Decision Power of Weak Asynchronous Models of Distributed Computing*

Philipp Czerner ⓘ, Roland Guttenberg ⓘ,
Martin Helfrich ⓘ, Javier Esparza ⓘ

{czerner, guttenbe, helfrich, esparza}@in.tum.de
Department of Informatics, TU München, Germany

August 21, 2021

Esparza and Reiter have recently conducted a systematic comparative study of models of distributed computing consisting of a network of identical finite-state automata that cooperate to decide if the underlying graph of the network satisfies a given property. The study classifies models according to four criteria, and shows that twenty-four initially possible combinations collapse into seven equivalence classes with respect to their decision power, i.e. the properties that the automata of each class can decide. However, Esparza and Reiter only show (proper) inclusions between the classes, and so do not characterise their decision power. In this paper we do so for *labelling* properties, i.e. properties that depend only on the labels of the nodes, but not on the structure of the graph. In particular, majority (whether more nodes carry label a than b) is a labelling property. Our results show that only one of the seven equivalence classes identified by Esparza and Reiter can decide majority for arbitrary networks. We then study the expressive power of the classes on bounded-degree networks, and show that three classes can. In particular, we present an algorithm for majority that works for all bounded-degree networks under adversarial schedulers, i.e. even if the scheduler must only satisfy that every node makes a move infinitely often, and prove that no such algorithm can work for arbitrary networks.

1

1. Introduction

A common feature of networks of natural or artificial devices, like molecules, cells, microorganisms, or nano-robots, is that agents have very limited computational power and no identities. Traditional distributed computing models are often inadequate to study the power and efficiency of these networks, which has led to a large variety of new models, including population protocols [4, 3], chemical reaction networks [29], networked finite state machines [15], the weak models of distributed computing of [20], and the beeping model [14, 1] (see e.g. [17, 27] for surveys and other models).

These new models share several characteristics [15]: the network can have an arbitrary topology; all nodes run the same protocol; each node has a finite number of states, independent of the size of the network or its topology; state changes only depend on the states of a bounded number of neighbours; nodes do not know their neighbours, in the sense of [2]. Unfortunately, despite such substantial common ground, the models still exhibit much variability. In [16] Esparza and Reiter have recently identified four fundamental criteria according to which they diverge:

- *Detection.* In some models, nodes can only detect the *existence* of neighbours in a certain state, e.g., [1, 20], while in others they can *count* their number up to a fixed threshold, e.g., [15, 20].

- *Acceptance.* Some models compute by *stable consensus*, requiring all nodes to eventually agree on the outcome of the computation, e.g. [4, 3, 29]; others require the nodes to produce an output and halt, e.g. [20, 22].

- *Selection.* Some models allow for *liberal selection*: at each moment, an arbitrary subset of nodes is selected to take a step [15, 28]. *Exclusive* models (also called *interleaving* models) select exactly one node (or one pair of neighbouring nodes) [4, 3, 29]. *Synchronous* models select all nodes at each step e.g.,[20] or classical synchronous networks [24].

- *Fairness.* Some models assume that selections are *adversarial*, only satisfying the minimal requirement that each node is selected infinitely often [18, 23]. Others assume *stochastic* or *pseudo-stochastic* selection (meaning that selections satisfy a fairness assumption capturing the main features of a stochastic selection) [4, 3, 29]. In this case, the selection scheduler is a source of randomness that can be tapped by the nodes to ensure e.g. that eventually all neighbours of a node will be in different states.

In [16], Esparza and Reiter initiated a comparative study of the computational power of these models. They introduced *distributed automata*, a generic formalism able to capture all combinations of the features above. A distributed automaton consists of a set of rules that tell the nodes of a labelled graph how to change their state depending on the states of their neighbours. Intuitively, the automaton describes an algorithm that allows the nodes to decide whether the graph satisfies a given property. The decision power of a class of automata is the set of graph properties they can decide, for example whether the graph contains more red nodes than blue nodes (the *majority* property), or
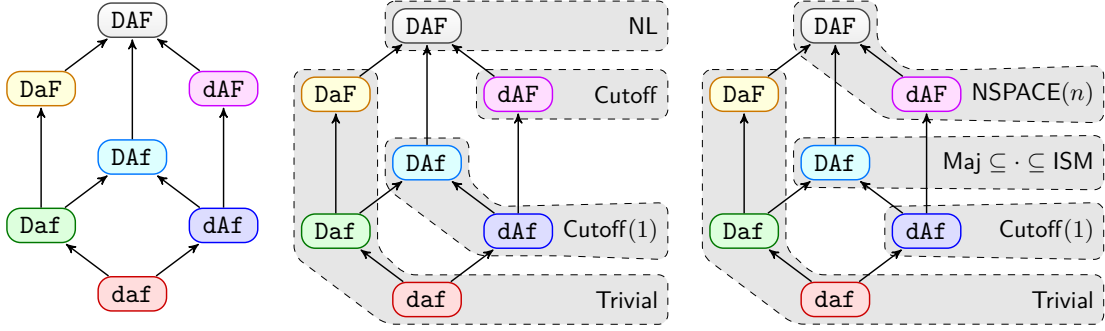
Figure 1: The seven distributed automata models of [16]; their decision power w.r.t. labelling predicates for arbitrary networks, and for bounded-degree networks. ISM stands for *invariant under scalar multiplication*. The other complexity classes are defined in Section 5.

whether the graph is a cycle. The main result of [16] was that the twenty-four classes obtained by combining the features above collapse into only seven equivalence classes w.r.t. their decision power. The collapse is a consequence of a fundamental result: the selection criterion does not affect the decision power. That is, the liberal, exclusive, or synchronous versions of a class with the same choices in the detection, acceptance, and fairness categories, have the same decision power. The seven equivalence classes are shown on the left of Figure 1, where `D` and `d` denote detection with and without the ability to count; `A` and `a` denote acceptance by stable consensus and by halting; and `F` and `f` denote pseudo-stochastic and adversarial fairness constraints. So, for example, `DAf` corresponds to the class of distributed automata in which agents can count, acceptance is by stable consensus, and selections are adversarial. (As mentioned above, the selection component is irrelevant, and one can assume for example that all classes have exclusive selection.) Intuitively, the capital letter corresponds to the option leading to higher decision power.

The results of [16] only prove inclusions between classes and separations, but give no information on which properties can be decided by each class, an information available e.g. for multiple variants of population protocols [6, 3, 7, 11, 19, 25]. In this paper, we characterise the decision power of all classes of [16] w.r.t. *labelling* properties, i.e. properties that depend only on the labels of the nodes. Formally, given a labelled graph G over a finite set Λ of labels, let $L_G \colon \Lambda \to \mathbb{N}$ be the *label count* of G that assigns to each label the number of nodes carrying it. A labelling property is a set $\mathcal{L}$ of label counts. A graph G satisfies $\mathcal{L}$ if $L_G \in \mathcal{L}$, and a distributed automaton decides $\mathcal{L}$ if it recognises exactly the graphs that satisfy $\mathcal{L}$. For example, the majority property is a labelling property, while the property of being a cycle is not.

Our first collection of results is shown in the middle of Figure 1. We prove that all classes with halting acceptance can only decide the trivial labelling properties $\emptyset$ and $\mathbb{N}^\Lambda$. More surprisingly, we further prove that the computational power of `DAf`, `dAf`, and `dAF` is very limited. Given a labelled graph G and a number K, let $\lceil L_G \rceil_K$ be the result of

substituting K for every component of L_G larger than K. The classes `DAf`, `dAf` can decide a property $\mathcal{L}$ iff membership of L_G in $\mathcal{L}$ depends only on $\lceil L_G \rceil_1$, and `dAF` iff membership depends only on $\lceil L_G \rceil_K$ for some $K \geq 1$. In particular, none of these classes can decide majority. Finally, moving to the top class `DAF` causes a large increase in expressive power: `DAF` can decide exactly the labelling properties in the complexity class NL, i.e. the properties $\mathcal{L}$ such that a nondeterministic Turing machine can decide membership of L_G in $\mathcal{L}$ using logarithmic space in the number of nodes of G. In particular, `DAF`-automata can decide majority, or whether the graph has a prime number of nodes.

In the last part of the paper, we obtain our second and most interesting collection of results. Molecules, cells, or microorganisms typically have short-range communication mechanisms, which puts an upper bound on their number of communication partners. So we re-evaluate the decision power of the classes for bounded-degree networks, as also done in [3] for population protocols on graphs. Intuitively, nodes know that they have at most k neighbours for some fixed number k, and can exploit this fact to decide more properties. Our results are shown on the right of Figure 1. Both `DAF` and `dAF` boost their expressive power to $\mathsf{NSPACE}(n)$, where n is the number of nodes of the graph. This is the theoretical upper limit since each node has a constant number of bits of memory. Further, the class `DAf` becomes very interesting. While we are not yet able to completely characterise its expressive power, we show that it can only decide properties *invariant under scalar multiplication* (ISM), i.e. labelling properties $\mathcal{L}$ such that $L_G \in \mathcal{L}$ iff $\lambda \cdot L_G \in \mathcal{L}$ for every $\lambda \in \mathbb{N}$, and that it can decide all properties satisfied by a graph G iff L_G is a solution to a system of homogeneous linear inequalities. In particular, `DAf` can decide majority, and we have the following surprising fact. If nodes have no information about the network, then they require stochastic-like selection to decide majority; however, if they know an upper bound on the number of their neighbours, they can decide majority even with adversarial selection. In particular, there is a synchronous majority algorithm for bounded-degree networks.

Related work. Decision power questions have also been studied in [20] for a model similar to `Daf`, and in [13] for a graph version of the mediated population protocol model [25]. The distinguishing feature of our work is the systematic study of the influence of a number of features on the decision power.

Further, there exist numerous results about the decision power of different classes of population protocols. Recall that agents of population protocols are indistinguishable and communicate by rendez-vous; this is equivalent to placing the agents in a clique and selecting an edge at every step. Angluin *et al.* show that standard population protocols compute exactly the semilinear predicates [6]. Extensions with absence detectors or cover-time services [26], consensus-detectors [7], or broadcasts [11] increase the power to NL (more precisely, in the case of [26] the power lies between L and NL). Our result $\mathsf{DAF} = \mathsf{NL}$ shows that these features can be replaced by a counting capability. Further, giving an upper bound on the number of neighbours increases the decision power to $\mathsf{NSPACE}(n)$, a class only reachable by standard population protocols (not on graphs) if agents have identities, or channels have memory [25, 19].

Structure of the paper.   Section 2 recalls the automata models and the results of [16]. Section 3 presents fundamental limitations of their decision power. Section 4 introduces a notion of simulating an automaton by another, and uses it to show that distributed automata with more powerful communication mechanisms can be simulated by standard automata. Section 5 combines the results of Sections 3 and 4 to characterise the decision power of the models of [16] on labelling properties (middle of Figure 1). Section 6 does the same for bounded-degree networks (right of Figure 1).

Due to the nature of this research, we need to state and prove many results. For the sake of brevity, each section concentrates on the most relevant result; all others are only stated, and their proofs are given in the appendix.

