a distributed language. We describe a local label-preserving reduction $(R, \mathcal{A})$ for $(\mathcal{L}, \textsc{miss})$ with respect to Π_2^{local}.

In essence, the local algorithm $\mathcal{A}$ for deciding MISS in Π_2^{local} is the generic algorithm described in the proof of Proposition 4. Recall that, in this generic algorithm, on a legal configuration (G, x), the existential c_2 certificate in $\mathcal{A}$ is pointing to an inconsistency in the given c_1 certificate which is supposed to describe the configuration (G, x). And, on an illegal configuration (G, x), the existential c_1 certificate in $\mathcal{A}$ does provide an accurate description of the configuration (G, x). For the purpose of label-preservation, we slightly modify the generic algorithm for MISS. Instead of viewing c_1 as a description of the configuration (G, x), the algorithm views it as a description of (G, x') where, at each node u, $x'(u)$ is the second item in $x(u)$ (the first item is the family $\mathcal{F}(u)$). The algorithm is then exactly the same as the generic algorithm with the only modification that the test when the flag $f(u) = 1$ is not regarding whether $(G, x') \in$ MISS, but whether $(G, x') \notin \mathcal{F}(u)$. On a legal configuration, all nodes accept. On an illegal instance, a node with $(G, x') \in \mathcal{F}(u)$ rejects.

The reduction R from $\mathcal{L}$ to MISS proceeds as follows, in a way similar to the one in [7]. A node u with identity $\mathrm{id}(u)$ and input $x(u)$ computes its *width* $\omega(u) = 2^{|\mathrm{id}(u)|+|x(u)|}$ where $|s|$ denotes the length of a bit-string s. Then u generates all configurations $(H, y) \notin \mathcal{L}$ such that H has at most $\omega(u)$ nodes and $y(v)$ has value at most $\omega(u)$, for every node v of H. It places all these configurations in $\mathcal{F}(u)$. The input $x'(u)$ is simply $x'(u) = x(u)$. If $(G, x) \in \mathcal{L}$, then $(G, x) \notin \mathcal{F}$ since only illegal instances are in $\mathcal{F}$, and thus $(G, R(G, x)) \in$ MISS. Conversely, if $(G, x) \notin \mathcal{L}$, then $(G, R(G, x)) \notin$ MISS. Indeed, there exists at least one node u with identity $\mathrm{id}(u) \geq n$, which guarantees that u generates the graph G. If no other node u' has width $\omega(u') > n$ then u generates $(G, x) \in \mathcal{F}(u)$. If there exists a node u' with $\omega(u') > n$ then u' generates $(G, x) \in \mathcal{F}(u')$. In each case, we have $(G, x) \in \mathcal{F}$, and thus $(G, R(G, x)) \notin$ MISS.

It remains to show that the existential certificate used in $\mathcal{A}$ for all configurations $(G, R(G, x))$ are the same for any given (G, x), independently of the identity assignment to G used to perform the reduction R. This directly follows from the nature of $\mathcal{A}$ since the certificates do not depend on the families $\mathcal{F}(u)$'s but only on the bit strings $x'(u)$'s. □

The following language is defined as MISS by replacing $\mathcal{F}$ by the closure under lift $\mathcal{F}^\uparrow$ of $\mathcal{F}$. That is, $\mathcal{F}^\uparrow$ is composed of $\mathcal{F}$ and all the lifts of the configurations in $\mathcal{F}$.

$$\text{MISS}^\uparrow = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{F}(u), x'(u)) \text{ and } (G, x') \notin \mathcal{F}^\uparrow\}$$

We show that $\text{MISS}^\uparrow$ is among the hardest decision tasks in NLD.

Proposition 12 $\text{MISS}^\uparrow$ is NLD-complete (and $\overline{\text{MISS}^\uparrow}$ is co-NLD-complete) under label-preserving reduction.

