Local Terminations and Distributed Computability in Anonymous Networks *

Jérémie Chalopin[1], Emmanuel Godard[1], and Yves Métivier[2]

[1] Laboratoire d'Informatique Fondamentale de Marseille
CNRS & Aix-Marseille Université
{jeremie.chalopin,emmanuel.godard}@lif.univ-mrs.fr
[2] LaBRI, Université de Bordeaux
metivier@labri.fr

Abstract. We investigate the computability of distributed tasks in reliable anonymous networks with arbitrary knowledge. More precisely, we consider tasks computable with *local termination*, i.e., a node knows when to stop to participate in a distributed algorithm, even though the algorithm is not necessarily terminated elsewhere. We also study *weak local termination*, that is when a node knows its final value but continues to execute the distributed algorithm, usually in order to provide information to other nodes.
We give the first characterization of distributed tasks that can be computed with weak local termination and we present a new characterization of tasks computed with local termination. For both terminations, we also characterize tasks computable by polynomial algorithms.

1 Introduction

We investigate the computability of distributed tasks in reliable anonymous networks with arbitrary knowledge. Impossibility results in anonymous networks have been investigated for a long time [Ang80]. Among the notable results are the ones of Boldi and Vigna [BV99,BV01], following works of Angluin [Ang80] and of Yamashita and Kameda [YK96a,YK96b]. In [BV99], a characterization of what is computable with arbitrary knowledge is presented. In a following paper [BV01], another characterization is presented but the processes have to know a bound on the number of nodes in the network. To quote the introduction of [BV99], "in a sense the whole issue becomes trivial, as one of the main problems – termination – is factored out *a priori*". That's why we focus in this paper not only on the way to solve a distributed task, but also on what is exactly at stake when one talks about termination in a distributed context.

About Terminations of Distributed Algorithms. Contrary to sequential algorithms, what is the termination of a distributed algorithm is not so intuitively obvious. If we take a global perspective, termination occurs when there is not anything left to do in the network: no message is in transit and no process can modify its state. But if we are interested in the local point of view of a

* partially supported by grant No ANR-06-SETI-015-03 awarded by A.N.R.

node executing a distributed algorithm, it is generally not obvious to detect when it can stop waiting for incoming messages. And as usual in the local-global relationship, this is not always possible, or it involves more computation.

Moreover, if we look carefully at what the distributed algorithm is aimed at, we have to begin to distinguish between the termination of the task we want to achieve (the associated computed values) and the termination of our tool, the distributed algorithm. Indeed, a node does not necessarily need to detect that the algorithm has *globally* terminated, but it is interesting if it can detect it has computed its final value (*local termination*). For example, in the case of composition of algorithms, or for local garbage collecting purpose, there is, a priori, no special need to wait that everyone in the network has computed its final value. One can define a hierarchy of termination for distributed tasks:

- *implicit termination*: the algorithm is globally terminated but no node is, or can be aware of this termination;
- *weak local termination* : every node knows when it has its final value, but does not immediately halt in order to transmit information needed by some other nodes;
- *local termination*: every node knows when it has its final value and stops participating in the algorithm;
- *global termination detection*: at least one node knows when every other node has computed its final value.

Related Works. In the seminal work of Angluin [Ang80], the first impossibility results for distributed computability in anonymous networks were established. Using the notion of coverings we also use in this paper, she prove that it is impossible to elect a leader in a wide class of "symmetric" networks. She also shows that it is impossible to have a universal termination detection algorithm for any class of graphs that strictly contains the family of all trees.

Distributed computability on asynchronous anonymous rings have been first investigated by Attiya, Snir and Warmuth [ASW88]. They show that any function can be computed with a quadratic number of messages. Some results have been presented when processes initially have ids (or inputs) but they are not assumed to be unique. In this setting, Flocchini *et al.* [FKK$^+$04] consider Election and Multiset Sorting. Mavronicolas, Michael and Spirakis [MMS06] present efficient algorithms for computing functions on some special classes of rings. In all these works, it is assumed that processes initially know the size of the ring. In [DP04], Dobrev and Pelc consider Leader Election on a ring assuming the processes initially know a lower and an upper bound on its size.

Yamashita and Kameda have investigated computability on anonymous arbitrary graphs in [YK96a]. They assume either that the topology of the network or a bound on the size is initially known by the processes. They use the notion of views to characterize computable functions. In [BV02b], Boldi and Vigna characterize what can be computed in a self-stabilizing way in a synchronous setting. This result enables them to characterize what can be computed in anonymous networks with an implicit termination. This characterization is based on fibrations and coverings, that are some tools we use in this paper. In [BV01], Boldi

and Vigna characterize what can be computed on an anonymous networks with local termination provided the processes initially know a bound on the size of the network. This characterization is the same as for implicit termination.

In [BV99], Boldi and Vigna consider tasks computable with local termination with arbitrary knowledge. Their characterization is based on partial views and is really different from the one given in [BV01]. As explained by Boldi and Vigna in [BV99], in all these works (except [BV99]), the processes initially know at least a bound on the size of the network. In this case, all kinds of terminations are equivalent: what can be computed with implicit termination can be computed with global termination detection. In the literature, one can found different algorithms to detect global termination provided that there exists a leader [DS80], that processes have unique ids [Mat87], or that processes know a bound on the diameter of the network [SSP85]. A characterization of tasks computable with global termination detection is presented in [CGMT07].

Our Results. In this regard where termination appears as a natural and key parameter for unification of distributed computability results – the link made by Boldi and Vigna in [BV02b] between computability with implicit termination on anonymous network and self-stabilization is very enlightening –, we present here two characterizations of computability with local and weak local terminations on asynchronous message passing networks where there is *no failure* in the communication system. By considering arbitrary families of labelled graphs, one can model arbitrary initial knowledge and arbitrary level of anonymity (from completely anonymous to unique ids).

We characterize the tasks that are computable with *weak local termination* (Theorem 5.2). Such tasks are interesting, because weak local termination is a good notion to compose distributed algorithms. Indeed, it is not necessary to ensure that all processes have terminated executing the first algorithm before starting the second one. We show that the following intuitive idea is necessary and sufficient for computability with weak local termination: if the k-neighbourhoods of two processes v, w cannot be distinguished locally, then if v computes its final value in k steps, w computes the same final value in k steps.

Then, we present a new characterization of the tasks that are computable with local termination (Theorem 8.3). Our characterization is built upon the one for weak local termination. When we deal with local termination, one has also to take into account that the subgraph induced by the processes that have not terminated may become disconnected during the execution. In some cases, it is impossible to avoid such a situation to occur (see Section 3 for examples).

With the results from [BV02b,CGMT07], we now get characterizations of computability for each kind of termination we discussed. What is interesting is that all of them can be expressed using the same combinatorial tools.

Moreover, the complexity of our universal algorithms is better than the view-based algorithms of Boldi and Vigna and of Yamashita and Kameda that necessitate exchanges of messages of exponential size. It enables us to characterize tasks that are computable with (weak) local termination by polynomial algorithms, i.e., algorithms where for each execution, the number of rounds, the number and the size of the messages are polynomial in the size of the network.

