What Can Be Verified Locally?∗

Alkida Balliu¹, Gianlorenzo D’Angelo², Pierre Fraigniaud†³, and Dennis Olivetti⁴

1 Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France; and Gran Sasso Science Institute (GSSI), L’Aquila, Italy
2 Gran Sasso Science Institute (GSSI), L’Aquila, Italy
3 Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France
4 Institut de Recherche en Informatique Fondamentale (IRIF), CNRS and University Paris Diderot, Paris, France; and Gran Sasso Science Institute (GSSI), L’Aquila, Italy

Abstract

We are considering distributed network computing, in which computing entities are connected by a network modeled as a connected graph. These entities are located at the nodes of the graph, and they exchange information by message-passing along its edges. In this context, we are adopting the classical framework for local distributed decision, in which nodes must collectively decide whether their network configuration satisfies some given boolean predicate, by having each node interacting with the nodes in its vicinity only. A network configuration is accepted if and only if every node individually accepts. It is folklore that not every Turing-decidable network property (e.g., whether the network is planar) can be decided locally whenever the computing entities are Turing machines (TM). On the other hand, it is known that every Turing-decidable network property can be decided locally if nodes are running non-deterministic Turing machines (NTM). However, this holds only if the nodes have the ability to guess the identities of the nodes currently in the network. That is, for different sets of identities assigned to the nodes, the correct guesses of the nodes might be different. If one asks the nodes to use the same guess in the same network configuration even with different identity assignments, i.e., to perform identity-oblivious guesses, then it is known that not every Turing-decidable network property can be decided locally.

In this paper, we show that every Turing-decidable network property can be decided locally if nodes are running alternating Turing machines (ATM), and this holds even if nodes are bounded to perform identity-oblivious guesses. More specifically, we show that, for every network property, there is a local algorithm for ATMs, with at most 2 alternations, that decides that property. 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 level $k$ contains those tasks solvable with ATMs with $k$ 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 properties that are locally decidable with TMs, NTMs, and ATMs. Finally, we establish the existence of completeness results for each of these classes, using novel notions of local reduction.

1998 ACM Subject Classification D.1.3 Concurrent Programming (Distributed programming), F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Distributed Network Computing, Distributed Algorithm, Distributed Decision, Locality

Digital Object Identifier 10.4230/LIPIcs.STACS.2017.8

* The first, third and fourth authors received additional support from the ANR project DISPLEXITY.
† Additional support from the INRIA project GANG.
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. In this paper, we are interested in local distributed decision. We insist on locality, as we want the checking protocols to avoid involving long-distance communications across the network, for they are generally costly and potentially unreliable. More specifically, we consider the standard LOCAL model of computation in networks [14]. 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 it limits the amount of computation that is performed by a node during a round. Indeed, the model places emphasis on the number of rounds before every node can output, as a measure of locality. (Note however that, up to some exceptions, our positive results involve messages of logarithmic size, and polynomial-time computation). A local algorithm is a distributed algorithm \( A \) satisfying that there exists a constant \( t \geq 0 \) such that \( A \) terminates in at most \( t \) rounds in all networks, for all inputs. The parameter \( t \) is called the radius of \( A \). In other words, in every network \( G \), and for all inputs to the nodes of \( G \), every node executing \( A \) 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 \( B_G(u,t) = \{ v \in V(G) : \text{dist}(u,v) \leq t \} \).