Proof. We do have $\text{MISS}^\uparrow \in \mathsf{NLD}$ because $\text{MISS}^\uparrow$ is closed under lift. Let $\mathcal{L} \in \mathsf{NLD}$. The reduction R from $\mathcal{L}$ to $\text{MISS}^\uparrow$ is the same as the one in the proof of Proposition 11. We describe a local algorithm for deciding $\text{MISS}^\uparrow$ in NLD which is label-preserving with respect to R. The certificate $c(u)$ is a description of (G, x') of the form $(M(u), \mathrm{data}(u), \mathrm{index}(u))$ as certificate c_1 in the proof of Proposition 4. This guarantees label-preservation with respect to R. For the verification part, each node u checks whether $(M(u), \mathrm{data}(u), \mathrm{index}(u))$ fits with its local neighborhood. If no, it rejects. Otherwise, it checks whether $(M(u), \mathrm{data}(u)) \notin \mathcal{F}(u)^\uparrow$, accepts if yes, and rejects otherwise. On a legal configuration $(G, x) \in \text{MISS}^\uparrow$, with the correct certificate $c = (G, x')$, all nodes accept. On an illegal configuration $(G, x) \notin \text{MISS}^\uparrow$, there are two cases. If $(M(u), \mathrm{data}(u))$ is neither (G, x') nor a lift of (G, x'), then some node will detect an inconsistency, and reject. If $(M(u), \mathrm{data}(u))$ is an accurate description of (G, x') or of a lift of (G, x'), then some node will detect that $(M(u), \mathrm{data}(u)) \in \mathcal{F}^\uparrow(u)$, and therefore will reject. □

6 Local hierarchies with bounded certificates

In this paper, the certificates given to the nodes are oblivious to the identities given to these nodes. Giving the ability to potentially assign different certificates for different identity assignments results in a stronger model, introduced in [9] and extended in [8] under the terminology of *locally checkable proofs*, abbreviated as LCP. We can then define $\Sigma_0^{\mathsf{lcp}} = \Pi_0^{\mathsf{lcp}} = \mathsf{LD}$, and $\mathsf{LCP} = \Sigma_1^{\mathsf{lcp}}$, i.e., the equivalent of $\mathsf{NLD} = \Sigma_1^{\mathsf{local}}$ but where the certificates can depend on the identity assignment to the nodes. We have $\Sigma_1^{\mathsf{local}} \subset \Sigma_1^{\mathsf{lcp}}$ because it is known that $\mathsf{LCP} = \mathsf{All}$ [8, 9], while NLD contains only languages that are closed under lift. In other words, while the local hierarchy collapses at the second level Π_2^{local} if certificates are oblivious to the identities given to the nodes, the local hierarchy collapses at the first level Σ_1^{lcp} when the certificates can be function of the identities given to the nodes. However, in both cases, such collapses hold under the condition of using very large certificates, of size $\Omega(\mathrm{poly}(n))$ bits. Therefore, the local hierarchy $(\Sigma_k^{\mathsf{lcp}}, \Pi_k^{\mathsf{lcp}})_{k\geq 0}$ was reconsidered in [3] under the constraint of having certificates with logarithmic size, motivated by applications to the so-called CONGEST model of distributed network computing, yielding the hierarchy $(\Sigma_k^{\mathsf{log\text{-}lcp}}, \Pi_k^{\mathsf{log\text{-}lcp}})_{k\geq 0}$. In this section, we study the same hierarchy, but assuming that the $O(\log n)$-bit certificates given to the nodes cannot depend on the identities given to the nodes, but solely on the given labeled graph. That is, we study the hierarchy

$$(\Sigma_k^{\mathsf{log\text{-}local}}, \Pi_k^{\mathsf{log\text{-}local}})_{k\geq 0},$$

which is defined as $(\Sigma_k^{\mathsf{local}}, \Pi_k^{\mathsf{local}})_{k\geq 0}$ but where certificates are bounded to be on $O(\log n)$ bits, where the constant hidden in the big-O notation depends only on the considered language.

We show that the two hierarchies $(\Sigma_k^{\mathsf{log\text{-}local}}, \Pi_k^{\mathsf{log\text{-}local}})_{k\geq 0}$ and $(\Sigma_k^{\mathsf{local}}, \Pi_k^{\mathsf{local}})_{k\geq 0}$ are actually identical beyond the first level. That is, there are no advantages of using certificates depending on the node identities at level two or more. This is established by combining a couple of results. First, we have the following:

Lemma 3 *For every $k \geq 1$, $\Pi_{2k}^{\mathsf{log\text{-}local}} = \Pi_{2k}^{\mathsf{log\text{-}lcp}}$.*

Proof. By definition, $\Pi_{2k}^{\mathsf{log\text{-}local}} \subseteq \Pi_{2k}^{\mathsf{log\text{-}lcp}}$, so we just need to establish the reverse inclusion. The idea is to add $O(\log n)$ bits to the first universal certificate, and to interpret these bits as *names* assigned to nodes. By doing so, one allows certificates to refer to these names.