2 Definitions

The Model. Our model corresponds to the usual asynchronous message passing model [Tel00,AW04]. A network is represented by a simple connected graph G where vertices correspond to processes and edges to direct communication links. The state of each process is represented by a label $\lambda(v)$ associated to the corresponding vertex $v \in V(G)$; we denote by $\mathbf{G} = (G, \lambda)$ such a labelled graph. We assume that each process can distinguish the different edges that are incident to it, i.e., for each $u \in V(G)$ there exists a bijection δ_u between the neighbours of u in G and $[1, \deg_G(u)]$ (thus, u knows $\deg_G(u)$). We denote by δ the set of functions $\{\delta_u \mid u \in V(G)\}$. The numbers associated by each vertex to its neigbours are called *port-numbers* and δ is called a *port-numbering* of G. A *network* is a labelled graph $\mathbf{G}$ with a port-numbering δ and is denoted by $(\mathbf{G}, \delta)$.

Each processor v in the network represents an entity that is capable of performing computation steps, sending messages via some port and receiving any message via some port that was sent by the corresponding neighbour. We consider asynchronous systems, i.e., each of the steps of execution may take an unpredictable (but finite) amount of time. Note that we consider only reliable systems: no fault can occur on processes or communication links. We also assume that the channels are FIFO, i.e., for each channel, the messages are delivered in the order they have been sent. In this model, a distributed algorithm is given by a local algorithm that all processes should execute (note that all the processes have the same algorithm). A local algorithm consists of a sequence of computation steps interspersed with instructions to send and to receive messages.

In the paper, we sometimes refer to the *synchronous* execution of an algorithm. Such an execution is a particular execution of the algorithm that can be divided in *rounds*. In each round, each process receives all the messages that have been sent to it by its neighbours in the previous round; then according to the information it gets, it can modify its state and send messages to its neighbours before entering the next round. Note that the synchronous execution of an algorithm is just a special execution of the algorithm and thus it belongs to the set of asynchronous executions of this algorithm.

Distributed Tasks and Terminations. As mentioned earlier, when we are interested in computing a task in a distributed way, we have to distinguish what kind of termination we want to compute the task with. Given a family $\mathcal{F}$ of networks, a network $(\mathbf{G}, \delta) \in \mathcal{F}$ and a process v in $(\mathbf{G}, \delta)$, we assume that the state of v during the execution of any algorithm is of the form $(\texttt{mem}(v), \texttt{out}(v), \texttt{term}(v))$: $\texttt{mem}(v)$ is the memory of v, $\texttt{out}(v)$ is its output value and $\texttt{term}(v)$ is a flag in $\{\textsc{Term}, \bot\}$ mentioning whether v has computed its final value or not. The initial state of v is $(\texttt{in}(v), \bot, \bot)$ where the input $\texttt{in}(v)$ is the label of v in $(\mathbf{G}, \delta)$.

A *distributed task* is a couple $(\mathcal{F}, \mathcal{S})$ where $\mathcal{F}$ is a family of labelled graphs and $\mathcal{S}$ is a vertex-relabelling relation (i.e., if $((G, \lambda), \delta)$ $\mathcal{S}$ $((G', \lambda'), \delta')$, then $G = G'$ and $\delta = \delta'$) such that for every $(\mathbf{G}, \delta) \in \mathcal{F}$, there exists $(\mathbf{G}', \delta)$ such that $(\mathbf{G}, \delta)$ $\mathcal{S}$ $(\mathbf{G}', \delta)$. The set $\mathcal{F}$ is the *domain* of the task, $\mathcal{S}$ is the *specification* of the task. The classical leader election problem on some family $\mathcal{F}$ of networks is described in our settings by a task $(\mathcal{F}, \mathcal{S})$ where for each $(\mathbf{G}, \delta) \in \mathcal{F}$,

$(\mathbf{G}, \delta)$ $\mathcal{S}$ $(\mathbf{G}', \delta)$ for any $\mathbf{G}' = (G, \lambda')$ such that there is a unique $v \in V(G)$ with $\lambda'(v) = \texttt{leader}$. Considering arbitrary families of labelled graphs enables to represent any initial knowledge: e.g. if the processes initially know the size of the network, then in the corresponding family $\mathcal{F}$, for each $(\mathbf{G}, \delta) \in \mathcal{F}$ and each $v \in V(G)$, $|V(G)|$ is a component of the initial label of v.

We say that an algorithm $\mathcal{A}$ has an *implicit termination* on $\mathcal{F}$ if for any execution of $\mathcal{A}$ on any graph $(\mathbf{G}, \delta) \in \mathcal{F}$, the network reaches a global state where no messages are in transit and the states of the processes are not modified any more. Such a global final state is called the *final configuration* of the execution.

Given a task $(\mathcal{F}, \mathcal{S})$, an algorithm $\mathcal{A}$ is *normalized* for $(\mathcal{F}, \mathcal{S})$ if $\mathcal{A}$ has an implicit termination on $\mathcal{F}$ and in the final configuration of any execution of $\mathcal{A}$ on some $(\mathbf{G}, \delta) \in \mathcal{F}$, for each $v \in V(G)$, $\texttt{term}(v) = \textsc{Term}$, $\texttt{out}(v)$ is defined and $((G, \texttt{in}), \delta)$ $\mathcal{S}$ $((G, \texttt{out}), \delta)$ (i.e., the output of the algorithm solves the task $\mathcal{S}$).

A task $(\mathcal{F}, \mathcal{S})$ is computable with *local termination* (Lt) if there exists a normalized algorithm $\mathcal{A}$ for $(\mathcal{F}, \mathcal{S})$ such that for each $v \in V(G)$, once $\texttt{term}(v) = \textsc{Term}$, $(\texttt{mem}(v), \texttt{out}(v), \texttt{term}(v))$ is not modified any more. A task $(\mathcal{F}, \mathcal{S})$ is computable with *weak local termination* (wLt) if there exists a normalized algorithm $\mathcal{A}$ for $(\mathcal{F}, \mathcal{S})$ such that for each $v \in V(G)$, once $\texttt{term}(v) = \textsc{Term}$, $(\texttt{out}(v), \texttt{term}(v))$ is not modified any more.

For both terminations, one can show that we can restrict ourselves to tasks where $\mathcal{F}$ is recursively enumerable. A vertex v is *active* if it has not stopped the execution of the algorithm, i.e., v can still modify the value of $\texttt{mem}(v)$. When we consider weak local termination, all vertices always remain active, whereas when we consider local termination, a vertex is active if and only if $\texttt{term}(v) \neq \textsc{Term}$. If a vertex is not active anymore, we say that it is *inactive*.

3 Examples of Tasks with Different Kinds of Terminations

We present here three simple examples that demonstrate the hierarchy of terminations. We consider the family $\mathcal{F}$ containing all networks $((G, \texttt{in}), \delta)$ where for each vertex v, its input value has the form $\texttt{in}(v) = (val(v), d(v))$ where $val(v) \in \mathbb{N}$ and $d(v) \in \mathbb{N} \cup \{\infty\}$. The specification we are interested in is the following: in the final configuration, for each $(\mathbf{G}, \delta)$ and for each vertex $v \in V(G)$, $\texttt{out}(v) = \max\{val(u) \mid \text{dist}_G(u, v) \leq d(v)\}$[1].

We add some restrictions on the possible initial value for $d(v)$ in order to define three different tasks. In the general case (no restriction on the values of $d(v)$), the task we just described is called the Maximum Problem. If we consider the same task on the family $\mathcal{F}'$ containing all networks such that for each $v \in V(G)$, $d(v) \neq \infty$, the corresponding task is called the Local Maximum Problem. If we consider the same task on the family $\mathcal{F}''$ containing all networks such that for each edge $\{v, w\}$, $|d(v) - d(w)| \leq 1$, then we obtain a different problem that is called the Locally Bounded Maximum Problem.