2. Preliminaries

Given sets X, Y, we denote by 2^X the power set of X, and by Y^X the set of functions $X \to Y$. We define a closed interval $[m:n] := \{i \in \mathbb{Z} : m \leq i \leq n\}$ and $[n] := [0:n]$, for any $m, n \in \mathbb{Z}$ such that $m \leq n$.

A *multiset* over a set X is an element of $\mathbb{N}^X$. Given a multiset $M \in \mathbb{N}^X$ and $\beta \in \mathbb{N}$, we let $\lceil M \rceil_\beta$ denote the multiset given by $\lceil M \rceil_\beta(x) := M(x)$ if $M(x) < \beta$ and $\lceil M \rceil_\beta(x) := \beta$ otherwise. We say that $\lceil M \rceil_\beta$ is the result of *cutting off* M at β, and call the function that assigns $\lceil M \rceil_\beta$ to M the *cutoff function* for β.

Let Λ be a finite set. A *(Λ-labelled, undirected) graph* is a triple $G = (V, E, \lambda)$, where V is a finite nonempty set of *nodes*, E is a set of undirected *edges* of the form $e = \{u, v\} \subseteq V$ such that $u \neq v$, and $\lambda : V \to \Lambda$ is a *labelling*.

Convention   Throughout the paper, all graphs are labelled, have at least three nodes, and are connected.

2.1. Distributed automata

Distributed automata [16] take a graph as input, and either accept or reject it. We first define distributed machines.

Distributed machines.   Let Λ be a finite set of *labels* and let $\beta \in \mathbb{N}_+$. A *(distributed) machine* with *input alphabet* Λ and *counting bound* β is a tuple $M = (Q, \delta_0, \delta, Y, N)$, where Q is a finite set of *states*, $\delta_0 : \Lambda \to Q$ is an *initialisation function*, $\delta : Q \times [\beta]^Q \to Q$ is a *transition function*, and $Y, N \subseteq Q$ are two disjoint sets of *accepting* and *rejecting* states, respectively. Intuitively, when M runs on a graph, each node v (or *agent*) with label γ is initially in state $\delta_0(\gamma)$ and uses δ to update its state, depending on the number of neighbours it has in each state; however v can only detect if it has $0, 1, ..., (\beta - 1)$, or at least β neighbours in a given state. We call β the *counting bound* of M.

Transitions given by δ are called *neighbourhood transitions*. We write $q, \mathcal{N} \mapsto q'$ for $\delta(q, \mathcal{N}) = q'$. If $q = q'$ the transition is *silent* and may not be explicitly specified in our constructions. Sometimes δ_0, Y, N are also irrelevant and not specified, and we just write $M = (Q, \delta)$.

Selections, schedules, configurations, runs, and acceptance. A *selection* of a graph $G = (V, E, \lambda)$ is a set $S \subseteq V$. A *schedule* is an infinite sequence of selections $\sigma = (S_0, S_1, S_2, ...) \in (2^V)^\omega$ such that for every $v \in V$, there exist infinitely many $t \geq 0$ such that $v \in S_t$. Intuitively, S_t is the set of nodes activated by the scheduler at time t, and schedules must activate every node infinitely often.

A *configuration* of $M = (Q, \delta_0, \delta, Y, N)$ on G is a mapping $C \colon V \to Q$. We let $N_v^C \colon Q \to [\beta]$ denote the *neighbourhood function* that assigns to each $q \in Q$ the number of neighbours of v in state q at configuration C, up to threshold β; in terms of the cutoff function, $N_v^C = \lceil M_v^C \rceil_\beta$, where $M_v^C(q) = |\{u : \{u, v\} \in E \wedge C(u) = q\}|$. The *successor configuration* of C via a selection S is the configuration $succ_\delta(C, S)$ obtained from C by letting all nodes in S evaluate δ simultaneously, and keeping the remaining nodes idle. Formally, $succ_\delta(C, S)(v) = \delta(C(v), N_v^C)$ if $v \in S$ and $succ_\delta(C, S)(v) = C(v)$ if $v \in V \setminus S$. We write $C \to C'$ if $C' = succ(C, S)$ for some selection S, and $\to^*$ for the reflexive and transitive closure of $\to$. Given a schedule $\sigma = (S_0, S_1, S_2, ...)$, the *run* of M on G scheduled by σ is the infinite sequence $(C_0, C_1, C_2, ...)$ of configurations defined inductively as follows: $C_0(v) = \delta_0(\lambda(v))$ for every node v, and $C_{t+1} = succ_\delta(C_t, S_t)$. We call C_0 the *initial configuration*. A configuration C is *accepting* if $C(v) \in Y$ for every $v \in V$, and *rejecting* if $C(v) \in N$ for every $v \in V$. A run $\rho = (C_0, C_1, C_2, ...)$ of M on G is *accepting* resp. *rejecting* if there is $t \in \mathbb{N}$ such that $C_{t'}$ is accepting resp. rejecting for every $t' \geq t$. This is called acceptance by *stable consensus* in [4].

Distributed automata. A *scheduler* is a pair $\Sigma = (s, f)$, where s is a *selection constraint* that assigns to every graph $G = (V, E, \lambda)$ a set $s(G) \subseteq 2^V$ of *permitted selections* such that every node $v \in V$ occurs in at least one selection $S \in s(G)$, and f is a *fairness constraint* that assigns to every graph G a set $f(G) \subseteq s(G)^\omega$ of *fair schedules* of G. We call the runs of a machine with schedules in $f(G)$ *fair runs* (with respect to Σ).

A *distributed automaton* is a pair $A = (M, \Sigma)$, where M is a machine and Σ is a scheduler satisfying the *consistency condition*: for every graph G, either all fair runs of M on G are accepting, or all fair runs of M on G are rejecting. Intuitively, whether M accepts or rejects G is independent of the scheduler's choices. A *accepts* G if some fair run of A on G is accepting, and *rejects* G otherwise. The language $L(A)$ of A is the set of graphs it recognises. The *property decided* by A is the predicate φ_A on graphs such that $\varphi_A(G)$ holds iff $G \in L(A)$. Two automata are equivalent if they decide the same property.

2.2. Classifying distributed automata.

Esparza and Reiter classify automata according to four criteria: detection capabilities, acceptance condition, selection, and fairness. The first two concern the distributed machine, and the last two the scheduler.

Detection. Machines with counting bound $\beta = 1$ or $\beta \geq 1$ are called *non-counting* or *counting*, respectively (abusing language, non-counting is considered a special case of counting).

Acceptance. A machine is *halting* if its transition function does not allow nodes to leave accepting or rejecting states, i.e. $\delta(q, P) = q$ for every $q \in Y \cup N$ and every $P \in [\beta]^Q$. Intuitively, a node that enters an accepting/rejecting state cannot change its mind later. Halting acceptance is a special case of acceptance by stable consensus.

Selection. A scheduler $\Sigma = (s, f)$ is *synchronous* if $s(G) = \{V\}$ for every $G = (V, E, \lambda)$ (at each step all nodes make a move); *exclusive* if $s(G) = \{\{v\} \mid v \in V\}$ (at each step exactly one node makes a move); and *liberal* if $s(G) = 2^V$ (at every step some set of nodes makes a move).

Fairness. A schedule $\sigma = (S_0, S_1, ...) \in s(G)^\omega$ of a graph G is *pseudo-stochastic* if for every finite sequence $(T_0, ..., T_n) \in s(G)^*$ there exist infinitely many $t \geq 0$ such that $(S_t, ..., S_{t+n}) = (T_0, ..., T_n)$. Loosely speaking, every possible finite sequence of selections is scheduled infinitely often. A scheduler $\Sigma = (s, f)$ is *adversarial* if for every graph G, the set $f(G)$ contains all schedules of $s(G)^\omega$ (i.e. we only require every node to be selected infinitely often), and *pseudo-stochastic* if it contains precisely the pseudo-stochastic schedules.

Whether or not a schedule σ of a graph $G = (V, E, \lambda)$ is pseudo-stochastic depends on $s(G)$. For example, if $s(G) = \{V\}$, i.e. if the only permitted selection is to select all nodes, then the synchronous schedule V^ω is pseudo-stochastic, but if $s(G) = 2^V$, i.e. if all selections are permitted, then it is not.

This classification yields 24 classes of automata (four classes of machines and six classes of schedulers). It was shown in [16] that the decision power of a class is independent of the selection type of the scheduler (liberal, exclusive, or synchronous). This leaves 8 classes, which we denote using the following scheme:

Detection	*Acceptance*	*Fairness*
d: non-counting	a: halting	f: adversarial scheduling
D: counting	A: stable consensus	F: pseudo-stochastic scheduling

Intuitively, the uppercase letter corresponds to the more powerful variant. Each class of automata is denoted by a string $xyz \in \{\mathtt{d}, \mathtt{D}\} \times \{\mathtt{a}, \mathtt{A}\} \times \{\mathtt{f}, \mathtt{F}\}$. Finally, it was shown in [16] that $\mathtt{daf}$ and $\mathtt{daF}$ have the same decision power, yielding the seven classes on the left of Figure 1.

In the rest of the paper, we generally assume that selection is exclusive (exactly one node is selected at each step). Since for synchronous automata there is only one permitted selection, adversarial and pseudo-stochastic scheduling coincide, and we therefore denote synchronous classes by strings $xy\$$; for example, we write $\mathtt{DA\$}$.

3. Limitations

Our lower bounds on the decision power of the seven classes follow from several lemmata proving limitations of their *discriminating* power, i.e. of their ability to distinguish two

graphs by accepting the one and rejecting the other. We present four limitations. We state the first three, and prove the last one, a non-trivial limitation of `dAF`-automata. Recall that φ_A denotes the property decided by the automaton A.

Automata with halting acceptance cannot discriminate cyclic graphs. Automata with halting acceptance necessarily accept all graphs containing a cycle, or reject all graphs containing a cycle. Intuitively, given two graphs G and H with cycles, if one is accepted and the other rejected, one can construct a larger graph in which some nodes behave as if they were in G, others as if they were in H. This makes some nodes accept and others reject, contradicting that for every graph the automaton accepts or rejects.

Lemma 3.1. *Let A be a `DaF`-automaton. For all graphs G and H containing a cycle, $\varphi_A(G) = \varphi_A(H)$.*

Automata with adversarial selection cannot discriminate a graph and its covering. Given two graphs $G = (V_G, E_G, \lambda_G)$ and $H = (V_H, E_H, \lambda_H)$, we say that H *covers* G if there is a *covering map* $f\colon V_H \to V_G$, i.e. a surjection that preserves labels and neighbourhoods by mapping the neighbourhood of each v in H bijectively onto the neighbourhood of $f(v)$ in G. Automata with adversarial selection cannot discriminate a graph from another one covering it. Intuitively, if H covers G then a node u of H and the node $f(u)$ of G visit the same sequence of states in the synchronous runs of A on G and H. Since these runs are fair for adversarial selection, both nodes accept, or both reject.