Let $k \geq 1$, and let $L \in \Pi_{2k}^{\mathsf{log\text{-}lcp}}$. We show that $L \in \Pi_{2k}^{\mathsf{log\text{-}local}}$. For this purpose, let $\mathcal{A}$ be an algorithm deciding L in $\Pi_{2k}^{\mathsf{log\text{-}lcp}}$, that is, $\mathcal{A}$ satisfies the following:

$$\begin{array}{lcl}(G,x) \in L & \Rightarrow & \forall \mathrm{id}\ \forall c_1\ \exists c_2\ \ldots\ \forall c_{2k-1}\ \exists c_{2k}\ \forall v \in V:\ \mathcal{A}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{accept};\\ (G,x) \notin L & \Rightarrow & \forall \mathrm{id}\ \exists c_1\ \forall c_2\ \ldots\ \exists c_{2k-1}\ \forall c_{2k}\ \exists v \in V:\ \mathcal{A}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{reject}.\end{array}$$

To show that $L \in \Pi_{2k}^{\mathsf{log\text{-}local}}$, we design an algorithm $\mathcal{B}$ satisfying the following:

$$\begin{array}{lcl}(G,x) \in L & \Rightarrow & \forall c_1\ \exists c_2\ \ldots\ \forall c_{2k-1}\ \exists c_{2k}\ \forall \mathrm{id}\ \forall v \in V:\ \mathcal{B}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{accept};\\ (G,x) \notin L & \Rightarrow & \exists c_1\ \forall c_2\ \ldots\ \exists c_{2k-1}\ \forall c_{2k}\ \forall \mathrm{id}\ \exists v \in V:\ \mathcal{B}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{reject}.\end{array}$$

To design an algorithm $\mathcal{B}$ deciding L in $\Pi_{2k}^{\mathsf{log\text{-}local}}$, the first part of the certificate $c_1(v)$ is interpreted, at each node v, as a name in some polynomial range, and, for every $i = 1, \ldots, k$, we set the certificate function c_{2i} as it would be set in $\Pi_{2k}^{\mathsf{log\text{-}lcp}}$ if the nodes were given identities equal to the names

provided by c_1, and ignoring the true identities. More specifically, the first $\min\{\lceil \log_2 n \rceil, |c_1(v)|\}$ bits of $c_1(v)$ are interpreted by $\mathcal{B}$ as the name of node v. (For the sake of simplicity, we assume here that n is known to every node, as if this was not the case, then we could simply add extra 1s at the beginning of $c_1(v)$, followed by a 0, that can be interpreted as the unary representation of $\lceil \log_2 n \rceil$). The remaining $\min\{\lceil \log_2 n \rceil, |c_1(v)|\} - |c_1(v)|$ bits of $c_1(v)$ form a binary string $c'_1(v)$. Hence, c_1 is viewed by $\mathcal{B}$ as

$$c_1(v) = (\text{name}(v), c'_1(v))$$

at every node v. For an instance (G, x) in L, the existential certificate $c_2(v)$ is then forged for $\mathcal{B}$ as follows:

$$c_2(v) = \begin{cases} (0, c'_2(v)) & \text{if there exist two nodes with identical names, where } c'_2 \\ & \text{is a distributed proof of existence for these two nodes;} \\ (1, c'_2(v)) & \text{otherwise, where } c'_2 \text{ is the certificate function for } \mathcal{A} \text{ given for } (G, x) \\ & \text{with certificate function } c'_1, \text{ and the names assigned to nodes by } c_1. \end{cases}$$

The distributed proof of existence for two nodes with identical names may just be composed of two spanning trees rooted at these two nodes, as in the proof of Proposition 4. For every $i = 2, \ldots, k$, we set the existential certificate for $\mathcal{B}$ as

$$c_{2i}(v) = \begin{cases} (0, \perp) & \text{if there exist two nodes with identical names,} \\ & \text{where } \perp \text{ represents the empty string;} \\ (1, c'_{2i}(v)) & \text{otherwise, where } c'_{2i} \text{ is the certificate function for } \mathcal{A} \text{ given for } (G, x) \text{ with} \\ & \text{certificate functions } c'_1, \ldots, c'_{2i-1}, \text{ and the names assigned to nodes by } c_1. \end{cases}$$