The Maximum Problem can be solved with implicit termination by a flooding algorithm. Suppose now that there exists an algorithm $\mathcal{A}$ that can solve the Maximum Problem with wLt. Consider the synchronous execution of $\mathcal{A}$ over

[1] when $d(v) = \infty$, $\texttt{out}(v)$ is the maximum value $val(u)$ on the entire graph.

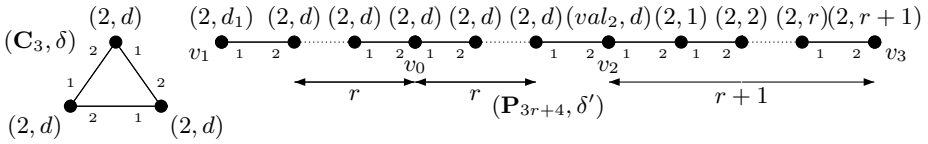

Fig. 1. Networks highlighting differences between the different kinds of termination.

the graph $(\mathbf{C}_3, \delta)$ of Figure 1 where $d = \infty$ and let r be the number of rounds of this execution; after r rounds, $\texttt{term}(v) = \textsc{Term}$ and $\texttt{out}(v) = 2$, for each $v \in V(C_3)$. Consider now the path $(\mathbf{P}_{3r+4}, \delta')$ on $3r + 4$ vertices of Figure 1 where $d = d_1 = \infty$ and $val_2 > 2$. After r synchronous rounds over $(\mathbf{P}_{3r+4}, \delta')$, v_0 gets exactly the same information as any $v \in V(C_3)$ and thus after r rounds, $\texttt{term}(v_0) = \textsc{Term}$ and $\texttt{out}(v_0) = 2$ whereas the correct output is $\texttt{out}(v_0) = val_2$. Thus $\mathcal{A}$ does not solve the Maximum Problem with wLt.

The Local Maximum Problem can be solved with weak local termination by a flooding algorithm running in waves. Suppose now that there exists an algorithm $\mathcal{A}$ that can solve the Local Maximum Problem with Lt. Consider the synchronous execution of $\mathcal{A}$ over the graph $(\mathbf{C}_3, \delta)$ of Figure 1 where $d = 1$ and let r be the number of rounds of this execution. Consider now the path $(\mathbf{P}_{3r+4}, \delta')$ of Figure 1 where $d = 1$, $val_2 > 2$ and $d_1 \geq 2r + 2$. After r synchronous rounds on P_{3r+4}, for the same reasons as before, $\texttt{term}(v_0) = \textsc{Term}$. Since v_0 has stopped before it knows val_2 and since after v_0 has stopped, no information can be transmitted through v_0, $\texttt{out}(v_1)$ cannot possibly be val_2, but the correct output is $\texttt{out}(v_1) = val_2$. Thus $\mathcal{A}$ does not solve the Local Maximum Problem with wLt.

The Locally Bounded Maximum Problem can be solved with local termination by the previous wave-based flooding algorithm. Suppose now that there exists an algorithm $\mathcal{A}$ that can solve the Locally Bounded Maximum Problem with global termination detection. Consider the synchronous execution of $\mathcal{A}$ over the graph $(\mathbf{C}_3, \delta)$ of Figure 1 where $d = 1$ and let r be the number of rounds of this execution; after r rounds, a vertex $v \in V(C_3)$ is in a state S indicating that all processes have computed their final values. Consider now the path $(\mathbf{P}_{3r+4}, \delta')$ of Figure 1 where $d_1 = d = 1$, $val_2 > 2$ and on the path $v_2 = w_0, w_1, \ldots, w_{r+1} = v_3$ between v_2 and v_3, for each $i \in [1, r]$, $d(w_i) = i$. After r synchronous rounds on P_{3r+4}, for the same reasons as before v is in the state S indicating that all processes have computed their final values. However, since $\text{dist}(v_2, v_3) = r + 1$, after r rounds, the vertex v_3 does not know the value of val_2 and thus $\texttt{out}(v_3)$ cannot possibly be val_2. Thus $\mathcal{A}$ does not solve the Locally Bounded Maximum Problem with global termination detection.

4 Digraphs and Coverings

Labelled Digraphs. In the following, we will consider directed graphs (digraphs) with multiple arcs and self-loops. A *digraph* $D = (V(D), A(D), s, t)$ is defined by a set $V(D)$ of vertices, a set $A(D)$ of arcs and by two maps s and t that assign to each arc two elements of $V(D)$: a source and a target. If a is an arc, we say that a is incident to $s(a)$ and $t(a)$. A *symmetric* digraph D is a

digraph endowed with a symmetry, that is, an involution $Sym : A(D) \to A(D)$ such that for every $a \in A(D), s(a) = t(Sym(a))$. In a symmetric digraph D, the degree of a vertex v is $\deg_D(v) = |\{a \mid s(a) = v\}| = |\{a \mid t(a) = v\}|$ and we denote by $N_D(v)$ the set of neighbours of v. Given two vertices $u, v \in V(D)$, a *path* π of length p from u to v in D is a sequence of arcs $a_1, a_2, \ldots a_p$ such that $s(a_1) = u, \forall i \in [1, p-1], t(a_i) = s(a_{i+1})$ and $t(a_p) = v$. If for each $i \in [1, p-1]$, $a_{i+1} \neq Sym(a_i)$, π is *non-stuttering*. A digraph D is *strongly connected* if for all vertices $u, v \in V(D)$, there exists a path from u to v in D. In a symmetric digraph D, the *distance* between two vertices u and v, denoted $\text{dist}_D(u, v)$ is the length of the shortest path from u to v in D.

A *homomorphism* γ between the digraph D and the digraph D' is a mapping $\gamma\colon V(D) \cup A(D) \to V(D') \cup A(D')$ such that for each arc $a \in A(D)$, $\gamma(s(a)) = s(\gamma(a))$ and $\gamma(t(a)) = t(\gamma(a))$. An homomorphism $\gamma : D \to D'$ is an *isomorphism* if γ is bijective.

Throughout the paper we will consider digraphs where the vertices and the arcs are labelled with labels from a recursive label set L. A digraph D labelled over L will be denoted by (D, λ), where $\lambda\colon V(D) \cup A(D) \to L$ is the labelling function. A mapping $\gamma\colon V(D) \cup A(D) \to V(D') \cup A(D')$ is a homomorphism from (D, λ) to (D', λ') if γ is a digraph homomorphism from D to D' which preserves the labelling, i.e., such that $\lambda'(\gamma(x)) = \lambda(x)$ for every $x \in V(D) \cup A(D)$. Labelled digraphs will be designated by bold letters like $\mathbf{D}, \mathbf{G}, \ldots$

In a symmetric digraph $\mathbf{D}$, we denote by $\mathbf{B_D}(v_0, r)$, the labelled ball of center $v_0 \in V(D)$ and of radius r that contains all vertices at distance at most r of v_0 and all arcs whose source or target is at distance at most $r-1$ of v_0.

Given a simple connected labelled graph $\mathbf{G} = (G, \lambda)$ with a port-numbering δ, we will denote by $(\text{Dir}(\mathbf{G}), \delta)$ the labelled digraph $(\text{Dir}(G), (\lambda, \delta))$ constructed in the following way. The vertices of $\text{Dir}(G)$ are the vertices of G and they have the same labels as in $\mathbf{G}$. Each edge $\{u, v\}$ is replaced by two arcs $a_{(u,v)}, a_{(v,u)} \in A(\text{Dir}(G))$ such that $s(a_{(u,v)}) = t(a_{(v,u)}) = u, t(a_{(u,v)}) = s(a_{(v,u)}) = v, \delta(a_{(u,v)}) = (\delta_u(v), \delta_v(u)), \delta(a_{(v,u)}) = (\delta_v(u), \delta_u(v))$ and $Sym(a_{(u,v)}) = a_{(v,u)}$. This construction encodes that a process can answer to a neighbour, i.e., "pong" any message.