Lemma 3.2. *Let A be a `DAf`-automaton. For all graphs G and H, if H is a covering of G, then $\varphi_A(G) = \varphi_A(H)$.*

Let $L_G\colon \Lambda \to \mathbb{N}$ assign to each label $\ell \in \Lambda$ the number of nodes $v \in V$ such that $\lambda(v) = \ell$. We call L_G the *label count* of G. Recall that a labelling property depends only on the label count of a graph, not on its structure. Based on the existence of a λ-fold covering graph for every G and $\lambda \in \mathbb{N}$, we immediately get the following.

Corollary 3.3. *Let A be a `DAf`-automaton deciding a labelling property. For all graphs G and H, if $L_H = \lambda L_G$ for some $\lambda \in \mathbb{N}_{>0}$, then $\varphi_A(G) = \varphi_A(H)$. This also holds when restricting to k-degree-bounded graphs.*

Automata with adversarial selection and non-counting automata cannot discriminate beyond a cutoff. Our final results show that for every `DAf`- or `dAF`-automaton deciding a labelling property there is a number K such that whether the automaton accepts a graph G or not depends only on $\lceil L_G \rceil_K$, and not on the "complete" label count L_G. In such a case we say that the property admits a cutoff. For `DAf`-automata, the cutoff K is simply $\beta + 1$, where β is the counting bound.

Lemma 3.4. *Let A be a `DAf`-automaton with counting bound β that decides a labelling property. For all graphs G and H, if $\lceil L_G \rceil_{\beta+1} = \lceil L_H \rceil_{\beta+1}$ then $\varphi_A(G) = \varphi_A(H)$, i.e. φ_A admits a cutoff.*

The proof that `dAF`-automata also cannot discriminate beyond a cut-off is more involved, and the cutoff value K is a complex function of the automaton. The proof technique is similar to that of Theorem 39 of [6].

Lemma 3.5. *Let A be a `dAF`-automaton that decides a labelling property. There exists $K \geq 0$ such that for every graph G and H, if $\lceil L_G \rceil_K = \lceil L_H \rceil_K$ then $\varphi_A(G) = \varphi_A(H)$, i.e. φ_A admits a cutoff.*

Proof (sketch). Let A be a `dAF`-automaton, and let Q be its set of states. In this proof we consider the class of *star graphs*. A star is a graph in which a node called the *centre* is connected to an arbitrary number of nodes called the *leaves*, and no other edges exist. Importantly, for every graph G, there is a star G' with the same label count. We consider labelling properties (which do not depend on the graph), so if the property has a cutoff for star graphs, then the property has a cutoff in general. A configuration of a star graph G is completely determined by the state of the centre and the number of leaves in each state. So in the rest of the proof we assume that such a configuration is a pair $C = (C_{\text{ctr}}, C_{\text{sc}})$, where C_{ctr} denotes the state of the centre of G, and C_{sc} is the *state count* of C, i.e. the mapping that assigns to each $q \in Q$ the number $C_{\text{sc}}(q)$ of leaves of G that are in state q at C. We denote the *cutoff* of C at a number m as $\lceil C \rceil_m := (C_{\text{ctr}}, \lceil C_{\text{sc}} \rceil_m)$.

Given a configuration C of A, recall that C is rejecting if all nodes have rejecting states. We say that C is *stably rejecting* if C can only reach configurations which are rejecting. Given an initial configuration C_0, it is clear that A must reject if it can reach a stably rejecting configuration C from A. Conversely, if it cannot reach such a C, then A will not reject C_0, as there is a fair run starting at C_0 which contains infinitely many configurations that are not rejecting.

In the appendix we now use Dickson's Lemma to show that there is a constant m s.t. a configuration C of A on a star is stably rejecting iff $\lceil C \rceil_m$ is. For this it is crucial that for stars stable rejection is *downwards closed* in the following sense: if such a C is stably rejecting and has at least two leaves in a state q, then the configuration C' that results from removing one of these leaves is still stably rejecting.

Now, let $C = (C_{\text{ctr}}, C_{\text{sc}})$ denote a configuration of A on a star $G = (V, E)$, and let q denote a state with $C_{\text{sc}}(q) \geq |Q|(m-1) + 1$. We will show: if A rejects C then it must also reject the configuration $C' = (C'_{\text{ctr}}, C'_{\text{sc}})$ which results from adding a leaf v_{new} in state q to G, i.e. $C'_{\text{ctr}} := C_{\text{ctr}}$, $C'_{\text{sc}}(q) := C_{\text{sc}}(q) + 1$, and $C'_{\text{sc}}(r) := C_{\text{sc}}(r)$ for states $r \neq q$.

We know that A rejects C, so there is some stably rejecting configuration D reachable from C. Our goal is to construct a configuration D' reachable from C' which fulfils $\lceil D \rceil_m = \lceil D' \rceil_m$, implying that D' would also be stably rejecting. For this, let $S \subseteq V$ denote the leaves of G which are in state q in C. There are $|Q|$ states and $(m-1)|Q| + 1$ nodes in S, so by the pigeonhole principle there is a state $r \in Q$ s.t. in configuration D at least m nodes in S are in state r. Let v_{old} denote one of these nodes.

To get D', we construct a run starting from C', where v_{new} behaves exactly as v_{old}, until D' is reached. Afterwards, the nodes may diverge because of the pseudo-stochastic scheduler. However, this does not matter as D' is stably rejecting.

Let $\rho = (v_1, ..., v_\ell) \in V^*$ denote a sequence of selections for A to go from C to D. We construct the sequence $\sigma \in V^*$ by inserting a selection of v_{new} after every selection of

v_{old}, and define D' as the configuration which A reaches after executing σ from C'. We claim that D' is the same as D, apart from having an additional leaf in the same state as v_{old}.

This follows from a simple induction: v_{old} and v_{new} start in the same state and see only the root node. As they are always selected subsequently, they will remain in the same state as each other. For the centre we use the property that A cannot count: it cannot differentiate between seeing just v_{old}, or seeing an additional node in the same state. We remark that G being a star is crucial for this argument, which does not extend to e.g. cliques.

To summarise, we have shown that for every rejected star G and state q with $L_G(q) \geq (m-1)|Q| + 2$ (note the centre), the input H obtained by adding a node with label q to G is still rejected. An analogous argument shows that the same holds for acceptance, and by induction we find that $K := m(|Q| - 1) + 2$ is a valid cutoff. $\qquad\square$

Since the majority property does not admit a cutoff, in particular we obtain:

Corollary 3.6. *No `DAf`- or `dAF`-automaton can decide majority.*

4. Extensions

We introduce automata with more powerful communication mechanisms, and show that they can be simulated by standard automata with only neighbourhood transitions. We first present our notion of simulation (Definitions 4.1-4.3), and then in Sections 4.1-4.3 extend automata with weak versions of broadcast (a node sends a message to all other nodes) and absence detection (a node checks globally if there exists a node occupying a given state), and with communication by rendezvous transitions (two neighbours change state simultaneously).

Definition 4.1. Let $G = (V, E, \lambda)$ be a labelled graph and let Q, Q' denote sets of states, with $Q \subseteq Q'$. For configurations $C_1, C_2 : V \to Q'$ we define the relation $\sim_Q$ as $C_1 \sim_Q C_2$ iff $C_1(v) = C_2(v)$ for all v with $C_1(v) \in Q$ and $C_2(v) \in Q$. Let π, π' denote runs over states Q and Q', respectively. We say that π' is an *extension* of π if there exists a monotonically increasing $g : \mathbb{N} \to \mathbb{N}$ with $\pi(i) = \pi'(g(i))$ for all $i \in \mathbb{N}$, and $\pi'(j) \sim_Q \pi'(g(i))$ or $\pi'(j) \sim_Q \pi'(g(i+1))$ for all $g(i) \leq j \leq g(i+1)$.

To implement complicated transitions in an automaton without extensions, we decompose them into multiple standard neighbourhood transitions. Instead of performing, say, a broadcast atomically in one step, agents perform a sequence of neighbourhood transitions, moving into intermediate states in the process. As mentioned in Section 2, the results of [16] allow us to use liberal or exclusive selection without changing the decision power. We assume that selection is exclusive, unless stated otherwise.

Definition 4.2. Let $G = (V, E, \lambda)$ be a labelled graph. Let π, π' denote runs of an automaton induced by schedules $s, s' \in V^\omega$, respectively. Let I, I' denote the set of indices where π or π', respectively, execute non-silent transitions, i.e. $I := \{i : \pi_i \neq \pi_{i+1}\}$.

We say that π' is a *reordering* of π if there exists a bijection $f\colon I \to I'$ s.t. $s(i) = s'(f(i))$ for all $i \in \mathbb{N}$, and $f(i) < f(j)$ for all $i < j$ where the nodes $s(i)$ and $s(j)$ are adjacent or identical. If that is the case, we also write $\pi_f := \pi'$ for the reordering induced by f.

While an extension of a run can execute a single complicated transition in many steps instead of atomically, steps of different transitions, or of different phases of a transition, should not "interfere". Ideally all the neighbourhood transitions simulating, say, a broadcast, should be executed before any of the transitions simulating the next one. However, in distributed automata this cannot be guaranteed. This is where we make use of reorderings: We will guarantee that every run can be reordered into an equivalent run in which transitions do not "interfere". We will only allow reordering of nodes that are not adjacent, thus ensuring that the reordered run yields the same answer as the original one.

Lastly, we now introduce a generic model encompassing all of our extended automata, which allows us to define our notion of simulation for all extensions simultaneously.

Definition 4.3. We say that $P = (Q, \mathrm{Run}, \delta_0, Y, N)$ is a *generalised graph protocol*, where Q are states, δ_0, Y, N are initialisation function, accepting states and rejecting states, respectively, and Run is a function mapping every labelled graph $G = (V, E, \lambda)$ over a given alphabet Λ to a subset $\mathrm{Run}(G) \subseteq (Q^V)^\omega$ of fair runs. We define accepting/rejecting runs and the statement "P decides a predicate φ" analogously to distributed automata. Further, let P' be an automaton with states $Q' \supseteq Q$. We say that P' *simulates* P, if for every fair run π' of P' there is a reordering π'_f of π' and a fair run $\pi \in \mathrm{Run}$ of P, s.t. π'_f is an extension of π. If P' simulates P, we refer to the states in $Q' \setminus Q$ as *intermediate* states.

We will apply this general definition to simulate broadcast, absence-detection, and rendezvous transitions by automata with only neighbourhood transitions. In the appendix we show that if π' is a reordering of π and v is the node satisfying $s(i) = v = s'(f(i))$, then the neighbourhood of v in π at time i and the neighbourhood of v in π' at time $f(i)$ coincide. Furthermore, we show that an automaton P' that simulates P can be easily transformed into an automaton P'' that also simulates P *and* is equivalent to P, i.e., decides the same property as P.

Lemma 4.4. *Let $P = (Q, \mathrm{Run}, \delta_0, Y, N)$ denote a generalised graph protocol deciding a predicate φ, and P' an automaton simulating P. Then there is an automaton P'' simulating P which also decides φ.*

The automaton P'' constructed in the proof of this lemma is basically P', except that nodes remember the last state $q \in Q$ they visited, in addition to their current state $q' \in Q'$. This allows us to define the accepting/rejecting states of Q'' as the pairs $(q', q) \in Q' \times Q$ such that q is an accepting/rejecting state of P.