Algorithm $\mathcal{B}$ proceeds as $\mathcal{A}$ but with the following modifications. First, every node checks that all its certificates with even indices share the same first bit, and that this bit is the same as the one of the certificates with even indices of all its neighbors. If this is not the case at a node v, then v rejects. A node v that passes this test carries on as follows:

- If the first bit of the certificates with even indices is 0, then v checks that c'_2 correctly proves the existence of two nodes with same name (as it is done in the proof of Proposition 4).
- If the first bit of the certificates with even indices is 1, then v applies $\mathcal{A}$ but using names instead of identities, and using certificates $c'_1, c'_2, c_3, c'_4, c_5, c'_6, \ldots, c_{2k-1}, c'_{2k}$.

We now prove the correctness of $\mathcal{B}$. Let us first consider an instance $(G, x) \in L$. If c_1 assigns the same name to two different nodes, then this will be detected by $\mathcal{B}$ using c'_2, leading all nodes to accept. If c_1 assigns distinct names to the nodes, then $\mathcal{B}$ also accepts since $\mathcal{A}$ accepts (G, x) with certificates $c'_1, c'_2, c_3, c'_4, c_5, c'_6, \ldots, c_{2k-1}, c'_{2k}$ as long as the names assigned to the nodes by c_1 are interpreted as their identities.

In case of an instance $(G, x) \notin L$, we have that

$$\forall \text{id} \ \ \exists c_1 \ \forall c_2 \ \ldots \ \exists c_{2k-1} \ \forall c_{2k} \ \ \exists v \in V : \ \mathcal{A}_{G,x,c_1,\ldots,c_{2k},\text{id}}(v) = \text{reject}.$$

Let $\hat{c}_1$ be the certificate function which assigns certificate $\hat{c}_1(v) = (f(v), c_1(v))$ to every node v, where f provides each node v with a distinct $\lceil \log_2 n \rceil$-bit name. Then, for $i = 2, \ldots, k$, let $\hat{c}_{2i-1} = (1, c_{2i-1})$. By construction, we get that

$$\forall \text{id} \ \ \exists v \in V : \ \mathcal{B}_{G,x,\hat{c}_1,c_2,\hat{c}_3,c_4,\ldots,\hat{c}_{2k-1},c_{2k},\text{id}}(v) = \text{reject},$$

as desired. □

Lemma 4 *For every $k \geq 1$, $\Sigma_{2k+1}^{\mathsf{log\text{-}local}} = \Sigma_{2k+1}^{\mathsf{log\text{-}lcp}}$.*

Proof. Since $\Sigma_{2k+1}^{\mathsf{log\text{-}local}} \subseteq \Sigma_{2k+1}^{\mathsf{log\text{-}lcp}}$, we just need to establish the reverse inclusion. The idea of this proof is the same as for $\Pi_{2k}^{\mathsf{log\text{-}local}}$ versus $\Pi_{2k}^{\mathsf{log\text{-}lcp}}$ in the proof of Lemma 3. Let $L \in \Sigma_{2k}^{\mathsf{log\text{-}lcp}}$. We show that $L \in \Sigma_{2k}^{\mathsf{log\text{-}local}}$. Let $\mathcal{A}$ be an algorithm deciding L in $\Sigma_{2k}^{\mathsf{log\text{-}lcp}}$, i.e., satisfying the following:

$$\begin{array}{rcl}(G,x) \in L & \Rightarrow & \forall \mathrm{id}\ \exists c_1\ \forall c_2\ \ldots\ \exists c_{2k-1}\ \forall c_{2k}\ \ \forall v \in V:\ \mathcal{A}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{accept};\\ (G,x) \notin L & \Rightarrow & \forall \mathrm{id}\ \forall c_1\ \exists c_2\ \ldots\ \forall c_{2k-1}\ \exists c_{2k}\ \ \exists v \in V:\ \mathcal{A}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{reject}.\end{array}$$