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 [6], 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. \( \mathcal{L} \). Note that 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 (this is because one may want to study the same language under different computational models, including ones that assume anonymous nodes). The class \( \mathcal{L}D \) is the set of all distributed languages that are locally decidable. That is, \( \mathcal{L}D \) is the class of all distributed languages \( \mathcal{L} \) for which there exists a local algorithm \( A \) satisfying that, for every configuration \((G,x)\),

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

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

\[ (G,x) \in \mathcal{L} \Rightarrow \forall id \in \text{ID}(G), \forall u \in V(G), \ A_{G,x,id}(u) = \text{accept} \]

\[ (G,x) \notin \mathcal{L} \Rightarrow \forall id \in \text{ID}(G), \exists u \in V(G), \ A_{G,x,id}(u) = \text{reject} \]

where \( A_{G,x,id}(u) \) is the output of Algorithm \( A \) running on the instance \((G,x)\) with identity-assignment \( id \), at node \( u \). For instance, the language \( \text{PROP-COL} \), composed of all (connected) properly colored graphs, is in \( \mathcal{L}D \). Similarly, the class \( \mathcal{LCL} \) of “locally checkable labelings”,

Note that the membership of a configuration in a distributed language is independent of the input \( x \) to the nodes of \( G \), for all inputs to the nodes of \( G \). The parameter \( t \) is called the radius of \( A \). In other words, in every network \( G \), and for all inputs to the nodes of \( G \), every node executing \( A \) 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 \( B_G(u,t) = \{ v \in V(G) : \text{dist}(u,v) \leq t \} \).
defined in [13], satisfies LCL ⊆ 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 \( L \) for which there exists a local algorithm \( \mathcal{A} \) verifying \( L \), i.e., satisfying that, for every configuration \((G, x)\),

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

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

\[(G, x) \in L \implies \exists c \in C(G), \forall u \in ID(G), \forall u \in V(G), \mathcal{A}_{G,x,c,\text{id}}(u) = \text{accepts}\]

\[(G, x) \notin L \implies \forall c \in C(G), \forall u \in ID(G), \exists u \in V(G), \mathcal{A}_{G,x,c,\text{id}}(u) = \text{rejects}\]

where \( C(G) \) is the class of all functions \( c : V(G) \to \{0, 1\}^* \), assigning 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. In the following, for the sake of simplifying the notations, we shall omit specifying the domain sets \( C(G) \) and \( ID(G) \) 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 \( r \). 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 \( c(v) < c(u) \). In [5], NLD was proved to be exactly the class of distributed languages that are closed under lift.

Finally, [6] defined the randomized versions BPLD\(_{p,q}\) and 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 \( p \) of acceptance for legal instances, and its probability \( q \) of rejection for illegal instances. By defining BPNLD = \( \cup_{p^2 + q \geq 1} \) BPNLD\(_{p,q}\), the landscape of local decision was pictured as follows:

\[\text{LD } \subset \text{NLD } \subset \text{BPNLD } = \text{All}\]

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 [15], and to the CONGEST model in [2], we observe that the class LD and NLD are in fact the basic levels of a “local hierarchy” defined as follows. Let \( \Sigma_0^{\text{loc}} = \Pi_0^{\text{loc}} = \text{LD} \), and, for \( k \geq 1 \), let \( \Sigma_k^{\text{loc}} \) be the class of all distributed languages \( L \) for which there exists a local algorithm \( \mathcal{A} \) satisfying that, for every configuration \((G, x)\),

\[(G, x) \in L \iff \exists c_1, \forall c_2, \ldots, Q c_k, \mathcal{A} \text{ accepts } (G, x) \text{ with certificates } c_1, c_2, \ldots, c_k.\]
What Can Be Verified Locally?

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

where the quantifiers alternate, and $Q$ is the universal quantifier if $k$ is even, and the existential one if $k$ is odd. The class $\Pi_{loc}^k$ is defined similarly, by starting with a universal quantifier, instead of an existential one. A local algorithm $A$ insuring membership to a class $C \in \{\Sigma_{loc}^k, k \geq 0\} \cup \{\Pi_{loc}^k, k \geq 0\}$ is called a $C$-algorithm. Hence, $\text{NLD} = \Sigma_{loc}^1$, and, for instance, $\Pi_{loc}^2$ is the class of all distributed languages $L$ for which there exists a $\Pi_{loc}^2$-algorithm, that is, a local algorithm $A$ satisfying the following: for every configuration $(G,x)$,

\[
(G,x) \in L \Rightarrow \forall c_1, \exists c_2, \forall id, \forall u \in V(G), A_{G,x,c_1,c_2,id}(u) = \text{accept};
\]

\[
(G,x) \notin L \Rightarrow \exists c_1, \forall c_2, \forall id, \exists u \in V(G), A_{G,x,c_1,c_2,id}(u) = \text{reject}.
\] (1)

Our main results are the following.