Notation. Because of Lemma 4.4, in simulation proofs we often leave out the accepting and rejecting states of generalised graph protocols and automata.

4.1. Weak Broadcasts

Intuitively, a broadcast transition $q \mapsto r, f$ models that an agent in state q, called the *initiating* agent or *initiator*, sends a signal to the other agents, and moves to state r; the other agents react to the signal by moving to new states, determined by their current state and by f, a mapping from states to states. Broadcasts are weak, meaning that multiple broadcasts can occur at the same time. When this happens, all initiators send their signals and move to their new states, and for every other agent the scheduler decides which signal it receives and reacts to. It is only guaranteed that every non-initiator receives exactly one signal, and that this signal has been sent.

Definition 4.5. A *distributed machine with weak broadcasts* is defined as a tuple $M = (Q, \delta_0, \delta, Q_B, B, Y, N)$, where $(Q, \delta_0, \delta, Y, N)$ is a distributed machine, $Q_B \subseteq Q$ is a set of *broadcast-initiating* states, and $B \colon Q_B \to Q \times Q^Q$ describes a set of *weak broadcast transitions*, one for each state of Q_B. In particular, B maps a state q to a pair (q', f), where q' is a state and $f \colon Q \to Q$ is a *response function*. We write broadcast transitions as $q \mapsto r, f$, where f is usually given as a set $\{r \mapsto f(r) : r \in Q\}$. (Mappings $r \mapsto r$, and silent transitions $q \mapsto q, \text{id}$, id being the identity function, may be omitted.) Given a configuration C and a selection $S \subseteq V$ of initiators such that $C(v) \in Q_B$ for every $v \in S$, the machine can move to any configuration C' satisfying the following conditions:

- If $v \in S$, then $C'(v) = q'$, where q' is the state such that $B(C(v)) = (q', f)$.
- If $v \notin S$, then $C'(v) = f(C(v))$, where $B(C(u)) = (q', f)$ for some $u \in S$, i.e., f is the response function of an initiator u.

A valid selection is a nonempty independent set of nodes of V. The set of valid selections is denoted $\mathcal{I}$. A *schedule* of M is a sequence $\sigma \in (\{n, b\} \times \mathcal{I})^\omega$; intuitively, $\sigma(i) = (n, S)$ means that at time i the scheduler asks the agents of S to perform a neighbourhood transition, and $\sigma(i) = (b, S)$ that it asks the agents of S to initiate weak broadcasts. Given a schedule σ, we generate a *run* $\pi = (C_0, C_1, ...)$ as follows. For each step $i \geq 1$ either $\sigma(i) = (n, S)$ for $S \subseteq V$ and we execute a neighbourhood transition for $S' := S \setminus C_i^{-1}(Q_B)$, or $\sigma(i) = (b, S)$ for $S \subseteq V$ and we execute a weak broadcast transition on $S' := S \cap C_i^{-1}(Q_B)$. (In either case, if S' is empty we set $C_{i+1} := C_i$ instead.)

A schedule σ is *adversarial* if there are infinitely many i with $\sigma(i) = (b, S)$ for some S, or for all $v \in V$ there are infinitely many i with $\sigma(i) = (n, S)$ and $v \in S$. It is *pseudo-stochastic*, if every finite sequence of selections $w \in (\{n, b\} \times \mathcal{I})^*$ appears infinitely often in σ. Given $xyz \in \{\text{d}, \text{D}\} \times \{\text{a}, \text{A}\} \times \{\text{f}, \text{F}\}$, an *xyz-automaton with weak broadcasts* is a tuple (M, Σ) defined as for an *xyz*-automaton, except that M is a distributed machine with weak broadcasts. In particular, we extend the definitions of fair runs, consensuses, and acceptance to automata with weak broadcasts.

A *strong broadcast protocol* is a tuple $P = (Q, \delta_0, B, Y, N)$ that is defined analogously to a `dAF`-automaton with weak broadcasts $(Q, \delta_0, \emptyset, Q, B, Y, N)$, except that the set of valid selections is $\mathcal{I} := \{\{v\} : v \in V\}$. In other words, only one agent can broadcast at a given time. This model corresponds to the broadcast consensus protocols of [11].[1]

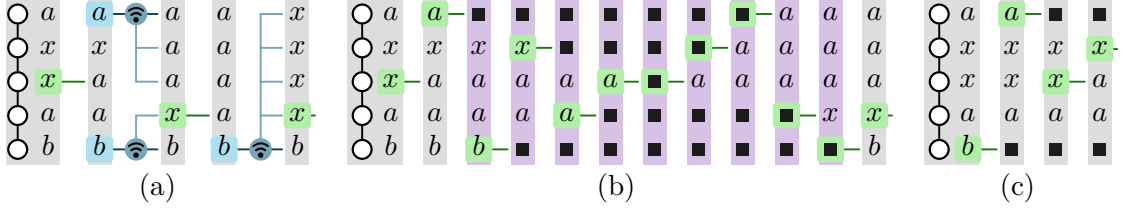[1]The protocols of [11] also contain rendez-vous transitions, but they can be removed without affecting

Figure 2: (a) Prefix of a run of the automaton of Example 4.6. on a line with exactly five nodes. (b) An extension of the same run, where ■ denotes intermediate states; only the first 12 steps are shown. (c) A reordering of the run of (b); only four steps are shown.

Additionally, to simplify our proofs we assume that all selections (n, S) satisfy $|S| = 1$, i.e. at each step the scheduler selects one single agent to execute a neighbourhood transition. Observe that we can assume $|S| = 1$ without loss of generality. Indeed, since S is an independent set by definition, it only contains non-adjacent nodes, and so after the agents of S execute a neighbourhood transition, be it simultaneously or sequentially, they reach the same states.

Example 4.6. Consider a `dAF`-automaton P with states $\{a, b, x\}$, a neighbourhood transition $x, N \mapsto a$ for every neighbourhood $N \colon Q \to [1]$ with $N(a) > 0$ (i.e. an agent moves from x to a if it has at least one neighbour in a), and weak-broadcast transitions

$$a \mapsto a, \{x \mapsto a\} \qquad \text{and} \qquad b \mapsto b, \{b \mapsto a, a \mapsto x\} \ .$$

Figure 2 shows sample runs of P on the graph consisting of a line with five nodes. Note that the simultaneous broadcasts at both ends of the line are executed simultaneously, and are received by three and two nodes, respectively. However, the next (and last) broadcast, which is initiated by the bottom node, reaches all nodes. The reordering depicted in (c) shows the interleaving of two different transitions: while the two ends have already initiated broadcasts, the information has not reached the middle node, and it can execute a neighbourhood transition.

Of course, our model of weak broadcasts would be of limited use if we were not able to simulate it. For this we use a construction similar to the three-phase protocol of Awerbuch's alpha-synchroniser [8]. Instead of simply using it to synchronise, we will propagate additional information, allowing the agents to perform the local update necessary to execute the broadcast.

Lemma 4.7. *Every automaton with weak broadcasts is simulated by some automaton of the same class without weak broadcasts.*

Proof (sketch). Let $P = (Q, \delta_0, \delta, Q_B, B)$ denote an automaton with weak broadcasts. We define an automaton $P' = (Q', \delta'_0, \delta')$ simulating P. The automaton P' has three phases, called 0, 1, and 2. A node moves to the next phase (modulo 3) only if every neighbour

---

expressive power.

13

is in the same phase or in the next. The states of P' are $Q' := Q \cup Q \times \{1,2\} \times Q^Q$. Intuitively, an agent of P' in state $q \in Q$ is in phase 0, and simulates an agent of P in state q; an agent of P' in state $(q, i, f) \in Q \times \{1,2\} \times Q^Q$ is in phase i, and simulates an agent executing P in state q, and initiating or responding to a broadcast with response function f.

Let β denote the counting bound of P. To specify the transitions, for a neighbourhood $N : Q' \to [\beta]$ we write $N[i] := \sum_{q,f} N((q, i, f))$ for $i \in \{1, 2\}$ and $N[0] := \sum_{q \in Q} N(q)$ to denote the number of adjacent agents in a particular phase, and choose a function $g(N) \in Q^Q \cup \{\square\}$ s.t. $g(N) = f \neq \square$ implies $N((q, 1, f)) > 0$, and $g(N) = \square$ implies $N[1] = 0$. The function g is used to select which broadcast to execute, if there are multiple possibilities. We define the following transitions for δ', for all states $q \in Q$ and neighbourhoods $N : Q \to [\beta]$.

$$
\begin{align}
q, N &\mapsto \delta(q, N) & &\text{if } q \notin Q_B \text{ and } N[0] = |N| \tag{1} \\
q, N &\mapsto (q', 1, f) & &\text{if } q \in Q_B \text{ and } N[0] = |N|, \text{ with } (q', f) := B(q) \tag{2} \\
q, N &\mapsto (f(q), 1, f) & &\text{if } g(N) = f \neq \square \tag{3} \\
(q, 1, f), N &\mapsto (q, 2, f) & &\text{if } N[0] = 0 \tag{4} \\
(q, 2, f), N &\mapsto q & &\text{if } N[1] = 0 \tag{5}
\end{align}
$$

If all neighbours are in phase 0, the agent either executes a neighbourhood transition via (1) or it initiates the broadcast in (2), depending on the state of the agent. For the latter, the agent immediately performs the local update. Once there is a phase 1 neighbour, the agent instead executes the broadcast of one of its neighbours via (3) (if there are multiple, g is used to select one). Note that (2) and (3) are indeed well-defined, as $N[0] = |N|$ holds iff $g(N) = \square$. Finally, transitions (4) and (5) move agents to the next phase, once all of their neighbours are in the same or the next phase. $\qquad\square$

4.2. Weak Absence Detection

Absence detection, introduced in [26], enables agents to determine the *support* of the current configuration, defined as the set of states currently populated by at least one agent. More precisely, an agent that executes an absence-detection transition moves to a new state that depends on the current support. We consider a weaker mechanism where, as for weak broadcasts, multiple absence-detection transitions may occur at the same time. In this case, each agent executing an absence-detection transition moves according to the support of a *subset* of the agents. However, it is ensured that every agent belongs to at least one of these subsets.

While it is possible to define and implement a more general model involving absence-detection, we limit ourselves to a special case to simplify our proofs. In particular, we define a model in which scheduling is synchronous. Further, we implement a simulation only for graphs of bounded degree.

Definition 4.8. A *distributed machine with weak absence-detection* is defined as a tuple $(Q, \delta_0, \delta, Q_A, A, Y, N)$, where $(Q, \delta_0, \delta, Y, N)$ is a distributed machine, Q_A is a set

of *initiating* states or *initiators*, and $A\colon Q_A \times 2^Q \to Q$ a set of *(weak) absence-detection transitions*. Given a configuration C, a selection $S \subseteq V$ of initiators such that $C(v) \in Q_A$ for every $v \in S$, and a set $S_v \subseteq V$ for every $v \in S$ satisfying $v \in S_v$ and $\bigcup_{v \in S} S_v = V$, the machine can move to any configuration C' with $C'(v) := A(v, C(S_v))$ for $v \in S$ and $C'(v) := C(v)$ for $v \notin S$. (Notice that the S_v need not be pairwise disjoint.) We write $q, S \mapsto q'$ to denote that $A(q, S) = q'$ for $q \in Q_A$, $q' \in Q$ and $S \subseteq Q$.