We design an algorithm $\mathcal{B}$ deciding L in $\Sigma_{2k}^{\mathsf{log\text{-}local}}$, i.e., satisfying the following:

$$\begin{array}{rcl}(G,x) \in L & \Rightarrow & \exists c_1\ \forall c_2\ \ldots\ \exists c_{2k-1}\ \forall c_{2k}\ \ \forall \mathrm{id}\ \ \forall v \in V:\ \mathcal{B}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{accept};\\ (G,x) \notin L & \Rightarrow & \forall c_1\ \exists c_2\ \ldots\ \forall c_{2k-1}\ \exists c_{2k}\ \ \forall \mathrm{id}\ \ \exists v \in V:\ \mathcal{B}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{reject}.\end{array}$$

For this purpose, let (G,x) in L, and let $f : V \to \{0,\ldots,n-1\}$ be a one-to-one function assigning names to nodes. Assuming that $\mathrm{id}(v) = f(v)$ for every node $v \in V$, let $c_1, c_2, \ldots, c_{2k}$ be a sequence of certificate function leading $\mathcal{A}$ to accept (G,x). We set the first existential certificate function $\hat{c}_1$ for $\mathcal{B}$ as the function which assigns certificate

$$\hat{c}_1(v) = (f(v), c_1(v))$$

to every node v. The second certificate $c_2(v)$ is interpreted by $\mathcal{B}$ as a pair $(b, c_2'(v))$ where

$$\begin{array}{rcl}c_2(v) = (0, c_2'(v)) & \Rightarrow & \text{there exist two nodes with identical names, and } c_2' \\ & & \text{is a distributed proof of existence for these two nodes;}\\ c_2(v) = (1, c_2'(v)) & \Rightarrow & \text{all nodes have distinct names.}\end{array}$$

Now, the second existential certificate $\hat{c}_3(v)$ is set for $\mathcal{B}$ as follows:

$$\hat{c}_3(v) = \begin{cases} (0, c_3'(v)) & \text{if } c_2(v) = (0, c_2'(v)) \text{ but all names are different,} \\ & \text{where } c_3' \text{ is a distributed proof that } c_2' \text{ is erroneous;} \\ (1, c_3(v)) & \text{otherwise.}\end{cases}$$

The distributed proof c_3' that c_2' is erroneous consists of a spanning tree pointing at the error, as it is done in the proof of Proposition 4, using $O(\log n)$-bit certificates. For every $i = 3, \ldots, k$, we set the remaining existential certificates for $\mathcal{B}$ (if any) as

$$\hat{c}_{2i-1}(v) = \begin{cases} (0, \perp) & \text{if there exist two nodes with identical names,} \\ & \text{where } \perp \text{ represents the empty string;} \\ (1, c_{2i-1}(v)) & \text{otherwise.}\end{cases}$$

Algorithm $\mathcal{B}$ proceeds as $\mathcal{A}$ but with the following modifications. First, every node checks that all its certificates $\hat{c}_i$ with odd indices $i \geq 3$ share the same first bit, and that this bit is the same as the one of the certificates with odd indices $i \geq 3$ of all its neighbors. If this is not the case at a node v, then v rejects. A node v that passes this test carries on as follows:

- If the first bit of the certificates with odd indices $i \geq 3$ is 0, then v checks that c_3' correctly proves that there are some inconsistencies in the distributed proof c_2' (as it is done in the proof of Proposition 4).

- If the first bit of the certificates with odd indices $i \geq 3$ is 1, then v applies $\mathcal{A}$ but using names instead of identities, and using certificates $c_1, \ldots, c_{2k}$.

We now prove the correctness of $\mathcal{B}$. Let us first consider an instance $(G, x) \in L$. If c_2 claims for the existence of different names with an erroneous proof, then this will be detected by $\mathcal{B}$ using c'_3, leading all nodes to accept. If c_2 agrees with the fact that all names are different, then $\mathcal{B}$ also accepts since $\mathcal{A}$ accepts (G, x) with certificates $c_1, c_2, \ldots, c_{2k}$ as long as the names assigned to the nodes by $\hat{c}_1$ are interpreted as their identities.