Given a set L, we denote by $\mathcal{D}_L$ the set of all symmetric digraphs $\mathbf{D} = (D, \lambda)$ where for each $a \in A(D)$, there exist $p, q \in \mathbb{N}$ such that $\lambda(a) = (p, q)$ and $\lambda(Sym(a)) = (q, p)$ and for each $v \in V(D)$, $\lambda(v) \in L$ and $\{p \mid \exists a, \lambda(a) = (p, q) \text{ and } s(a) = v\} = [1, \deg_D(v)]$. In other words, $\mathcal{D}_L$ is the set of digraphs that locally look like some digraph obtained from a simple labelled graph $\mathbf{G}$.

Symmetric Coverings, Quasi-Coverings. The notion of symmetric coverings is fundamental in this work; definitions and main properties are presented in [BV02a]. This notion enables to express "similarity" between two digraphs.

A (labelled) digraph $\mathbf{D}$ is a *covering* of a digraph $\mathbf{D'}$ via φ if φ is a homomorphism from $\mathbf{D}$ to $\mathbf{D'}$ such that each arc $a' \in A(D')$ and for each vertex $v \in \varphi^{-1}(t(a'))$ (resp. $v \in \varphi^{-1}(s(a'))$, there exists a unique arc $a \in A(D)$ such that $t(a) = v$ (resp. $s(a) = v$) and $\varphi(a) = a'$. A symmetric digraph $\mathbf{D}$ is a *symmetric covering* of a symmetric digraph $\mathbf{D'}$ via φ if $\mathbf{D}$ is a covering of $\mathbf{D'}$ via φ and if for each arc $a \in A(D)$, $\varphi(Sym(a)) = Sym(\varphi(a))$.

The following lemma shows the importance of symmetric coverings when we deal with anonymous networks. This is the counterpart of the lifting lemma that Angluin gives for coverings of simple graphs [Ang80] and the proof can be found in [BCG$^+$96,CM07].

Lemma 4.1 (Lifting Lemma [BCG$^+$96]). *If $\mathbf{D}$ is a symmetric covering of $\mathbf{D}'$ via φ, then any execution of an algorithm $\mathcal{A}$ on $\mathbf{D}'$ can be lifted up to an execution on $\mathbf{D}$, such that at the end of the execution, for any $v \in V(D)$, v is in the same state as $\varphi(v)$.*

In the following, one also needs to express similarity between two digraphs up to a certain distance. The notion of quasi-coverings was introduced in [MMW97] for this purpose. The next definition is an adaptation of this tool to digraphs.

Definition 4.2. *Given two symmetric labelled digraphs $\mathbf{D}_0, \mathbf{D}_1$, an integer r, a vertex $v_1 \in V(D_1)$ and a homomorphism γ from $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ to $\mathbf{D}_0$, the digraph $\mathbf{D}_1$ is a* quasi-covering *of $\mathbf{D}_0$ of center v_1 and of radius r via γ if there exists a finite or infinite symmetric labelled digraph $\mathbf{D}_2$ that is a symmetric covering of $\mathbf{D}_0$ via a homomorphism φ and if there exist $v_2 \in V(D_2)$ and an isomorphism δ from $\mathbf{B}_{\mathbf{D}_1}(v_1, r)$ to $\mathbf{B}_{\mathbf{D}_2}(v_2, r)$ such that for any $x \in V(B_{D_1}(v_1, r)) \cup A(B_{D_1}(v_1, r))$, $\gamma(x) = \varphi(\delta(x))$.*

If a digraph $\mathbf{D}_1$ is a symmetric covering of $\mathbf{D}_0$, then for any $v \in V(D_1)$ and for any $r \in \mathbb{N}$, $\mathbf{D}_1$ is a quasi-covering of $\mathbf{D}_0$, of center v and of radius r. Reversely, if $\mathbf{D}_1$ is a quasi-covering of $\mathbf{D}_0$ of radius r strictly greater than the diameter of $\mathbf{D}_1$, then $\mathbf{D}_1$ is a covering of $\mathbf{D}_0$. The following lemma is the counterpart of the lifting lemma for quasi-coverings.

Lemma 4.3 (Quasi-Lifting Lemma). *Consider a digraph $\mathbf{D}_1$ that is a quasi-covering of $\mathbf{D}_0$ of center v_1 and of radius r via γ. For any algorithm $\mathcal{A}$, after r rounds of the synchronous execution of an algorithm $\mathcal{A}$ on $\mathbf{D}_1$, v_1 is in the same state as $\gamma(v_1)$ after r rounds of the synchronous execution of $\mathcal{A}$ on $\mathbf{D}'$.*

5 Characterization for Weak Local Termination

We note $\mathcal{V}$ the set $\{(\mathbf{D}, v) \mid \mathbf{D} \in \mathcal{D}_L, v \in V(D)\}$. In other words, the set $\mathcal{V}$ is the disjoint union of all symmetric labelled digraphs in $\mathcal{D}_L$. Given a family of networks $\mathcal{F}$, we denote by $\mathcal{V}_\mathcal{F}$ the set $\{((\mathrm{Dir}(\mathbf{G}), \delta), v) \mid (\mathbf{G}, \delta) \in \mathcal{F}, v \in V(G)\}$.

A function $f : \mathcal{V} \longrightarrow L \cup \{\perp\}$ is an *output* function for a task $(\mathcal{F}, \mathcal{S})$ if for each network $(\mathbf{G}, \delta) \in \mathcal{F}$, the labelling obtained by applying f on each $v \in V(G)$ satisfies the specification $\mathcal{S}$, i.e., $(\mathbf{G}, \delta)\ \mathcal{S}\ (\mathbf{G}', \delta)$ where $\mathbf{G}' = (G, \lambda')$ and $\lambda'(v) = f((\mathrm{Dir}(\mathbf{G}), \delta), v)$ for all $v \in V(G)$.

In order to give our characterization, we need to formalize the following idea. When the neighbourhood at distance k of two processes v, v' in two digraphs $\mathbf{D}, \mathbf{D}'$ cannot be distinguished (this is captured by the notion of quasi-coverings and Lemma 4.3), and if v computes its final value in less than k rounds, then v' computes the same final value in the same number of rounds. In the following definition, the value of $r(\mathbf{D}, v)$ can be understood as the number of rounds needed by v to compute in a synchronous execution its final value in $\mathbf{D}$.

Definition 5.1. *Given a function $r : \mathcal{V} \longrightarrow \mathbb{N} \cup \{\infty\}$ and a function $f : \mathcal{V} \longrightarrow L'$ for some set L', the function f is* r-lifting closed *if for all $\mathbf{D}, \mathbf{D}' \in \mathcal{D}_L$ such that $\mathbf{D}$ is a quasi-covering of $\mathbf{D}'$, of center $v_0 \in V(G)$ and of radius R via γ with $R \geq \min\{r(\mathbf{D}, v_0), r(\mathbf{D}', \gamma(v_0))\}$, then $f(\mathbf{D}, v_0) = f(\mathbf{D}', \gamma(v_0))$.*

Using the previous definition, we now give the characterization of tasks computable with WLT. We also characterize distributed tasks computable with WLT by polynomial algorithms (using a polynomial number of messages of polynomial size). We denote by $|\mathbf{G}|$ the size of $V(G)$ plus the maximum over the sizes (in bits) of the initial labels that appear on $\mathbf{G}$.