\textbf{Theorem 1.} $\text{LD} \subset \Pi_{loc}^1 \subset \text{NLD} = \Sigma_{loc}^2 \subset \Pi_{loc}^2 = \text{All}$, where all inclusions are strict.

That is, $\Pi_{loc}^1 \supset \Pi_{loc}^0$, while $\Sigma_{loc}^2 = \Sigma_{loc}^1$, and the whole local hierarchy collapses to the second level, at $\Pi_{loc}^2$. 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 [6], 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.

\textbf{Theorem 2.} $\text{NLD}$ and $\Pi_{loc}^2$ have complete distributed languages under local label-preserving reductions.

Finally, Figure 1 summarizes all our separation results.

1.3 Related Work

Several form 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, [15] 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 $\Sigma_k^{\text{loc}}$, $k \geq 0$, 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) [8], extended to locally checkable proofs (LCP) [7], give the ability to certify predicates using certificates that can take benefits 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 $\Sigma_1^{\text{loc}} = \text{All}$. However, this holds only if the certificates can be as large as $\Omega(n^2)$ bits. In [2], the class LogLCP [7], 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 $\Sigma_2^{\text{loc}}$ of that hierarchy, while there are languages outside the hierarchy.

In [6], the authors introduced the model investigated in this paper. In particular, they defined and characterized the class NLD, which is nothing else than $\Sigma_1^{\text{loc}}$, 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 NLD $\neq$ All, randomization helps a lot, as the randomized version BPNLD of NLD satisfies BPNLD = All. It is also proved that, with the oracle $\#\text{nodes}$ providing each node with the number of nodes in the network, we get $\text{NLD}^{\text{nodes}} = \text{All}$. Interestingly, it was proved [5] 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 $\text{NLD} = \text{LD}$, where the “O” in NLD 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., $\text{LD} \subseteq \text{LDO}$ (see [4]).

Finally, it is worth mentioning that the ability to decide a distributed language locally has impact on the ability to design construction algorithms [12] 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 [13]. 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 [6]). More generally, the reader is invited to consult [3, 9, 10, 11, 14, 16] for good introductions to local computing, and/or samples of significant results related to local computing.

### 2 All languages are $\Pi_2^{\text{loc}}$ decidable

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

**Proposition 3.** $\Pi_2^{\text{loc}} = \text{All}$.

**Proof.** Let $\mathcal{L}$ be a distributed language. We give an explicit $\Pi_2^{\text{loc}}$-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 $(M(u), \text{data}(u), \text{index}(u))$ where $M(u)$ is an $m \times m$ boolean matrix, $\text{data}(u)$ is a linear...
array with \( m \) entries, and \( \text{index}(u) \in \{1, \ldots, m\} \). Informally, \( c_1(u) \) aims at proving to node \( u \) that it is node labeled \( \text{index}(u) \) in the \( m \)-node graph with adjacency matrix \( M(u) \), and that the whole input data is \( \text{data}(u) \). 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 \( \text{dist}(u, v) \) denote the distance between any two nodes \( u \) and \( v \) in the actual network \( G \), that is, \( \text{dist}(u, v) \) 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 \( u \neq v \) such that \((M(v), \text{data}(v)) \neq (M(v'), \text{data}(v'))\), or there is at least one node \( v \) in which \( c_1(v) \) cannot be read as a triple \((M(v), \text{data}(v), \text{index}(v))\). Then we set one of these nodes as witness \( w \), and we set \( c_2(u) = (0, \text{dist}(u, w)) \) at every node \( u \).