In case of an instance $(G, x) \notin L$, we have that

$$\forall \mathrm{id} \ \ \forall c_1 \ \exists c_2 \ \ldots \ \forall c_{2k-1} \ \exists c_{2k} \ \ \exists v \in V : \ \mathcal{A}_{G,x,c_1,\ldots,c_{2k},\mathrm{id}}(v) = \mathrm{reject}.$$

For certificate functions c_1 which are not assigning distinct names to the nodes, c_2 is set as $c_2 = (0, c'_2)$ where c'_2 proves that there are nodes with identical names. If c'_3 aims at proving that the second certificates are incorrect, then some node will detect some inconsistencies, and reject, as desired. Otherwise, the nodes will apply $\mathcal{A}$, leading them to reject. For certificate functions c_1 which are assigning distinct names to the nodes, the second certificate is set as $\hat{c}_2 = (1, c_2)$. If the third certificate function is inconsistent with this setting, then at least one node will reject. Finally, if c_3 is consistent with the setting of the second certificate, then some node will reject, since $\mathcal{A}$ rejects. This completes the proof. □

It was proved in [3] that, for every $k \geq 1$, we have $\Sigma_{2k}^{\mathsf{log\text{-}lcp}} = \Sigma_{2k-1}^{\mathsf{log\text{-}lcp}}$, and $\Pi_{2k+1}^{\mathsf{log\text{-}lcp}} = \Pi_{2k}^{\mathsf{log\text{-}lcp}}$, that is, the last universal quantifier plays no role (whenever preceded by an existential quantifier). Thanks to this result, and to the previous two lemmas, we can show the following.

Lemma 5 *For every $k \geq 1$, we have $\Sigma_{2k+2}^{\mathsf{log\text{-}local}} = \Sigma_{2k+1}^{\mathsf{log\text{-}local}}$, and $\Pi_{2k+1}^{\mathsf{log\text{-}local}} = \Pi_{2k}^{\mathsf{log\text{-}local}}$.*

Proof. Let $k \geq 1$. We have

$$\Pi_{2k+1}^{\mathsf{log\text{-}local}} \supseteq \Pi_{2k}^{\mathsf{log\text{-}local}} = \Pi_{2k}^{\mathsf{log\text{-}lcp}} = \Pi_{2k+1}^{\mathsf{log\text{-}lcp}} \supseteq \Pi_{2k+1}^{\mathsf{log\text{-}local}}$$

where the first equality follows from Lemma 3, and the second equality follows from [3]. Similarly, we have

$$\Sigma_{2k+2}^{\mathsf{log\text{-}local}} \supseteq \Sigma_{2k+1}^{\mathsf{log\text{-}local}} = \Sigma_{2k+1}^{\mathsf{log\text{-}lcp}} = \Sigma_{2k+2}^{\mathsf{log\text{-}lcp}} \supseteq \Sigma_{2k+2}^{\mathsf{log\text{-}local}}$$

where the first equality follows from Lemma 4, and the second equality follows from [3]. □

Combining all the results in the previous lemmas, we eventually get that identities do not help beyond level 1, hence establishing Theorem 3 .

Proposition 13 *For every $k \geq 3$, we have $\Sigma_k^{\mathsf{log\text{-}local}} = \Sigma_k^{\mathsf{log\text{-}lcp}}$. Similarly, for every $k \geq 2$, we have $\Pi_k^{\mathsf{log\text{-}local}} = \Pi_k^{\mathsf{log\text{-}lcp}}$.*

Regarding the first levels of the hierarchy, we have that $\Sigma_1^{\mathsf{log\text{-}local}} \subseteq \Sigma_1^{\mathsf{log\text{-}lcp}}$ by definition. In fact, the inclusion is strict. Indeed, there are languages in $\Sigma_1^{\mathsf{log\text{-}lcp}}$ that are not in $\Sigma_1^{\mathsf{log\text{-}local}}$. This is for instance the case of the language spanning tree, which is not in $\Sigma_1^{\mathsf{log\text{-}local}}$ because it is not closed under lift, while it is in $\Sigma_1^{\mathsf{log\text{-}lcp}}$ (see, e.g., [9]).