Theorem 5.2. *A task $(\mathcal{F}, \mathcal{S})$ where $\mathcal{F}$ is recursively enumerable is computable with* WLT *if and only if there exist a function $r : \mathcal{V} \longrightarrow \mathbb{N} \cup \{\infty\}$ and an output function $f : \mathcal{V} \longrightarrow L \cup \{\perp\}$ for $(\mathcal{F}, \mathcal{S})$ such that*

(i) for all $\mathbf{D} \in \mathcal{D}_L$, for all $v \in V(D)$, $r(\mathbf{D}, v) \neq \infty$ if and only if $f(\mathbf{D}, v) \neq \perp$,
(ii) $f_{|\mathcal{V}_\mathcal{F}}$ and $r_{|\mathcal{V}_\mathcal{F}}$ are recursive functions,
(iii) f and r are r-lifting-closed.

The task $(\mathcal{F}, \mathcal{S})$ is computable by a polynomial algorithm with WLT *if and only if there exist such f and r and a polynomial p such that for each $(\mathbf{G}, \delta) \in \mathcal{F}$ and each $v \in V(G)$, $r((\mathrm{Dir}(\mathbf{G}), \delta), v) \leq p(|\mathbf{G}|)$.*

Proof (of the necessary condition). Assume $\mathcal{A}$ is a distributed algorithm that computes the task $(\mathcal{F}, \mathcal{S})$ with weak local termination. We define r and f by considering the synchronous execution of $\mathcal{A}$ on any digraph $\mathbf{D} \in \mathcal{D}_L$. For any $v \in V(D)$, if $\mathtt{term}(v) = \perp$ during the whole execution, then $f(\mathbf{D}, v) = \perp$ and $r(\mathbf{D}, v) = \infty$. Otherwise, let r_v be the first round after which $\mathtt{term}(v) =$ TERM; in this case, $f(\mathbf{D}, v) = \mathtt{out}(v)$ and $r(\mathbf{D}, v) = r_v$. Since $\mathcal{A}$ computes $(\mathcal{F}, \mathcal{S})$, it is easy to see that f is an output function and that f and r satisfy (i) and (ii).

Consider two digraphs $\mathbf{D}, \mathbf{D}' \in \mathcal{D}_L$ such that $\mathbf{D}$ is a quasi-covering of $\mathbf{D}'$, of center $v_0 \in V(G)$ and of radius R via γ with $R \geq r_0 = \min\{r(\mathbf{D}, v_0), r(\mathbf{D}', \gamma(v_0))\}$. If $r_0 = \infty$, then $r(\mathbf{D}, v_0) = r(\mathbf{D}', \gamma(v_0)) = \infty$ and $f(\mathbf{D}, v_0) = f(\mathbf{D}', \gamma(v_0)) = \perp$. Otherwise, from Lemma 4.3, we know that after r_0 rounds, $\mathtt{out}(v_0) = \mathtt{out}(\gamma(v_0))$ and $\mathtt{term}(v_0) = \mathtt{term}(\gamma(v_0)) =$ TERM. Thus $r_0 = r(\mathbf{D}, v_0) = r(\mathbf{D}', \gamma(v_0))$ and $f(\mathbf{D}, v_0) = f(\mathbf{D}', \gamma(v_0))$. Consequently, f and r are r-lifting closed. □

The sufficient condition is proved in Section 7 and relies on a general algorithm described in Section 6. Using this theorem, one can show that there is no universal election algorithm for the family of networks with non-unique ids where at least one id is unique, but that there exists such an algorithm for such a family where a bound on the multiplicity of each id in any network is known.

6 A General Algorithm

In this section, we present a general algorithm that we parameterize by the task and the termination we are interested in, in order to obtain our sufficient conditions. This algorithm is a combination of an election algorithm for symmetric minimal graphs presented in [CM07] and a generalization of an algorithm of

Szymanski, Shy and Prywes (the SSP algorithm for short) [SSP85]. The algorithm described in [CM07] is based on an enumeration algorithm presented by Mazurkiewicz in a different model [Maz97] where each computation step involves some synchronization between adjacent processes. The SSP algorithm enables to detect the global termination of an algorithm with local termination provided the processes know a bound on the diameter of the graph. The Mazurkiewicz-like algorithm always terminates (implicitly) on any network $(\mathbf{G}, \delta)$ and during its execution, each process v can reconstruct at some computation step i a digraph $\mathbf{D}_i(v)$ such that $(\mathrm{Dir}(\mathbf{G}), \delta)$ is a quasi-covering of $\mathbf{D}_i(v)$. However, this algorithm does not enable v to compute the radius of this quasi-covering. We use a generalization of the SSP algorithm to compute a lower bound on this radius, as it has already be done in Mazurkiewicz's model [GMT06].

We consider a network $(\mathbf{G}, \delta)$ where $\mathbf{G} = (G, \lambda)$ is a simple labelled graph and where δ is a port-numbering of $\mathbf{G}$. The function $\lambda : V(G) \to L$ is the initial labelling. We assume there exists a total order $<_L$ on L and we assume that if the label $\lambda(v)$ is modified during the execution, then it can only increase for $<_L$.

The state of each v is a tuple $(\lambda(v), n(v), N(v), M(v), a(v), A(v))$ where:

- $\lambda(v) \in L$ is the initial label of v and if it is modified during the execution, it will necessarily increase for $<_L$.
- $n(v) \in \mathbb{N}$ is the *number* of v computed by the algorithm; initially $n(v) = 0$.
- $N(v) \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N} \times L \times \mathbb{N}^2)$[2] is the *local view* of v. At the end of the execution, if $(m, \ell, p, q) \in N(v)$, then v has a neighbour u whose number is m, whose label is ℓ and the arc from u to v is labelled (p, q). Initially $N(v) = \{(0, \perp, 0, q) \mid q \in [1, \deg_G(v)]\}$.
- $M(v) \subseteq \mathbb{N} \times L \times \mathcal{P}_{\mathrm{fin}}(\mathbb{N} \times L \times \mathbb{N}^2)$ is the *mailbox* of v; initially $M(v) = \emptyset$. It contains all information received by v during the execution of the algorithm. If $(m, \ell, N) \in M(v)$, it means that at some previous step of the execution, there was a vertex u such that $n(u) = m$, $\lambda(u) = \ell$ and $N(u) = N$.
- $a(v) \in \mathbb{Z} \cup \{\infty\}$ is a counter and initially $a(v) = -1$. In some sense, $a(v)$ represent the distance up to which all vertices have the same mailbox as v. If $a(v) = \infty$, it means that v has terminated the algorithm (local termination).
- $A(v) \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N} \times (\mathbb{Z} \cup \{\infty\}))$ encodes the information v has about the values of $a(u)$ for each neighbour u. Initially, $A(v) = \{(q, -1) \mid q \in [1, \deg_G(v)]\}$.

In our algorithm, processes exchange messages of the form $< (n, \ell, M, a), p >$. If a vertex u sends a message $< (n, \ell, M, a), p >$ to one of its neighbour v, then the message contains following information: n is the current number $n(u)$ of u, ℓ is the label $\lambda(u)$ of u, M is the mailbox of u, a is the value of $a(u)$ and $p = \delta_u(v)$.