We use the synchronous scheduler, so the only valid selection is V. A step at a configuration C is performed by having each agent execute a neighbourhood transition simultaneously, moving to C', followed by an absence-detection with $S := C^{-1}(S_A)$ as set of initiators, to go from C' to C''. If S is empty, the computation hangs, and we instead set $C'' := C$. A *DA$-automaton with (weak) absence-detection* is defined analogously to a `DA$`-automaton.

As for broadcasts, absence detection is implemented using a three phase protocol. To allow the information to propagate back, we use a distance labelling that effectively embeds a rooted tree for each initiating agent.

Lemma 4.9. *Every `DA$`-automaton with weak absence detection is simulated by some `DAf`-automaton, when restricted to bounded-degree graphs.*

4.3. Rendez-vous transitions

In rendez-vous transitions two neighbours interact and move to new states according to a joint transition function. They are the communication mechanism of population protocols [4]. In fact, population protocols on graphs have also been studied previously [3], and we use exactly the same model.

A rendez-vous transition $p, q \mapsto p', q'$ allows two neighbouring nodes u and v in states p and q to interact and change their states to p' and q', respectively. Like neighbourhood transitions, rendez-vous transitions are local, i.e., they only involve adjacent nodes. They are useful to model transactions such as transferring a token from one node to another. A population protocol on graphs, or graph population protocol, is a pair (Q, δ) where q is a set of states and $\delta\colon Q^2 \to Q^2$ is a set of rendez-vous transitions, and $p, q \mapsto p', q'$ denotes $\delta(p, q) = (p', q')$. The formal definition can be found in the appendix.

Lemma 4.10. *Every graph population protocol is simulated by some `DAF`-automaton.*

5. Unrestricted Communication Graphs

We prove the characterisation of the decision power of the different classes as presented in the introduction. The classes are defined as follows. For a labelling property $\varphi\colon \mathbb{N}^\Lambda \to \{0, 1\}$ we have

- $\varphi \in \mathsf{Trivial}$ iff φ is either always true or always false,
- $\varphi \in \mathsf{Cutoff}(1)$ iff $\varphi(L) = \varphi(\lceil L \rceil_1)$ for all multisets $L \in \mathbb{N}^\Lambda$,
- $\varphi \in \mathsf{Cutoff}$ iff there exists a $K \in \mathbb{N}$ s.t. $\varphi(L) = \varphi(\lceil L \rceil_K)$ for all $L \in \mathbb{N}^\Lambda$, and

- $\varphi \in \mathsf{NL}$ iff φ is decidable by a non-deterministic log-space Turing machine.

The proof proceeds in the following steps:

1. `DaF` and therefore all automata-classes with weak acceptance have an upper bound of Trivial and thus decide exactly Trivial. This proof also works when restricted to degree-bounded graphs.

2. `DAf` and therefore also `dAf` can decide at most $\mathsf{Cutoff}(1)$.

3. `dAf` and therefore also `DAf` can decide at least $\mathsf{Cutoff}(1)$.

4. `dAF` can decide exactly Cutoff.

5. `DAF` can decide exactly the labelling propertis in NL.

In this section we sketch the hardest proof, the characterisation for `DAF`. All other proofs can be found in the appendix. We start with some conventions and notations.

Conventions and notations. When describing automata of a given class (possibly with weak broadcasts or weak absence detection) we specify only the machine; the scheduler is given implicitly by the fairness condition and selection criteria of the class. Further, when the initialisation function and the accepting/rejecting states are straightforward, which is usually the case, we only describe the sets of states and transitions. So, for example, we speak of the automaton (Q, δ), or the automaton with weak broadcasts (Q, δ, Q_B, B). We even write $(Q, \delta) + B$; in this case Q_B is implicitly given as the states of B initiating non-silent broadcasts, i.e. $Q_B := \{q : B(q) \neq (q, \mathrm{id})\}$.

Given an automaton P (possibly with weak broadcasts or absence detection) with set of states Q and a set Q', we let $P \times Q'$ denote the automaton with set of states $Q \times Q'$ whose transitions leave the Q' component of the state untouched. In other words, if a transition makes a node move from state (q, q') to state (p, p'), then $q' = p'$. The definition of the transitions is straightforward, and we omit it.

We often combine the two notations above. Given an automaton P', we write for example $P = P' \times Q' + B$ to denote the automaton with weak broadcasts obtained by first constructing $P' \times Q'$, and then adding the set B of weak broadcast transitions.

Lemma 5.1. *`DAF`-automata decide exactly the labelling properties in* NL*.*

Proof (sketch). First, we argue why `DAF`-automata can decide only labelling properties in NL. Let P be a `DAF`-automaton deciding a labelling property φ. We exhibit a log-space Turing machine that given a labelled graph $G = (V, E, \lambda)$ decides whether P accepts G. Since φ is a labelling property, $\varphi(G) = \varphi(\hat{G})$ for the unique clique $\hat{G}$ with set of nodes V and labelling λ. The Turing machine therefore ignores G and simulates M on $\hat{G}$. A configuration of $\hat{G}$ is completely characterized up to isomorphism by the *number* of agents in each state; in particular, it can be stored using logarithmic space. In [11, Proposition 4] it is shown that any class of automata whose configurations have this property, and whose step relation is in NL (i.e., there is a log-space Turing machine that on input (C, C') decides if the automaton can move from C to C'), can only decide

properties in NL. Since the step relation of DAF-automata on cliques is certainly in NL, the result follows.

Now we show the other direction. It is known that strong broadcast protocols decide exactly the predicates in NL [11, Theorem 15]. Therefore, it suffices to show that for every strong broadcast protocol there is an equivalent DAF-automaton. By Lemma 4.7 DAF-automata can simulate weak broadcasts, and so, loosely speaking, the task is to simulate strong broadcasts with weak ones.

Let $P = (Q, \delta, I, O)$ be a strong broadcast protocol. We start with a graph population protocol $P_{\text{token}} := (Q_{\text{token}}, \delta_{\text{token}})$, with states $Q_{\text{token}} := \{0, L, L', \bot\}$ and rendez-vous transitions δ_{token} given by

$$(L, L) \mapsto (0, \bot), \quad (0, L) \mapsto (L, 0), \quad (L, 0) \mapsto (L', 0) \qquad \langle\text{token}\rangle$$

Now we construct a DAF-automaton $P'_{\text{token}} = (Q'_{\text{token}}, \delta'_{\text{token}})$ simulating P_{token} using Lemma 4.10, and combine it with P by setting $P_{\text{step}} := P'_{\text{token}} \times Q + \langle\text{step}\rangle$, where $\langle\text{step}\rangle$ is a weak broadcast defined as

$$(L', q) \mapsto (L, q'), \{(t, r) \mapsto (t, f(r)) : (t, r) \in Q'_{\text{token}} \times Q\} \qquad \langle\text{step}\rangle$$

for each broadcast $q \mapsto q', f$ in δ. Finally, let $P'_{\text{step}} = (Q'_{\text{step}}, \delta'_{\text{step}})$ be a DAF-automaton simulating P_{step}, which exists by Lemma 4.7.

Intuitively, agents in states L, L' have a *token*. If we could ensure that initially there is only one token in L, L', then we would be done. Indeed, in this case at each moment only the agent with the token can move; if in L, it initiates a (simulated) rendez-vous transition, and if in L', a weak broadcast. Since no other agent is executing a weak broadcast at the same time, the weak broadcast is received by all agents, and has the same effect as a strong broadcast.

We cannot ensure that initially there is only one token, but if the computation starts with more than one, then two tokens eventually meet using transition $\langle\text{token}\rangle$ and an agent moves into the *error state* $\bot$. We design a mechanism to restart the computation after this occurs, now with fewer agents in state $(L, \cdot)$, guaranteeing that eventually the computation is restarted with only one token. For this we again add an additional component to each state and consider the protocol $P_{\text{reset}} := P'_{\text{step}} \times Q + \langle\text{reset}\rangle$, where $\langle\text{reset}\rangle$ are the following broadcast transitions, for each $q, q_0 \in Q$.

$$((\bot, q), q_0) \mapsto ((L, q_0), q_0), \{(r, r_0) \mapsto ((0, r_0), r_0) : r \in Q'_{\text{step}}, r_0 \in Q\} \qquad \langle\text{reset}\rangle$$

For P_{reset} we define the input mapping $I_{\text{reset}}(x) := ((L, I(x)), I(x))$ and the set of accepting states $O_{\text{reset}} := \{((r, q), q_0) : q \in O, q_0 \in Q, r \in \{0, L\}\}$. Using Lemma 4.7 (and Lemma 4.4) we get a DAF-Automaton equivalent to P_{reset}, so it suffices to show that P_{reset} is equivalent to P.

In the appendix, we show that a run of P_{reset} starting with more than one token will eventually reset and restart the computation with strictly fewer tokens, until only one token is left. After this moment, $\langle\text{reset}\rangle$ is never executed again, and so we are left with a run of P_{step}, which stabilises to a correct consensus. $\qquad\square$

6. Bounded-degree Communication Graphs

We characterise the decision power of the models when the degree of the input graphs is at most k for some constant $k \in \mathbb{N}$. Many results for the unrestricted set of graphs continue to hold, in particular Corollary 3.3, proving that `DAf`-automata can only compute properties invariant under scalar multiplication (called ISM in Figure 1), as well as the result that automata with halting acceptance can only decide trivial properties. The new results are:

1. For every $k \geq 3$ the expressive power of `dAf` is precisely Cutoff(1).

2. `DAF`- and `dAF`-automata decide exactly the labelling properties in NSPACE(n).

3. `DAf` can decide all homogeneous threshold predicates, in particular majority. This is a proper subset of ISM (the latter contains e.g. the divisibility predicate $\varphi(x, y) \Leftrightarrow x|y$), so there is a gap between our upper and lower bounds for `DAf`.

We describe the `DAf`-automata for homogeneous threshold predicates. All other proofs can be found in the appendix.

6.1. `DAf` decides all homogeneous threshold predicates on bounded-degree graphs

Let $\varphi : \mathbb{N}^l \to \{0, 1\}$, $\varphi(x_1, ..., x_l) \Leftrightarrow a_1 x_1 + ... + a_l x_l \geq 0$ denote an arbitrary homogeneous threshold predicate, with $a_1, ..., a_l \in \mathbb{Z}$, and let k denote the maximum degree of the communication graph.

Local Cancellation. We first define a protocol that performs local updates. Each agent stores a (possibly negative) integer contribution. If the absolute value of the contribution is large, then the agent will try to distribute the value among its neighbours. In particular, if a node v has contribution x with $x > k$, then it will "send" one unit to each of its neighbours with contribution $y \leq k$. Those neighbours increment their contribution by 1, while v decrements its contribution accordingly. (This happens analogously for $x < -k$, where -1 units are sent.) Agents may receive multiple updates in a single step, or may simultaneously send and receive updates.