The status of $\Pi_1^{\mathsf{log\text{-}local}}$ versus $\Pi_1^{\mathsf{log\text{-}lcp}}$ is not clear. We have $\mathsf{LD} \subset \Pi_1^{\mathsf{log\text{-}local}} \subset \Sigma_1^{\mathsf{log\text{-}local}}$ with strict inclusions because Propositions 6 and 7 can be adapted to apply for certificates of logarithmic size

(in particular, note that ALTS $\in \Sigma_1^{\mathsf{log\text{-}local}} \setminus \Pi_1^{\mathsf{log\text{-}local}}$). Nevertheless, it is not clear whether or not there are languages in $\Pi_1^{\mathsf{log\text{-}lcp}}$ but not in $\Pi_1^{\mathsf{log\text{-}local}}$.

Also, the status of $\Sigma_2^{\mathsf{log\text{-}local}}$ versus $\Sigma_1^{\mathsf{log\text{-}local}}$ is not clear, neither is clear the status of $\Sigma_2^{\mathsf{log\text{-}local}}$ versus $\Sigma_2^{\mathsf{log\text{-}lcp}} = \Sigma_1^{\mathsf{log\text{-}lcp}}$. Using the same arguments as in Proposition 5, we can state that both $\Sigma_1^{\mathsf{log\text{-}local}}$ and $\Sigma_2^{\mathsf{log\text{-}local}}$ are closed under lift. However, this fact alone is not sufficient to conclude that $\Sigma_2^{\mathsf{log\text{-}local}} = \Sigma_1^{\mathsf{log\text{-}local}}$ as the latter class does not necessarily include all languages that are closed under lift. We know that $\Sigma_1^{\mathsf{log\text{-}local}} \subset \Sigma_1^{\mathsf{log\text{-}lcp}} = \Sigma_2^{\mathsf{log\text{-}lcp}}$ (because spanning tree is not closed under lift, while it belongs to $\Sigma_1^{\mathsf{log\text{-}lcp}}$) but it is not clear where $\Sigma_2^{\mathsf{log\text{-}local}}$ stands between $\Sigma_1^{\mathsf{log\text{-}local}}$ and $\Sigma_1^{\mathsf{log\text{-}lcp}} = \Sigma_2^{\mathsf{log\text{-}lcp}}$.

7 Conclusion

Our investigation raises several intriguing questions. In particular, identifying a distributed language in $\Pi_1^{\mathsf{local}} \setminus \mathsf{LD}$ was a difficult task. We succeeded to find one such language, but we were unable to identify a Π_1^{local}-complete problem, if any. In fact, completeness results are very sensitive to the type of local reductions that are used. We have identified label-preserving local reduction as an appropriate notion. It would be interesting to determine whether NLD-complete and Π_2^{local}-complete languages exist for *identity-oblivious* reductions. This latter type of reductions is indeed the most natural one in a context in which nodes may not want to leak information about their identities. It is easy to see that the class co-LD has a complete language for identity-oblivious reductions, namely, OR is co-LD-complete for identity-oblivious reductions. However, we do not know whether this can be achieved for NLD or Π_2^{local}.

This paper is aiming at providing a proof of concept for the notion of interactive local verification: Π_2^{local} can be viewed as the interaction between two players, with conflicting objectives, one is aiming at proving the instance, while the other is aiming at disproving it. As a consequence, for this first attempt, we voluntarily ignored important parameters such as the size of the certificates, and the individual computation time, and we focussed only on the locality issue. The impact of limiting the certificate size was recently investigated in [3]. Regarding the individual computation time, our completeness results involve local reductions that are very much time consuming at each node. Insisting on local reductions involving polynomial-time computation at each node is crucial for practical purpose. At this point, we do not know whether non-trivial hardness results can be established under polynomial-time local reductions. Proving or disproving the existence of such hardness results is left as an open problem.

Acknowledgement: The authors are thankful to Laurent Feuilloley for fruitful discussions about the topic of the paper.

References

[1] L. Feuilloley, P. Fraigniaud: Randomized Local Network Computing. In proc. 27th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp340-349, 2015

[2] L. Feuilloley, Pierre Fraigniaud: Survey of Distributed Decision. Bulletin of the EATCS 119 (2016)

[3] L. Feuilloley, P. Fraigniaud, J. Hirvonen: A Hierarchy of Local Decision. In proc. 43rd International Colloquium on Automata, Languages and Programming (ICALP), pp118:1-118:15, 2016.

[4] P. Floréen, J. Kaasinen, P. Kaski, J. Suomela: An optimal local approximation algorithm for max-min linear programs. In proc. 21st ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp260-269, 2009.