As in Mazurkiewicz's algorithm [Maz97], the nice properties of the algorithm rely on a total order on local views, i.e., on finite subsets of $\mathbb{N}^3 \times L$. Given two distinct sets $N_1, N_2 \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N} \times L \times \mathbb{N}^2)$, we define $N_1 \prec N_2$ if the maximum of the symmetric difference $N_1 \triangle N_2 = (N_1 \setminus N_2) \cup (N_2 \setminus N_1)$ for the lexicographic order belongs to N_2. One also says that $(\ell, N) \prec (\ell', N')$ if either $\ell <_L \ell'$, or $\ell = \ell'$ and $N \prec N'$. We denote by $\preceq$ the reflexive closure of $\prec$.

[2] for any set S, $\mathcal{P}_{\mathrm{fin}}(S)$ denotes the set of finite subsets of S

Our algorithm $\mathcal{A}_{gen}(\varphi)$ is described in Algorithm 1. The algorithm for the vertex v_0 is expressed in an event-driven description. The first rule **I** can be applied by a process v on wake-up only if it has not received any message: it takes the number 1, updates its mailbox and informs its neighbours. The second rule **R** describes the instructions a process v has to follow when it receives a message m from a neighbour. It updates its mailbox $M(v)$ and its local view $N(v)$ according to m. Then, if it discovers the existence of another vertex with the same number and a stronger local view, it takes a new number. Then, if its mailbox has not changed, it updates $A(v)$ and increases $a(v)$ if possible (according to a function φ). Finally, if $M(v)$ or $a(v)$ has been modified, it informs its neighbours.

Later, we will add rules that enable a process to compute its final value and we will define the function $\varphi(v)$ depending on the termination we are interested in. Using the information stored in its mailbox, each v will be able to reconstruct a digraph $\mathbf{D}$ such that $(\mathrm{Dir}(\mathbf{G}), \delta)$ locally looks like $\mathbf{D}$ up to distance $a(v)$.

Algorithm 1: The general algorithm $\mathcal{A}_{gen}(\varphi)$.

I : $\{n(v_0) = 0$ and no message has arrived at $v_0\}$
begin
  $n(v_0) := 1$;
  $M(v_0) := \{(n(v_0), \lambda(v_0), \emptyset)\}$;
  $a(v_0) := 0$;
  for $i := 1$ **to** $\deg(v_0)$ **do**
    send $< (n(v_0), \lambda(v), M(v_0), a(v_0)), i >$ through i ;
end

R : $\{$A message $< (n_1, \ell_1, M_1, a_1), p_1 >$ has arrived at v_0 through port $q_1\}$
begin
  $M_{old} := M(v_0)$;
  $a_{old} := a(v_0)$;
  $M(v_0) := M(v_0) \cup M_1$;
  if $n(v_0) = 0$ *or* $\exists (n(v_0), \ell', N') \in M(v_0)$ *such that* $(\lambda(v_0), N(v_0)) \prec (\ell', N')$ **then**
    $n(v_0) := 1 + \max\{n' \mid \exists (n', \ell', N') \in M(v_0)\}$;
  $N(v_0) := N(v_0) \setminus \{(n', \ell', p', q_1) \mid \exists (n', \ell', p', q_1) \in N(v_0)\} \cup \{(n_1, \ell_1, p_1, q_1)\}$;
  $M(v_0) := M(v_0) \cup \{(n(v_0), \lambda(v_0), N(v_0))\}$;
  if $M(v_0) \neq M_{old}$ **then**
    $a(v_0) := -1$;
    $A(v_0) := \{(q', -1) \mid \exists (q', a') \in A(v_0)$ with $a' \neq \infty\}$;
  if $M(v_0) = M_1$ **then**
    $A(v_0) := A(v_0) \setminus \{(q_1, a') \mid \exists (q_1, a') \in A(v_0)\} \cup \{(q_1, a_1)\}$;
  if $\forall (q', a') \in A(v_0), a(v_0) \leq a'$ *and* $(\varphi(v) = \textsc{True}$ *or* $\exists (q', a') \in A(v)$ *such that* $a(v_0) < a'$ *and* $a' \neq \infty)$ **then** $a(v_0) := a(v_0) + 1$;
  if $M(v_0) \neq M_{old}$ *or* $a(v_0) \neq a_{old}$ **then**
    for $q := 1$ **to** $\deg(v_0)$ **do**
      if $(q, \infty) \notin A(v)$ **then**
        send $< (n(v_0), \lambda(v), M(v_0), a(v_0)), q >$ through port q ;
end

Properties of the Algorithm. We consider a graph $\mathbf{G}$ with a port numbering δ and an execution of Algorithm 1 on $(\mathbf{G}, \delta)$. For each vertex $v \in V(G)$, we note $(\lambda_i(v), n_i(v), N_i(v), M_i(v), a_i(v), A_i(v))$ the state of v after the ith computation step. The following proposition summarizes some nice properties that are satisfied during any execution of Algorithm 1 on $(\mathbf{G}, \delta)$.

Proposition 6.1 ([CM07,Cha06]). *Consider a vertex v and a step $i \geq 1$. Then, $\lambda_{i-1}(v) \leq_L \lambda_i(v)$, $n_{i-1}(v) \leq n_i(v)$, $N_{i-1}(v) \preceq N_i(v)$, $M_{i-1}(v) \subseteq M_i(v)$.*

If $M_{i-1}(v) = M_i(v)$ and if v is active at step i, then $a_{i-1}(v) \leq a_i(v) \leq a_{i-1}(v) + 1$ and $a_i(v) \geq \min\{a \mid \exists (q,a) \in A_i(v)\}$ if $\exists (q,a) \in A_i(v)$ with $a \neq \infty$.

For each $(m, \ell, N) \in M_i(v)$ and each $m' \in [1, m]$, $\exists (m', \ell', N') \in M_i(v), \exists v' \in V(G)$ such that $n_i(v') = m'$. If $m = n_i(v)$, $(\ell, N) \leq (\lambda_i(v), N_i(v))$.

If $a_i(v) \geq 1$, for each $w \in N_G(v)$, there exists a step $j \leq i-1$ such that w is inactive at step j, or $a_j(w) \geq a_i(v) - 1$ and $M_j(w) = M_i(v)$.

An interesting corollary of Proposition 6.1 is that if the label $\lambda(v)$ of each v is modified only finitely many times, then there exists a step i_0 after which for any v, the value of $(\lambda(v), n(v), N(v), M(v))$ is not modified any more.

7 Tasks computable with Weak Local Termination

In order to show that the conditions of Theorem 5.2 are sufficient, we use the general algorithm presented in Section 6 parameterized by the functions f and r. In the following, we consider a function $\mathsf{enum}_{\mathcal{F}}$ that enumerates the elements of $\mathcal{F}$. During the execution of this algorithm on any graph $(\mathbf{G}, \delta) \in \mathcal{F}$, for any $v \in V(G)$, the value of $\lambda(v) = \mathtt{in}(v)$ is not modified.

Consider the mailbox $M = M(v)$ of a vertex v during the execution of the algorithm $\mathcal{A}_{gen}$ on a graph $(\mathbf{G}, \delta) \in \mathcal{F}$. We say that an element $(n, \ell, N) \in M$ is *maximal* in M if there does not exists $(n, \ell', N') \in M$ such that $(\ell, N) \prec (\ell', N')$. We denote by $S(M)$ be the set of maximal elements of M. From Proposition 6.1, after each step of Algorithm 1, $(n(v), \lambda(v), N(v))$ is maximal in $M(v)$. The set $S(M)$ is said *stable* if it is non-empty and if for all $(n_1, \ell_1, N_1) \in S(M)$, for all $(n_2, \ell_2, p, q) \in N_1$, $p \neq 0$, $n_2 \neq 0$ and $\ell_2 \neq \perp$ and for all $(n_2', \ell_2', N_2') \in S(M)$, there exists $(n_2', \ell_2'', p', q') \in N_1$ if and only if $\ell_2' = \ell_2''$ and $(n_1, \ell_1, q', p') \in N_2'$. From [CM07], we know that once the values of $n(v), N(v), M(v)$ are final, then $S(M(v))$ is stable. Thus, if $S(M(v))$ is not stable, $M(v)$ will be modified.