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

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

**Case 2.** \( n > m \), i.e., \( |V(G)| \) is larger than the dimension \( m \) of \( M \), or \( \text{index()} \) is not injective. Then we set the certificate \( c_2(u) = (2, i, d(u, w), d(u, w')) \) 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 \( \text{index()} \) 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 \( \text{index()} \) is injective, but \((G, x)\) is not isomorphic to \((M, \text{data})\). Then we set as witness a node \( w \) whose neighborhood in \((G, x)\) does not fit with what it should be according to \((M, \text{data})\), and we set \( c_2(u) = (4, d(u, w)) \) for every node \( u \).

The local verification algorithm \( A \) 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 \( f(u) = 0 \), then \( u \) checks that at least one of its neighbors has a distance to the witness that is smaller than its own distance. A node with distance 0 to the witness checks that there is indeed an inconsistency with its \( c_1 \) certificate (i.e., its \( c_1 \) certificate cannot be read as a pair matrix-data, or its \( c_1 \) certificate is distinct from the one of its neighbors).
- Every node accepts or rejects accordingly.
- 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
These two classes have in common that the universal quantifier is positioned last. It results which stands for “exactly two selected” be the following language. We set that these two classes seem to be limited, as witnessed by the following two propositions.

In this section, we prove the part of Theorem 1 related to the two classes \( \Pi^1_{loc} \) and \( \Sigma^2_{loc} \). 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.

\[ \text{Lemma 4} \] [5]. NLD is the class of distributed languages closed under lift.

Since \( \text{exts} \) is not closed under lift, it results from Lemma 4 that \( \text{exts} \notin \text{NLD} \).

3 On the impact of the last universal quantifier

In this section, we prove the part of Theorem 1 related to the two classes \( \Pi^1_{loc} \) and \( \Sigma^2_{loc} \).
What Can Be Verified Locally?

Figure 2. An illustration of the distributed language $\text{iter}$.

\textbf{Proposition 5.} $\Sigma^\text{loc}_2 = \text{NLD}$.

To show that $\Pi^\text{loc}_1 \neq \text{NLD}$, we consider the language $\text{alts}$, which stands for “at least two selected”. (Note that $\text{alts}$ is the complement of the language $\text{amos}$ introduced in [6], where $\text{amos}$ stands for “at most one selected”). We set $(G,x) \in \text{alts}$ if $x(u) \in \{\perp, \top\}$ for every node $u \in V(G)$, and $|\{ u \in V(G) : x(u) = \top\}| \geq 2$. To separate $\text{NLD}$ and $\Pi^\text{loc}_1$, we show that $\text{alts} \in \text{NLD} \setminus \Pi^\text{loc}_1$.

\textbf{Proposition 6.} $\Pi^\text{loc}_1 \subset \text{NLD}$ (the inclusion is strict).

While $\Pi^\text{loc}_1$ is in $\text{NLD}$, the universal quantifier adds some power compared to $\text{LD}$. We show that $\text{LD} \neq \Pi^\text{loc}_1$ by exhibiting a language in $\Pi^\text{loc}_1 \setminus \text{LD}$. 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 $\text{LD}$ and $\Pi^\text{loc}_1$ would be identical. For instance, given a $t$-round algorithm $\mathcal{A}$ deciding a language $\mathcal{L}$ in $\Pi^\text{loc}_1$, one could define the following mechanism for deciding the same language in $\text{LD}$. 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, $\text{NLD}$ would still not collapse to $\text{LD}$ 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 $\text{NLD}$ 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 $\Pi^\text{loc}_1$, does not work for $\text{NLD}$. 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, $\Pi^\text{loc}_1 \setminus \text{LD} \neq \emptyset$.