We define a `DA$`-automaton with weak absence detection $P_{\text{cancel}} := (Q_{\text{cancel}}, \delta_{\text{cancel}}, \emptyset, \emptyset)$, but use only neighbourhood transitions for the moment. We use states $Q_{\text{cancel}} := \{-E, ..., E\}$. Here $E := \max\{|a_1|, ..., |a_l|\} \cup \{2k\}$ is the maximum contribution an agent must be able to store: any agent with contribution x s.t. $|x| \leq k$ may receive an increment or decrement from up to k neighbours, so $E \geq k + k$. The transitions δ_{cancel} are

$$
\begin{array}{lll}
x, N \mapsto x - N[-E, -k-1] + N[k+1, E] & \text{for } x = -k, ..., k & \\
x, N \mapsto x - N[-E, k] & \text{for } x = k+1, ..., E & \langle\text{cancel}\rangle \\
x, N \mapsto x + N[-k, E] & \text{for } x = -E, ..., -k-1 &
\end{array}
$$

Here we write $N[a, b] := \sum_{i=a}^{b} N(i)$ for the total number of adjacent agents with contribution in the interval $[a : b]$. As we use the synchronous scheduler, at each step all agents

make a move. It is thus easy to see that $\langle \mathsf{cancel} \rangle$ preserves the sum of all contributions $\sum_v C(v)$ for a configuration C, and that it does not increase $\sum_v |C(v)|$.

We can now show that the above protocol converges in the following sense:

Lemma 6.1. *Let $\pi = (C_0, C_1, ...)$ denote a run of P_{cancel} with $\sum_v C_0(v) < 0$. Then there exists $i \geq 0$ such that either all configurations $C_i, C_{i+1}, ...$ only have states in $\{-E, ..., -1\}$, or they only have states $\{-k, ..., k\}$.*

Convergence and Failure Detection.   The overall protocol waits until P_{cancel} converges, i.e. either all agents have "small" contributions, or all contributions are negative. In the latter case, we can safely reject the input, as the total sum of contributions is negative. In the former case we perform a broadcast, doubling all contributions. As we only double once all contributions are small, each agent can always store the new value. This idea of alternating cancelling and doubling phases has been used extensively in the population protocol literature [5, 9, 10, 21].

To detect whether P_{cancel} has already converged, and to perform the doubling, we elect a subset of agents as leaders. A "true" leader election, with only one leader at the end, is impossible due to weak fairness, but we can elect a "good enough" set of leaders: whenever two leaders disagree, we can eliminate one of them and restart the computation with a non-empty, proper subset of the original set of leaders.

We use weak absence-detection transitions to determine whether P_{cancel} has converged. Set $Q_L := \{0, L, L_{\mathrm{double}}, L_\square\}$ and let $(Q, \delta) := P_{\mathrm{cancel}} \times Q_L$. (Recall the notation from Section 5.) We define $P_{\mathrm{detect}} := (Q \cup \{\bot, \square\}, \delta, Q_{\mathrm{cancel}} \times \{L\}, A)$, where A are the following absence-detection transitions, for $x \in Q_{\mathrm{cancel}}, s \subseteq Q \cup \{\bot, \square\}$.

$$
\begin{aligned}
(x, L), s &\mapsto \bot && \text{if } \square \in s \\
(x, L), s &\mapsto (x, 0) && \text{if } \bot \in s \\
(x, L), s &\mapsto (x, L_{\mathrm{double}}) && \text{if } s \subseteq \{-k, ..., k\} \times \{0\} \\
(x, L), s &\mapsto (x, L_\square) && \text{if } s \subseteq \{-E, ..., -1\} \times \{0\}
\end{aligned}
\qquad \langle \mathsf{detect} \rangle
$$

Intuitively, $\bot$ and $Q_{\mathrm{cancel}} \times \{L, L_{\mathrm{double}}, L_\square\}$ are leader states, and $\square$ is the (only) rejecting state. State $\bot$ is an error state: an agent in that state will eventually restart the computation. Via Lemma 4.9 we get a `DAf`-automaton $P'_{\mathrm{detect}} = (Q'_{\mathrm{detect}}, \delta'_{\mathrm{detect}})$ simulating P_{detect}.

We want our broadcasts to interrupt any (simulated) absence-detection transitions of P'_{detect}, by moving agents in intermediate states $Q'_{\mathrm{detect}} \setminus Q_{\mathrm{detect}}$ to their last "good" state in Q_{detect}. To this end, we introduce the mapping $\mathsf{last} : Q'_{\mathrm{detect}} \to Q_{\mathrm{detect}}$, which fulfils $\mathsf{last}(C_i(v)) \in \{\mathsf{last}(C_{i-1}(v)), C_i(v)\}$ for all runs $\pi = C_0 C_1 ...$ of P'_{detect} and $i > 0$, where C_0 has only states of Q_{detect}. It is, of course, not true that last exists for *any* simulation P'_{detect} of P_{detect}. However, one can extend any simulation which does not, by having each agent "remember" its last state in Q_{detect}.

We construct a `DAf`-automaton with weak broadcasts P_{bc} by adding the following

transitions to P'_{detect}.

$$(x, L_{\text{double}}) \mapsto (2x, L), (\{(y,0) \mapsto (2y,0) : y \in \{-k+1, ..., k-1\}\}$$
$$\cup \{q \mapsto \bot : q \in Q_{\text{cancel}} \times \{L, L_{\text{double}}, L_{\square}\}\}) \circ \mathsf{last} \qquad \langle\mathsf{double}\rangle$$

$$(x, L_{\square}) \mapsto \square, (\{(y,0) \mapsto \square : y \in \{-E, ..., -1\}\}$$
$$\cup \{q \mapsto \bot : q \in Q_{\text{cancel}} \times \{L, L_{\text{double}}, L_{\square}\}\}) \circ \mathsf{last} \qquad \langle\mathsf{reject}\rangle$$

These transitions are written somewhat unintuitively. Recall that we write a weak broadcast transition as $q \mapsto q', f$, where $q, q' \in Q'_{\text{detect}}$ are states and $f : Q'_{\text{detect}} \to Q'_{\text{detect}}$ is the transfer function. Usually, we specify f as simply a set of mappings $\{r \mapsto f(r) : r \in Q'_{\text{detect}}\}$. Here, our transition essentially is $q \mapsto q', (f \circ \mathsf{last})$, where $\circ$ denotes function composition, and f is given as a set of mappings. This means that broadcasts first move all agents to their last state in Q_{detect}, and then apply the other mappings as specified.

Before extending P_{bc} with resets that restart the computation from an error state, we analyse the behaviour of P_{bc} in more detail. To talk about accepting/rejecting runs, we define the set of rejecting states as $\{\square\}$. (All other states are accepting.) Let $\pi := (C_0, C_1, ...)$ denote a fair run of P_{bc} starting in a configuration C_0 where all agents are in states $\{-E, ..., E\} \times \{0, L\}$, and at least one agent is in a state $(\cdot, L)$. We refer to the agents starting in $(\cdot, L)$ as *leaders*. Note that it is not possible to enter a state in $Q \times \{L, L_{\text{double}}, L_{\square}\} \cup \{\bot\}$ without being a leader. We usually disregard the first component (if any) while referring to states of leaders.

To argue correctness, we state two properties of P_{bc}. First, it is not possible for *all* leaders to enter $\bot$, which ensures that a reset restarts the computation with a proper subset of the leaders. Second, P_{bc} works correctly if no agent enters an error state. Here, $L_G : X \to \mathbb{N}$ denotes the label count of the input graph, i.e. $L_G(x_i) = |C_0^{-1}(a_i)|$ for $i = 1, ..., l$.

Lemma 6.2. *Assuming that no agent enters state $\bot$, π is accepting iff $\varphi(L_G) = 1$. Additionally, π cannot reach a configuration with all leaders in state $\bot$.*

Resets. Finally, we can add resets to the protocol, to restart the computation in case of errors. We use Lemma 4.7 to construct a `DAf`-computation $P'_{\text{bc}} = (Q'_{\text{bc}}, \delta'_{\text{bc}})$ simulating P_{bc}, and then set $P_{\text{reset}} := P'_{\text{bc}} \times Q_{\text{cancel}} + \langle\mathsf{reset}\rangle$, where the broadcasts are defined as follows, for $q_0 \in Q_{\text{cancel}}$.

$$(\bot, q_0) \mapsto ((q_0, L), q_0), \{(r, r_0) \mapsto ((r_0, 0), r_0) : (r, r_0) \in Q'_{\text{bc}} \times Q_{\text{cancel}}\} \qquad \langle\mathsf{reset}\rangle$$

To actually compute φ, we add the initialisation function $I(x_i) := ((a_i, L), a_i)$ and the set of rejecting states $N := \{\square\}$ to P_{reset} (all other states are accepting).

Proposition 6.3. *For every predicate $\varphi : \mathbb{N}^l \to \{0, 1\}$ such that $\varphi(x_1, ..., x_l) \Leftrightarrow a_1 x_1 + ... + a_l x_l \geq 0$ with $a_1, ..., a_l \in \mathbb{Z}$ there is a bounded-degree `DAf`-automaton computing φ.*

7. Conclusion

We have characterised the decision power of the weak models of computation studied in [16] for properties depending only on the labelling of the graph, not on its structure. For arbitrary networks,the initially twenty-four classes of automata collapse into only four; further, only `DAF` can decide majority. For bounded-degree networks (a well-motivated restriction in a biological setting, also used in e.g. in [3, 12]), the picture becomes more complex. Counting and non-counting automata become equally powerful, an interesting fact because biological models are often non-counting. Further, the class `DAf`, which uses adversarial scheduling, substantially increases its power, and becomes able to decide majority. So, while majority algorithms require (pseudo-)random scheduling to work correctly for arbitrary networks, they can work correctly under adversarial scheduling for bounded-degree networks. In particular, there exist a synchronous deterministic algorithm for majority in bounded-degree networks.

References

[1] Yehuda Afek, Noga Alon, Ziv Bar-Joseph, Alejandro Cornejo, Bernhard Haeupler, and Fabian Kuhn. Beeping a maximal independent set. *Distributed Comput.*, 26(4):195–208, 2013.

[2] Dana Angluin. Local and global properties in networks of processors (extended abstract). In *STOC*, pages 82–93. ACM, 1980.

[3] Dana Angluin, James Aspnes, Melody Chan, Michael J Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In *International Conference on Distributed Computing in Sensor Systems*, pages 63–74. Springer, 2005.

[4] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[5] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Comput.*, 21(3):183–199, 2008.

[6] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Comput.*, 20(4):279–304, 2007.

[7] James Aspnes. Clocked population protocols. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 431–440, 2017.

[8] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. `doi:10.1145/4221.4227`.