If the set $S(M)$ is stable, one can construct a labelled symmetric digraph $\mathbf{D}_M = (D_M, \lambda_M)$ as follows. The set of vertices $V(D_M)$ is the set $\{n \mid \exists (n, \ell, N) \in S(M)\}$. For any $(n, \ell, N) \in S(M)$ and any $(n', \ell', p, q) \in N$, there exists an arc $a_{n,n',p,q} \in A(D_M)$ such that $t(a) = n, s(a) = n'$, $\lambda_M(a) = (p, q)$. Since $S(M)$ is stable, we can define Sym by $Sym(a_{n,n',p,q}) = a_{n',n,q,p}$.

Proposition 7.1. *If $S(M(v))$ is stable, $(\mathrm{Dir}(\mathbf{G}), \delta)$ is a quasi-covering of $\mathbf{D}_{M(v)}$ of radius $a(v)$ of center v via a mapping γ where $\gamma(v) = n(v)$.*

Thus, once v has computed $\mathbf{D}_{M(v)}$, it can enumerate networks $(\mathbf{K}', \delta_K') \in \mathcal{F}$ and vertices $w' \in V(K')$ until it finds a $(\mathbf{K}', \delta_K')$ such that $\mathbf{K}(v) = (\mathrm{Dir}(\mathbf{K}'), \delta_{K'})$ is a quasi-covering of $\mathbf{D}_{M(v)}$ of center $w(v) \in V(K)$ and of radius $a(v)$ via some homomorphism γ such that $\gamma(w(v)) = n(v)$ (this enumeration terminates by Proposition 7.1). We add a rule to the algorithm, called $\textsc{wLt}(\mathsf{enum}_{\mathcal{F}}, f, r)$, that

a process v can apply to computes its final value, once it has computed $\mathbf{K}(v)$ and $w(v)$. We also add priorities between rules such that a vertex that can apply the rule $\text{WLT}(f, r)$ cannot apply the rule $\mathbf{R}$ of algorithm $\mathcal{A}_{gen}(\varphi)$.

Procedure `wLT(`*enum*$_{\mathcal{F}}$`,f,r)`: the rule added to the algorithm for WLT

if *term*$(v_0) \neq$ TERM *and* $a(v_0) \geq r(\mathbf{K}(v_0), w(v_0))$ **then**
  `out`$(v_0) := f(\mathbf{K}(v_0), w(v_0))$;
  `term`$(v_0) :=$ TERM ;

We now define the function φ that enables a vertex v to increase $a(v)$. The function φ is true for v only if `term`$(v) \neq$ TERM and $S(M(v))$ is stable (otherwise, v knows that its mailbox will be modified in the future) and $r(\mathbf{K}(v), w(v)) > a(v)$ (otherwise, v can compute its final value).

Correction of the Algorithm. We denote by $\mathcal{A}_{\text{WLT}}(\mathsf{enum}_{\mathcal{F}}, f, r)$ the algorithm defined by $\mathcal{A}_{gen}(\varphi)$ and by $\text{WLT}(\mathsf{enum}_{\mathcal{F}}, f, r)$. We consider a network $(\mathbf{G}, \delta) \in \mathcal{F}$ and an execution of $\mathcal{A}_{\text{WLT}}(\mathsf{enum}_{\mathcal{F}}, f, r)$ on $(\mathbf{G}, \delta)$. Let $\mathbf{G'} = (\text{Dir}(\mathbf{G}), \delta)$.

Using Propositions 6.1 and 7.1, one can show that the execution terminates (implicitly) and that in the final configuration, for any $v \in V(G)$, `term`$(v) =$ TERM. Since f is an output function for $(\mathcal{F}, \mathcal{S})$, the next proposition shows that $\mathcal{A}_{\text{WLT}}(\mathsf{enum}_{\mathcal{F}}, f, r)$ computes the task $(\mathcal{F}, \mathcal{S})$ with weak local termination.

Proposition 7.2. *For any $v \in V(G)$, if* *term*$(v) =$ TERM, *out*$(v) = f(\mathbf{G'}, v)$.

Proof. Consider a process v just after it has applied Procedure $\text{WLT}(\mathsf{enum}_{\mathcal{F}}, f, r)$: $S(M(v))$ is stable, $r(\mathbf{K}(v), w(v)) \leq a(v)$ and `out`$(v) = f(\mathbf{K}(v), w(v))$.

Since $\mathbf{K}(v)$ is a quasi-covering of $\mathbf{D}_{M(v)}$ of radius $a(v) \geq r(\mathbf{K}(v), w(v))$ and of center $w(v)$ via a mapping γ such that $\gamma(w(v)) = n(v)$ and since f and r are r-lifting closed, `out`$(v) = f(\mathbf{K}(v), w(v)) = f(\mathbf{D}_{M(v)}, n(v))$ and $r(\mathbf{K}(v), w(v)) = r(\mathbf{D}_{M(v)}, n(v))$. From Proposition 7.1, since $a(v) \geq r(\mathbf{D}_{M(v)}, n(v))$ and since f is r-lifting closed, `out`$(v) = f(\mathbf{D}_{M(v)}, n(v)) = f(\mathbf{G'}, v)$. □

8 Tasks Computable with Local Termination.

When we consider local termination, one needs to consider the case where some vertices that terminate quickly disconnect the graph induced by active vertices.

We extend to symmetric digraphs the notion of views that have been introduced to study leader election by Yamashita and Kameda [YK96b] for simple graphs and by Boldi *et al.* [BCG+96] for digraphs.

Definition 8.1. *Consider a symmetric digraph $\mathbf{D} = (D, \lambda) \in \mathcal{D}_L$ and a vertex $v \in V(D)$. The* view *of v in $\mathbf{D}$ is an infinite rooted tree denoted by $\mathbf{T}_{\mathbf{D}}(v) = (T_D(v), \lambda')$ and defined as follows:*

- *$V(T_D(v))$ is the set of non-stuttering paths $\pi = a_1, \ldots, a_p$ in $\mathbf{D}$ with $s(a_1) = v$. For each path $\pi = a_1, \ldots, a_p$, $\lambda'(\pi) = \lambda(t(a_p))$.*
- *for each $\pi, \pi' \in V(T_D(v))$, there are two arcs $a_{\pi,\pi'}, a_{\pi',\pi} \in A(T_D(v))$ such that $Sym(a_{\pi,\pi'}) = a_{\pi',\pi}$ if and only if $\pi' = \pi, a$. In this case, $\lambda'(a_{\pi,\pi'}) = \lambda(a)$ and $\lambda'(a_{\pi',\pi}) = \lambda(Sym(a))$.*

– *the root of $T_D(v)$ is the vertex corresponding to the empty path and its label is $\lambda(v)$.*

Consider the view $\mathbf{T_D}(v)$ of a vertex v in a digraph $\mathbf{D} \in \mathcal{D}_L$ and an arc a such that $s(a) = v$. We define $\mathbf{T_{D-}}_a(v)$ be the infinite tree obtained from $\mathbf{T_D}(v)$ by removing the subtree rooted in the vertex corresponding to the path a. Given $n \in \mathbb{N}$ and an infinite tree $\mathbf{T}$, we note $\mathbf{T}\restriction^n$ the truncation of the tree at depth n. Thus the truncation of the view at depth n of a vertex v in a symmetric digraph $\mathbf{D}$ is denoted by $\mathbf{T_D}(v)\restriction^n$. It is easy to see that for any $\mathbf{D} \in \mathcal{D}_L$, any $v \in V(D)$ and any integer $n \in \mathbb{N}$, $\mathbf{T_D}(v)\restriction^n$ is a quasi-covering of $\mathbf{D}$ of center v' and of radius n where v' is the root of $\mathbf{T_D}(v)\restriction^n$.