\textbf{Proposition 7.} $\text{LD} \subset \Pi^\text{loc}_1$ where the inclusion is strict.

\textbf{Proof.} We describe the distributed language $\text{iter}$, 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 $f_M : \mathbb{N} \rightarrow \mathbb{N}$ 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 $i$. We define $\text{iter}$ 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 2).

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 2). A configuration in $\text{iter}$ 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 of the path are given the machine $M$ as input, and the pivot $v$ is also given $a$ and $b$ as inputs. The node of the left path (resp., right path) at distance $i$ from $v$ is given...
To show the correctness of \( f \) with respect to the given machine \( M \), we let \( a \) and \( b \) be the two nodes at the extremity of the left path and of the right path, respectively. 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:

\[
(f_{|L|, L} \in \{0, 1\} \text{ and } f_{|R|, R} \in \{0, 1\}) \text{ and } (f_{|L|, L} = 0 \text{ or } f_{|R|, R} = 0).
\]

In fact, for technical reasons, it is also required that both \( |L| \) and \( |R| \) are powers of 2. Indeed, on top of \( L \) and \( R \) are two complete binary trees \( T_L \) and \( T_R \), respectively, with horizontal paths connecting nodes of the same depth in each tree (see Figure 2). The nodes of \( L \) and \( R \) are the leaves of these two trees. Finally, every node \( u \) of the graph receives as input a pair of labels \( (\ell_1, \ell_2) \in \{0, 1, 2\}^2 \). The label \( \ell_1 \) is the distance modulo 3 from \( u \) to the right-most node (resp., left-most node) of the path if \( u \) is an internal node of \( T_L \) (resp., \( T_R \)), and, for nodes in the path \( P \), \( \ell_1 \) is simply the distance modulo 3 from the pivot \( v \). The label \( \ell_2 \) is the height of the node in its tree modulo 3. (The pivot, which belongs to none of the trees, has height 0). A configuration \((G, x) = \text{iter} \) if and only if \((G, x)\) satisfies all the above conditions with respect to the given machine \( M \).

In other words, \( f_M \) is defined so that 1 denotes the rejecting state with any tape content, and any head position, while 0 denotes the accepting state with any tape content, and any head position. All the 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 the head position. If the machine halts, then we discard the tape content as well as the head position, and we simply set \( f_M \) equal to 0 or 1 accordingly. 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 \( \text{iter} \), denoted by \( \text{iter}^- \) where the condition of Eq. (2) is replaced by just: \( f_{|L|, L} \in \{0, 1\} \) and \( f_{|R|, R} \in \{0, 1\} \). Thanks to the labeling \( (\ell_1, \ell_2) \) at each node, which “rigidifies” the structure, we have \( \text{iter}^- \in \text{LD} \) using the same arguments as the ones in [4]. Moreover, \( \text{iter} \in \Pi^p_\infty \). To see why, we describe a local algorithm \( \mathcal{A} \) using certificates. The algorithm first checks whether \((G, x) \in \text{iter}^- \). All nodes, but the pivot \( v \), decide according to this checking. If the pivot rejected \((G, x) \in \text{iter}^- \), 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 \((G, x) \in \text{iter} \). 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 \((G, x) \notin \text{iter} \). If \((G, x) \notin \text{iter}^- \) then at least one node rejects, independently of the certificate. So, we assume that \((G, x) \in \text{iter}^- \setminus \text{iter} \). 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, \( \text{iter} \in \Pi^p_\infty \).