[9] Petra Berenbrink, Robert Elsässer, Tom Friedetzky, Dominik Kaaser, Peter Kling, and Tomasz Radzik. A population protocol for exact majority with o(log5/3 n) stabilization time and theta(log n) states. In *DISC*, volume 121 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[10] Andreas Bilke, Colin Cooper, Robert Elsässer, and Tomasz Radzik. Brief announcement: Population protocols for leader election and exact majority with $O(\log^2 n)$ states and $O(\log^2 n)$ convergence time. In *PODC*, pages 451–453. ACM, 2017.

[11] Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive power of broadcast consensus protocols. In *CONCUR*, volume 140 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[12] Olivier Bournez and Jonas Lefèvre. Population protocols on graphs: A hierarchy. In *UCNC*, volume 7956 of *Lecture Notes in Computer Science*, pages 31–42. Springer, 2013.

[13] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, and Paul G. Spirakis. The computational power of simple protocols for self-awareness on graphs. *Theor. Comput. Sci.*, 512:98–118, 2013.

[14] Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *DISC*, volume 6343 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2010.

[15] Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *PODC*, pages 137–146. ACM, 2013.

[16] Javier Esparza and Fabian Reiter. A classification of weak asynchronous models of distributed computing. In *CONCUR*, volume 171 of *LIPIcs*, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[17] Ofer Feinerman and Amos Korman. Theoretical distributed computing meets biology: A review. In *ICDCIT*, volume 7753 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[18] Nissim Francez. *Fairness*. Texts and Monographs in Computer Science. Springer, 1986.

[19] Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *ICALP (2)*, volume 5556 of *Lecture Notes in Computer Science*, pages 484–495. Springer, 2009.

[20] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015.

[21] Adrian Kosowski and Przemyslaw Uznanski. Brief announcement: Population protocols are fast. In *PODC*, pages 475–477. ACM, 2018.

[22] Fabian Kuhn, Nancy A. Lynch, and Rotem Oshman. Distributed computation in dynamic networks. In *STOC*, pages 513–522. ACM, 2010.

[23] Daniel Lehmann, Amir Pnueli, and Jonathan Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *ICALP*, volume 115 of *Lecture Notes in Computer Science*, pages 264–277. Springer, 1981.

[24] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[25] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theor. Comput. Sci.*, 412(22):2434–2450, 2011.

[26] Othon Michail and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *J. Parallel Distributed Comput.*, 81-82:1–10, 2015.

[27] Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, 2015.

[28] Fabian Reiter. Asynchronous distributed automata: A characterization of the modal mu-fragment. In *ICALP*, volume 80 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[29] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

A. Proofs of Section 3

Definition A.1. For every labelled graph $G = (V, E, \lambda)$ over the finite set of Labels $\mathcal{L}$ we write L_G for the multiset of labels occurring in G, i.e. $L_G : \mathcal{L} \to \mathbb{N}, L_G(x) = |\{v \in V | \lambda(v) = x\}|$ for all labels x. We call L_G the *label count* of G.

A graph property φ is called a *labelling property* if for all labelled graphs G, G' with $L_G = L_{G'}$ we have $\varphi(G) = \varphi(G')$. In such a case we also write $\varphi(L_G)$ instead of $\varphi(G)$.

Lemma 3.1. *Let A be a `DaF`-automaton. For all graphs G and H containing a cycle, $\varphi_A(G) = \varphi_A(H)$.*

Proof. Assume there exist cyclic graphs G and H such that A accepts G and rejects H. We construct a graph GH and a run of A on GH such that at least one node of GH halts in an accepting state, and at least one node of GH halts in a rejecting state. This contradicts the assumption that A satisfies the consistency condition.

Let ρ_G and ρ_H be fair runs of A on G and H, and let g and h be the earliest times at which all nodes of G and H have already halted.

Fix edges $e_G = \{u_G, v_G\}$ and $e_H = \{u_H, v_H\}$ belonging to cycles of G and H. We construct the graph GH in three steps. First, we put $2g + 1$ copies of G and $2h + 1$ copies of H side by side. Let G^i, H^i denote the i-th copy of G and H, and let w_G^i and w_H^i denote the copy of a node w_G in G^i or w_H in H^i. Second, we remove the edges $\{u_G^0, v_G^0\}, ..., \{u_G^{2g}, v_G^{2g}\}, \{u_H^0, v_H^0\}, ..., \{u_H^{2h}, v_H^{2h}\}$. Third, we add the edges

$$\{v_G^0, u_G^1\}, ..., \{v_G^{2g-1}, u_G^{2g}\}, \{v_G^{2g}, u_H^0\}, \{v_H^0, u_H^1\}, ..., \{v_H^{2h-1}, u_H^{2h}\}$$

The construction is depicted in Figure 3. Observe that, since e_G and e_H belong to cycles of G and H, the graph GH is connected.
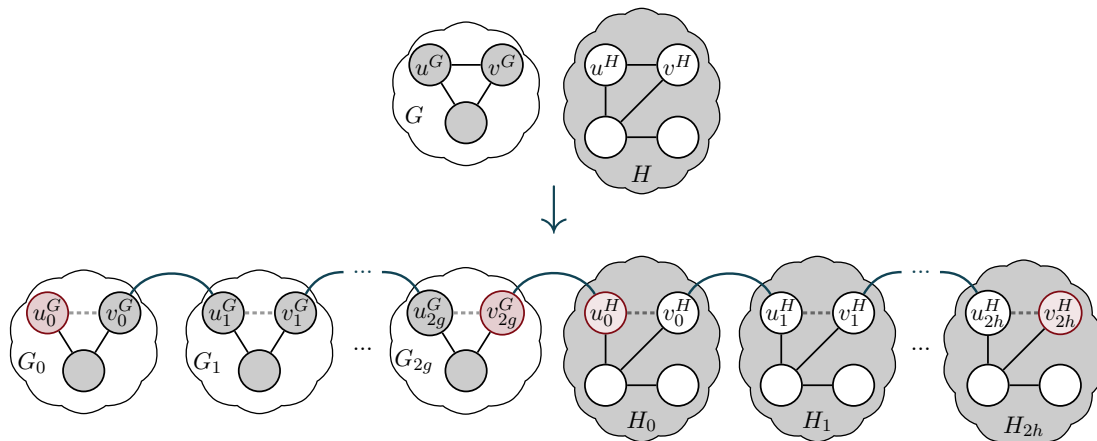

Figure 3: Construction in proof of Lemma 3.1. The dashed edges are removed and replaced by the blue edges. Only the four red states can initially detect the change.

Let ρ_{GH} be any fair run of A on GH that during the first $\max\{g, h\}$ steps selects exactly the copies of the nodes selected at the corresponding steps of ρ_G and ρ_H. (Notice that ρ_{GH} exists, because whether a run is fair or not does not depend on any finite prefix of the run.) Initially, every node of GH except u_G^0, v_G^{2g}, u_H^0, and v_H^{2h} "sees" the same neighbourhood as its corresponding node in G or H (i.e. the same number of neighbours in the same states). Therefore, after the first step of ρ_{GH} all nodes of GH, except possibly these four, are in the same state as their corresponding nodes in G or H after one step of ρ_G or ρ_h. Since the nodes of G^g are at distance at least g from u_G^0, v_G^{2g}, u_H^0, and v_H^{2h}, during the first g steps of ρ_{GH} any node w_G^g of G^g visits the same sequence of states as the node w_G of G during the first g steps of ρ_G. Since all nodes of G halt after at most g steps by definition, all nodes of G^g halt in accepting states. Similarly, after h steps all nodes of H^h halt in rejecting states. $\qquad\square$

Lemma 3.2. *Let A be a `DAf`-automaton. For all graphs G and H, if H is a covering of G, then $\varphi_A(G) = \varphi_A(H)$.*

Proof. Let A be a `DA$`-automaton accepting φ. Let $f: V_H \to V_G$ be a covering map

24

respecting the labelling, i.e. fulfilling $\lambda_H = \lambda_G \circ f$. We prove that A accepts G iff it accepts H.

Let $\rho_G = (C_0, C_1, ...)$ be the synchronous run of A on G, and let $\rho_H = (C'_0, C'_1, ...)$ be the synchronous run of A on H. Observe that, since selection is adversarial, the synchronous runs are fair runs. Since A satisfies the consistency condition, it suffices to show that ρ_G accepts G iff it accepts H. For this we prove by induction on t that $C_t(v) = C_t(f(v))$ holds for every node v of H and $t \geq 0$. For $t = 0$ this follows from the fact that f respects the labelling. For $t > 0$, assume $C_t(v) = C_t(f(v))$ we prove $C_{t+1}(v) = C_{t+1}(f(v))$. Pick an arbitrary node u. Since $C_t(u) = C_t(f(u))$, both u and $f(u)$ occupy the same state in C_t. Since the run is synchronous, both are selected. Since the restriction of the covering f to the neighbourhoods of u and $f(u)$ is a bijection, and $C_t(v) = C_t(f(v))$ holds for all v, in particular for all neighbours of u, both u and $f(u)$ move to the same states. So $C_{t+1}(u) = C_{t+1}(f(u))$ $\qquad\square$

Corollary 3.3. *Let A be a `DAf`-automaton deciding a labelling property. For all graphs G and H, if $L_H = \lambda L_G$ for some $\lambda \in \mathbb{N}_{>0}$, then $\varphi_A(G) = \varphi_A(H)$. This also holds when restricting to k-degree-bounded graphs.*

Proof. Let L be a multiset of labels and enumerate it as $L = (\lambda_1, \lambda_2, ..., \lambda_{|L|})$. Enumerate $\lambda \cdot L$ by repeating this sequence λ times. Since φ is a labelling property, the underlying graph does not influence whether the property holds. We consider the following graphs: The cycle G labelled with L in the order we established, and the cycle G' labelled with $\lambda \cdot L$ in the order above. We have that G' covers G, and therefore using Lemma 3.2 obtain $\varphi(L) = \varphi(\lambda \cdot L)$. Since the graphs G and G' are 2-degree-bounded, this statement holds also when restricting to k-bounded-degree. $\qquad\square$

Lemma 3.4. *Let A be a `DAf`-automaton with counting bound β that decides a labelling property. For all graphs G and H, if $\lceil L_G \rceil_{\beta+1} = \lceil L_H \rceil_{\beta+1}$ then $\varphi_A(G) = \varphi_A(H)$, i.e. φ_A admits a cutoff.*

Proof. Let A be a `DA$`-automaton with counting bound β that decides φ.

Since φ is a labelling property, A accepts a graph G iff it accepts the unique clique G' (up to isomorphism) such that $L_G = L_{G'}$. Therefore, it suffices to prove $\varphi(G) = \varphi(H)$ for the case in which G and H are *cliques* satisfying $\lceil L_G \rceil_{\beta+1} = \lceil L_H \rceil_{\beta+1}$.

Since A is an automaton with adversarial selection, the synchronous runs $\rho_G = (C_{G0}, C_{G1}, ...)$ of A on G, and $\rho_H = (C_{H0}, C_{H1}, ...)$ of A on H are fair runs of A. Since A satisfies the consistency condition, A accepts G iff ρ_G is an accepting run, and similarly for H. So it suffices to show that ρ_G is an accepting run iff ρ_H is.