[5] P. Fraigniaud, M. Göös, A. Korman, J. Suomela: What can be decided locally without identifiers? 32nd ACM Symp. on Principles of Dist. Comput. (PODC), pp157-165, 2013.

[6] P. Fraigniaud, M. Halldórsson, A. Korman. On the Impact of Identifiers on Local Decision. In proc. 16th Int. Conference on Principles of Distributed Systems (OPODIS). Springer, LNCS 7702, pp224-238, 2012.

[7] P. Fraigniaud, A. Korman, D. Peleg. Towards a complexity theory for local distributed computing. J. ACM 60(5): 35 (2013) (Preliminary version in FOCS 2011).

[8] M. Göös, J. Suomela: Locally checkable proofs. In proc. 30th ACM Symposium on Principles of Distributed Computing (PODC), pp159-168, 2011.

[9] A. Korman, S. Kutten, D. Peleg (2010). Proof labeling schemes. Distributed Computing 22(4): 215-233.

[10] F. Kuhn, T. Moscibroda, R. Wattenhofer: What cannot be computed locally! In proc. 23rd ACM Symp. on Principles of Distributed Computing (PODC), pp300-309, 2004.

[11] C. Lenzen, Y. Anne Oswald, R. Wattenhofer: What can be approximated locally? In proc. 20th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA), pp46-54, 2008.

[12] C. Lenzen, R. Wattenhofer: Leveraging Linial's Locality Limit. In proc. 22nd Int. Symp. on Distributed Computing (DISC), pp394-407, 2008.

[13] N. Linial. Locality in Distributed Graph Algorithms. SIAM J. Comp. 21(1): 193-201 (1992)

[14] M. Naor, L. Stockmeyer. What Can be Computed Locally? SIAM J. Comput. 24(6): 1259-1277 (1995)

[15] D. Peleg. Distributed Computing: A Locality-Sensitive Approach. SIAM (2000)

[16] F. Reiter: Distributed Graph Automata. In proc. 30th ACM/IEEE Symposium on Logic in Computer Science (LICS), pp192-201, 2015.

[17] J. Suomela: Survey of local algorithms. ACM Comput. Surv. 45(2): 24 (2013)

APPENDIX

A Verifying COVER with quasi linear certificates

Proposition 14 Let U be the ground set of COVER. Then COVER has a local decision algorithm for NLD, using certificates of size $O(n(\log n + \log |U|))$ bits.

Proof. Given $(G, x) \in$ COVER, where G is an n-node graph, the prover assigns the following certificates to the nodes. For any $u \in V(G)$, we have $c(u) = (d_0, (d_1, e_1), (d_2, e_2), \ldots, (d_n, e_n))$, where, for every $i \in \{0, \ldots, n\}$, d_i is a non-negative integer, and, for every $i \in \{1, \ldots, n\}$, $e_i \in U$. This certificate is on $O(n(\log n + \log |U|))$ bits. The d_i's measure distances: d_0 is the distance from u to the node v which has a set $S_i(v)$ covering e, and, for every $i \in \{1, \ldots, n\}$, d_i is the distance from u to a node u' with $e(u') = e_i$.

The verifier acts as follows, in just one communication round. Every node u checks that it has the same number of distance entries in its certificate as all its neighbors, and that the ith elements coincide between neighbors, for every $i = 1, \ldots, n$. Next, it checks that one and only one of its distances d_i with $i \in \{1, \ldots, n\}$ is null, and that $e(u) = e_i$. Next, if $d_0 = 0$, it checks that it has a set $S_j(u) = \{e_i \mid i = 1, \ldots, n\}$. Finally, it checks that the distances are consistent, that is, for every i such that $d_i \neq 0$, it checks that it has at least one neighbor whose ith distance is smaller than d_i. If all tests are passed, then u accepts, otherwise it rejects.

By construction, if $(G, x) \in$ COVER then all nodes accept. Conversely, let us assume that all nodes accept. Since the distances are decreasing, for each element e_i there must exist at least one node u such that $x(u) = e_i$. Conversely, every node has its element appearing in the certificate (because it must have one distance equal to 0). Finally, since the distance d_0 is decreasing, there must exist at least one node u that has a set $S_j(u) = \{e_i \mid i = 1, \ldots, n\}$. This implies that $(G, x) \in$ COVER. □