It remains to show that \( \text{iter} \notin \text{LD} \). Let us assume, for the purpose of contradiction, that there exists a \( t \)-round algorithm \( \mathcal{A} \) deciding \( \text{iter} \). Since \( \text{iter} \in \text{LD} \), 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 \(A\) at all nodes in the ball of radius \(t\) around \(v\), with identities between 1 and the size of the ball of radius 2\(t\) around the pivot, a sequential algorithm can determine, given a Turing machine \(M\), and given \(a\) and \(b\), whether there exist \(\ell\) and \(r\) such that \(f_M^{(L)}(a) = f_M^{(R)}(b) = 1\) or not, which is actually Turing undecidable. This contradiction implies that, indeed, \(\text{iter} \notin \text{LD}\). ▶

4 Complement classes

Given a class \(\mathcal{C}\) of distributed languages, the class \(\text{co-}\mathcal{C}\) 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, \(\text{co-}\Pi_1^{\text{loc}}\) is the class of languages \(\mathcal{L}\) for which there exists a local algorithm \(A\) such that, for every configuration \((G, x)\),

\[
\begin{align*}
(G, x) \in \mathcal{L} &\Rightarrow \exists e, \forall id, \exists u \in V(G), A_{G,x,e,id}(u) = \text{accepts}; \\
(G, x) \notin \mathcal{L} &\Rightarrow \forall e, \forall id, \forall u \in V(G), A_{G,x,e,id}(u) = \text{rejects}.
\end{align*}
\]

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 \(\text{AND}\) (resp., in \(\text{OR}\)) if and only if the logical conjunction (resp., disjunction) of the inputs is true. These two languages enable to separate \(\text{LD}\) from its co-class. Indeed, \(\text{OR} \notin \text{LD}\) 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, \(\text{AND} \in \text{LD}\): every node accepts if and only if its input is 1. The class \(\text{LD} \cap \text{co-LD}\) is quite restricted. Nevertheless, it contains distributed languages such as \(\text{Diam}_k\), the class of graphs with diameter at most \(k\), for any fixed \(k\). We have the following separation.

\begin{itemize}
  \item \textbf{Proposition 8.} \(\text{OR} \in \text{co-LD} \setminus \Pi_1^{\text{loc}}, \text{and AND} \in \text{LD} \setminus \text{co-}\Pi_1^{\text{loc}}\).
\end{itemize}

Similarly, the languages \(\text{ALTS}\) and \(\text{AMOS}\) introduced in the proof of Proposition 6 enable to separate \(\text{NLD}\) from its co-class. Indeed, \(\text{ALTS} = \overline{\text{AMOS}}\), \(\text{ALTS}\) is closed under lift, and \(\text{AMOS}\) is not closed under lift. Moreover, consider the language \(\text{EXTS}\) defined at the end of Section 2. Both \(\text{EXTS}\) and \(\text{EXTS}\) are not closed under lift. So, overall, by Lemma 4, we get:

\begin{itemize}
  \item \textbf{Proposition 9.} \(\text{ALTS} \in \text{NLD} \setminus \text{co-NLD}, \text{AMOS} \in \text{co-NLD} \setminus \text{NLD}, \text{and EXTS} \notin \text{NLD} \cup \text{co-NLD}\).
\end{itemize}

More interesting is the position of the \(\Pi_1^{\text{loc}}\) w.r.t. \(\text{NLD}\) and \(\text{co-NLD}\):

\begin{itemize}
  \item \textbf{Proposition 10.} \(\Pi_1^{\text{loc}} \cup \text{co-}\Pi_1^{\text{loc}} \subset \text{NLD} \cap \text{co-NLD}, \text{where the inclusion is strict}\).
\end{itemize}

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 \(e : V(G) \rightarrow U\), and let \(S : V(G) \rightarrow 2^U\). That is, \(e\) assigns an element \(e(u) \in U\) to every node \(u \in V(G)\), and \(S\) assigns a collection of sets \(S(u) = \{S_1(u), \ldots, S_k(u)\}\) to every node \(u \in V(G)\), with \(k_u \geq 1\) and \(S_i : V(G) \rightarrow 2^U\) for every \(i \geq 1\). We say that \(S\) 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 [6], the authors defined the language \(\text{COVER} = \{(G, x) : \forall u \in V(G), x(u) = (S(u), e(u)) \text{ such that } S \text{ covers } e\}\)
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 4, COVER ∈ NLD. \(^1\)