Given a digraph $\mathbf{D}$ and a process $v_0 \in V(D)$ that stops its computation after n steps, the only information any other process v can get from v_0 during the execution is contained in the n-neighbourhood of v_0. In order to take this property into account, we define an operator split. In $\mathsf{split}(\mathbf{D}, v_0, n)$, we remove v_0 from $V(D)$ and for each neighbour v of v_0, we add a new neighbour v'_0 to v that has a n-neighbourhood indistinguishable from the one of v_0. Thus, for any process $v \neq v_0$, in a synchronous execution, both v_0 and the vertices we have just added stop in the same state after n rounds and consequently v should behave in the same way in the two networks and stop with the same final value after the same number of rounds. This idea is formalized in Definition 8.2.

Given a digraph $\mathbf{D} = (D, \lambda) \in \mathcal{D}_L$, a vertex $v_0 \in V(D)$ and an integer $n \in \mathbb{N}$, $\mathsf{split}(\mathbf{D}, v_0, n) = (D', \lambda')$ is defined as follows. First, we remove v_0 and all its incident arcs from $\mathbf{D}$. Then for each arc $a \in A(D)$ such that $s(a) = v_0$, we add a copy of $\mathbf{T_{D-}}_a(v_0)\restriction^n$ to the graph. We denote by $v(a)$ the root of this tree and we add two arcs a_0, a_1 to the graph such that $Sym(a_0) = a_1$, $s(a_0) = t(a_1) = v(a)$, $s(a_1) = t(a_2) = t(a)$, $\lambda'(a_0) = \lambda(a)$ and $\lambda'(a_1) = \lambda(Sym(a))$. Note that for any vertex $v \neq v_0 \in V(D)$, v can be seen as a vertex of $\mathsf{split}(\mathbf{D}, v_0, n)$.

Definition 8.2. *Given a function $r : \mathcal{V} \longrightarrow \mathbb{N} \cup \{\infty\}$, a function $f : \mathcal{V} \longrightarrow L'$ is* r-splitting closed *if for any $\mathbf{D} \in \mathcal{D}_L$, for any vertex $v_0 \in V(D)$ and any vertex $v \neq v_0 \in V(D)$, $f(\mathbf{D}, v) = f(\mathsf{split}(\mathbf{D}, v_0, n), v)$ where $n = r(\mathbf{D}, v_0)$.*

We now give the characterization of tasks computable with LT. The proof is postponed to the journal version. For the necessary condition, we uses the same ideas as for the necessary condition of Theorem 5.2 and the proof of the sufficient condition also relies on the general algorithm presented in Section 6.

Theorem 8.3. *A task $(\mathcal{F}, S)$ where $\mathcal{F}$ is recursively enumerable is computable with* LT *if and only if there exist some functions f and r satisfying the conditions of Theorem 5.2 and such that f and r are r-splitting closed.*

The task $(\mathcal{F}, \mathcal{S})$ is computable by a polynomial algorithm with LT *if and only if there exist such f and r and a polynomial p such that for each $(\mathbf{G}, \delta) \in \mathcal{F}$ and each $v \in V(G)$, $r((\mathrm{Dir}(\mathbf{G}), \delta), v) \leq p(|\mathbf{G}|)$.*

References

[Ang80] D. Angluin. Local and global properties in networks of processors. In *Proc. of STOC'80*, pages 82–93, 1980.

[ASW88] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *J. ACM*, 35(4):845–875, 1988.

[AW04] H. Attiya and J. Welch. *Distributed computing: fundamentals, simulations, and advanced topics*. John Wiley and Sons, 2004.

[BCG+96] P. Boldi, B. Codenotti, P. Gemmell, S. Shammah, J. Simon, and S. Vigna. Symmetry breaking in anonymous networks: characterizations. In *Proc. of ISTCS'96*, pages 16–26. IEEE Press, 1996.

[BV99] P. Boldi and S. Vigna. Computing anonymously with arbitrary knowledge. In *Proc. of PODC'99*, pages 181–188. ACM Press, 1999.

[BV01] P. Boldi and S. Vigna. An effective characterization of computability in anonymous networks. In *Proc. of DISC'01*, volume 2180 of *LNCS*, pages 33–47. Springer-Verlag, 2001.

[BV02a] P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1-3):21–66, 2002.

[BV02b] P. Boldi and S. Vigna. Universal dynamic synchronous self-stabilization. *Distributed Computing*, 15(3):137–153, 2002.

[CGMT07] J. Chalopin, E. Godard, Y. Métivier, and G. Tel. About the termination detection in the asynchronous message passing model. In *Proc. of SOFSEM'07*, volume 4362 of *LNCS*, pages 200–211. Springer-Verlag, 2007.

[Cha06] J. Chalopin. *Algorithmique distribuée, calculs locaux et homomorphismes de graphes*. PhD thesis, Université Bordeaux 1, 2006.

[CM07] J. Chalopin and Y. Métivier. An efficient message passing election algorithm based on Mazurkiewicz's algorithm. *Fundamenta Informaticae*, 80(1–3):221–246, 2007.

[DP04] S. Dobrev and A. Pelc. Leader election in rings with nonunique labels. *Fundamenta Informaticae*, 59(4):333–347, 2004.

[DS80] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computation. *Information Processing Letters*, 11(1):1–4, 1980.

[FKK+04] P. Flocchini, E. Kranakis, D. Krizanc, F. Luccio, and N. Santoro. Sorting and election in anonymous asynchronous rings. *J. Parallel Distrib. Comput.*, 64(2):254–265, 2004.

[GMT06] E. Godard, Y. Métivier, and G. Tel. Termination detection of distributed tasks. Technical Report 1418–06, LaBRI, 2006.

[Mat87] F. Mattern. Algorithms for distributed termination detection. *Distributed computing*, 2(3):161–175, 1987.

[Maz97] A. Mazurkiewicz. Distributed enumeration. *Information Processing Letters*, 61(5):233–239, 1997.

[MMS06] M. Mavronicolas, L. Michael, and P. Spirakis. Computing on a partially eponymous ring. In *Proc. of OPODIS'06*, pages 380–394, 2006.

[MMW97] Y. Métivier, A. Muscholl, and P.-A. Wacrenier. About the local detection of termination of local computations in graphs. In *Proc. of SIROCCO'97*, pages 188–200. Carleton Scientific, 1997.

[SSP85] B. Szymanski, Y. Shy, and N. Prywes. Synchronized distributed termination. *IEEE Transactions on software engineering*, 11(10):1136–1140, 1985.

[Tel00] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 2000.

[YK96a] M. Yamashita and T. Kameda. Computing functions on asynchronous anonymous networks. *Math. Systems Theory*, 29(4):331–356, 1996.

[YK96b] M. Yamashita and T. Kameda. Computing on anonymous networks: Part I - characterizing the solvable cases. *IEEE Transactions on parallel and distributed systems*, 7(1):69–89, 1996.