This is in contradiction with the claim in [6] regarding the hardness of COVER. The reason for this contradiction is that the local reduction used in [6] 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 \(\Sigma^\text{loc}_k\) and \(\Pi^\text{loc}_k\), \(k \geq 0\), 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 [6] that a local reduction of \(\mathcal{L} \to \mathcal{L}'\) is a local algorithm \(R\) which maps any configuration \((G, x)\) to a configuration \((G, y)\), where \(y = R(G, x, \text{id})\) may depend on the identity assignment \(\text{id}\), such that: \((G, x) \in \mathcal{L}\) if and only if, for every identity assignment \(\text{id}\) to the nodes of \(G\), \((G, y) \in \mathcal{L}'\) where \(y = R(G, x, \text{id})\). Ideally, we would like \(R\) 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 [6].

**Definition 11.** Let \(\mathcal{C}\) be a class of distributed languages, and let \(\mathcal{L}\) and \(\mathcal{L}'\) be two distributed languages. Let \(A\) be a \(C\)-algorithm deciding \(\mathcal{L}'\), and let \(R\) be a local reduction of \(\mathcal{L}\) to \(\mathcal{L}'\). We say that \((R, A)\) 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 \(A\) for \((G, y)\) where \(y = R(G, x, \text{id})\) are the same for all identity assignments \(\text{id}\) to \(G\).

The following result shows that the notion of reduction in Definition 11 preserves the classes of distributed languages.

**Lemma 12.** 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 \((R, A)\) be a label-preserving local reduction for \((\mathcal{L}, \mathcal{L}')\). Then \(\mathcal{L} \in \mathcal{C}\).

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 \(\mathcal{F} = \bigcup_{u \in V(G)} \mathcal{F}(u)\). In short, we consider the language

\[
\text{Miss} = \{(G, x) : \forall u \in V(G), x(u) = (\mathcal{F}(u), x'(u)) \text{ and } (G, x') \notin \mathcal{F}\}
\]

We show that Miss is among the hardest decision tasks, under local label-preserving reductions. Note that Miss \(\notin\) 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 13.** Miss is \(\Pi^\text{loc}_2\)-complete under local label-preserving reductions.

\(^1\) 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.
What Can Be Verified Locally?

**Proof.** Let $\mathcal{L}$ be a distributed language. We describe a local label-preserving reduction $(R, \mathcal{A})$ for $(\mathcal{L}, \text{MISS})$ with respect to $\Pi^2_{2\infty}$.

In essence, the local algorithm $\mathcal{A}$ for deciding MISS in $\Pi^2_{2\infty}$ is the generic algorithm described in the proof of Proposition 3. 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 $\mathcal{F}(u)$ (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 $(G, x') \in \text{MISS}$, 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 [6]. A node $u$ with identity $\text{id}(u)$ and input $x(u)$ computes its width $\omega(u) = 2^{||\text{id}(u)|| + |x(u)|}$ where $|s|$ denotes the length of a bit-string $s$. Then $u$ generates all configurations $(H, y) \in \mathcal{L}$ 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 $(G, R(G, x)) \notin \text{MISS}$. Conversely, if $(G, x) \notin \mathcal{L}$, then $(G, R(G, x)) \notin \text{MISS}$. Indeed, there exists at least one node $u$ with identity $\text{id}(u) \geq n$, 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 $(G, R(G, x)) \notin \text{MISS}$.

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 MISS$^\uparrow$ is among the hardest decision tasks in NLD.

**Proposition 14.** MISS$^\uparrow$ is NLD-complete (and MISS$^{\uparrow\downarrow}$ is co-NLD-complete) under label-preserving reduction.

**Conclusion**

This paper is aiming at providing a proof of concept for the notion of interactive local verification: $\Pi^2_{2\infty}$ 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 [2].

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