Let $Q_{Gt} \colon Q \to \mathbb{N}$ be the mapping that assigns to each state q of A the number of nodes of G that are in state q at time t. Define Q_{Ht} analogously. We claim: $\lceil Q_{Gt} \rceil_{\beta+1} = \lceil Q_{Ht} \rceil_{\beta+1}$ for every $t \geq 0$. The proof is by induction on t. For the base case $t = 0$, let q be a state. By definition, $Q_{G0}(q)$ is the number of nodes of G that are initially at state q. Let Λ_q be the set of labels that are mapped to q by the initialisation function of A. Then we have $Q_{G0}(q) = \sum_{\ell \in \Lambda_q} L_G(\ell)$. Since $\lceil L_G \rceil_{\beta+1} = \lceil L_H \rceil_{\beta+1}$, we get $\lceil Q_{G0} \rceil_{\beta+1} = \lceil Q_{H0} \rceil_{\beta+1}$.

For the induction step, assume $\lceil Q_{Gt} \rceil_{\beta+1} = \lceil Q_{Ht} \rceil_{\beta+1}$. We prove $\lceil Q_{G(t+1)} \rceil_\beta = \lceil Q_{H(t+1)} \rceil_\beta$. It suffices to show that if two nodes u and v of $G \cup H$ are in the same state at time t, then they are also in the same state (possibly a different one) at time $t + 1$. For this, observe first that, since G and H are cliques, all nodes of G respectively H are neighbours. So, since $\lceil Q_{Gt} \rceil_{\beta+1} = \lceil Q_{Ht} \rceil_{\beta+1}$, the nodes u and v see the same neighbourhood up to β at time t (i.e. they see the same number of nodes in each state up to bound β; we go from $\beta + 1$ to β because the neighbourhood of the node does not contain the node itself). In other words, $N_u^{C_{Gt}} = N_v^{C_{Gt}}$ holds. Since the runs ρ_G and ρ_H are synchronous, both u and v are selected at time t to make a move. Since $N_u^{C_{Gt}} = N_v^{C_{Gt}}$, they move to the same state, and the claim is proved.

Assume that ρ_G is an accepting run of A. Then there is a time t such that for every $j \geq 0$ all nodes of G are at accepting states in $C_{G(t+j)}$. By the claim, the same holds for $C_{H(t+j)}$. So ρ_H is an accepting run of A. The other direction is analogous. $\qquad\square$

Lemma 3.5. *Let A be a `dAF`-automaton that decides a labelling property. There exists $K \geq 0$ such that for every graph G and H, if $\lceil L_G \rceil_K = \lceil L_H \rceil_K$ then $\varphi_A(G) = \varphi_A(H)$, i.e. φ_A admits a cutoff.*

Proof. We start by repeating some notation of the proof sketch. Let A be the `dAF`-automaton, and let Q be its set of states. We first gather some properties of A on *star graphs*. A star is a graph in which a node called the *center* is connected to an arbitrary number of nodes called the *leaves*, and no other edges exist. Since we consider graphs up to isomorphism, a configuration of a star graph is completely determined by the state of the center and the number of nodes in each state. So in the rest of the proof we assume that a configuration of a star graph G is a pair $C = (C^{\text{ctr}}, C^{\text{sc}})$, where C^{ctr} denotes the state of the center of G, and C^{sc} is the *state count* of C, i.e. the mapping that assigns to each $q \in Q$ the number $C^{\text{sc}}(q)$ of nodes of G that are in state q at C. We denote the *cutoff* of C as $\lceil C \rceil_m := (C^{\text{ctr}}, \lceil C^{\text{sc}} \rceil_m)$.

Given a configuration C of A, recall that C is rejecting if all states are rejecting. We say that C is *stably rejecting* if C can only reach configurations which are rejecting. Given an initial configuration C_0, it is clear that A must reject if it can reach a stably rejecting configuration C from A. Conversely, if it cannot reach such a C, then A will not reject C_0, as there is a fair run starting at C_0 which contains infinitely many configurations which are not rejecting.

The statement missing in the proof sketch is the following: There exists a number $m \in \mathbb{N}$ such that a star configuration C is stably rejecting if and only if $\lceil C \rceil_m$ is stably rejecting.

To define m, we have to first define some additional concepts. Given two configurations C, D of A on stars G and H, we say that $C \preceq D$ holds if (a) $C^{\text{ctr}} = D^{\text{ctr}}$, (b) $C^{\text{sc}} \geq D^{\text{sc}}$, and (c) $D^{\text{sc}}(q) = 0$ implies $C^{\text{sc}}(q) = 0$ for every $q \in Q$. It is easy to see that $\preceq$ is a partial order. Further, if $C \preceq D$ then C is accepting (rejecting) iff D is accepting (rejecting). A set $\mathcal{C}$ of configurations is *upward closed* if $C \in \mathcal{C}$ and $D \succeq C$ implies $D \in \mathcal{C}$.

Let $C \rightarrow^* D$ denote that A can reach the configuration D from C in zero or more steps. Given a set of configurations $\mathcal{C}$, let $Pre^*(\mathcal{C})$ be the set of configurations C such that $C \rightarrow^* D$ for some $D \in \mathcal{C}$. The following two claims will finally allow us to define m:

(1) If $C \to^* D$ and $C' \succeq C$, there exists $D' \succeq D$ such that $C' \to^* D'$.

Since $C' \succeq C$, we can obtain C' from C by adding leaves in states which already occur. Similar to the last argument given in the proof sketch, we let every one of these extra leaves copy one of the leaves from C which starts in the same state. Formally, let $v_{new,1}, ..., v_{new,n}$ be the extra leaves. For every extra leaf $v_{new,i}$ let $v_{old,i}$ be some leaf starting in the same state, not necessarily distinct for $i_1 \neq i_2$. Now let $\rho = (v_1, ..., v_\ell) \in V^*$ denote a sequence of selections for A to go from C to D. We construct a sequence $\sigma \in V^*$ by inserting a selection of $v_{new,i}$ after every selection of $v_{old,i}$, if multiple i have the same $v_{old,i}$, insert all of them after $v_{old,i}$ in some order. Define D' as the configuration which A reaches after executing σ from C'. We claim that D' is the same as D, apart from having additional leaves in the same states as v_{old}. This follows from a simple induction: $v_{old,i}$ and $v_{new,i}$ start in the same state and see only the root node. As they are always selected without the root being selected in between, they will remain in the same state as each other. For the centre we use the property that A cannot count: it cannot differentiate between seeing just $v_{old,i}$, or seeing one (or maybe more) additional nodes in the same state.

(2) For every upward-closed set of configurations $\mathcal{C}$, the set $Pre^*(\mathcal{C})$ has finitely many minimal configurations w.r.t. $\preceq$. We denote this finite set by $MinPre^*(\mathcal{C})$.

Assume for contradiction that $Pre^*(\mathcal{C})$ has infinitely many minimal configurations. We can enumerate its elements to obtain an infinite sequence $C_1, C_2, ...$ of configurations of star graphs. By the pigeonhole principle, there exists an infinite subsequence $C_{i_1}, C_{i_2}, ...$ such that $C_{i_j}^{\mathrm{ctr}} = q$ for some state q and every $j \geq 0$, and $C_{i_j}^{\mathrm{sc}}(q) = 0$ iff $C_{i_k}^{\mathrm{sc}}(q) = 0$ for every $j, k \geq 1$ and every state q. By Dickson's Lemma (for every infinite sequence of vectors $v_1, v_2, ... \in \mathbb{N}^k$, there exist two indices $i < j$ such that $v_i \leq v_j$ with respect to the pointwise partial order), there exist j, k such that $C_{i_j} \leq C_{i_k}$. But then C_{i_j} and C_{i_k} satisfy conditions (a)-(c) of the definition of $\preceq$, and so $C_{i_j} \preceq C_{i_k}$, which is a contradiction.

Now we can define m. Consider the set $\mathcal{C}$ of non-rejecting configurations of A on all star graphs, with any number of leaves. It is easy to see that $\mathcal{C}$ is upward closed. By the claim the set $MinPre^*(\mathcal{C})$, i.e. the set of smallest configurations from which it is possible to reach a non-rejecting configuration, is finite. Let m be the number of nodes of the largest star such that some configuration of it belongs to $MinPre^*(\mathcal{C})$. In other words: for every star with more than m nodes, and for every configuration C of this star that can reach a non-rejecting configuration, i.e. is not stably rejecting, there is a configuration $C' \prec C$ of another star that can also reach a non-rejecting configuration, i.e. is not stably rejecting. Combining this with the fact that $MinPre^*(\mathcal{C})$ is upward-closed, we obtain that for every configuration C, C is not stably rejecting if and only if $\lceil C \rceil_m$ is not stably rejecting. By contraposition, this implies the statement we wanted to prove. $\qquad\square$

B. Proofs of Section 4

The main goal of this section as a whole is to prove that the different models with weak broadcasts, weak absence detection as well as rendezvous transitions can be simulated. We will proceed as follows.

1. We start by proving Lemmata B.1 and 4.4, which are general properties of reorderings.

2. In subsection B.1 we show a general lemma concerning reorderings of three-phase protocols, which are used for both weak broadcasts and weak absence detection. In particular, we prove that there is a reordering where all nodes move in lock step.

3. This will dramatically shorten the proofs that weak broadcasts and weak absence detection can be simulated, which make up the next two subsections.

4. At last, we prove that rendezvous transitions can be simulated by `DAF`-automata.

Lemma B.1. *Let $G = (V, E, \lambda)$ be a labelled graph. Let $\pi = (C_0, C_1, ...)$ denote a run on G, $\pi_f = (C'_0, C'_1, ...)$ a reordering of π, and v a node. For all $t \in \mathbb{N}_0$ where v is selected for a non-silent transition and all nodes u adjacent or identical to v we have $C_t(u) = C'_{f(t)}(u)$.*

Proof. Write the node sequence π as $(v_0, v_1, v_2, ...)$. Write the neighbourhood of a node v as $N(v)$. Write the reordered run as $\pi' = (C'_0 = C_0, C'_1, C'_2, ...)$. We assume wlog that all silent transitions were removed, unless no non-silent transition is enabled. The proof will proceed by induction on t.

For the induction basis $t = 0$ we have to prove that before time $f(0)$, no neighbour of v_0 has changed state yet in the reordered run. Assume for contradiction that some neighbour has changed state already, i.e. we have $f(i) < f(0)$ for some i with $v_i \in N(v_0)$. Since $i > 0$ and $\{v_i, v_0\} \in E$, we would have $f(i) > f(0)$, contradicting the definition of a reordering.

For the induction step, let $v \in N(v_t)$. We have to prove $C_t(v) = C'_{f(t)}(v)$. Consider the latest time $s < t$ where v has been selected. We obtain $C_t(v) = C_{s+1}(v)$. By induction hypothesis, we have $C_s(N(v)) = C'_{f(s)}(N(v))$. This implies $C_{s+1}(v) = C'_{f(s)+1}(v)$. Since v is a neighbour of v_t, we have $f(s) < f(t)$. We claim that v has not been selected between time $f(s)$ and $f(t)$ in the reordered run. Assume for contradiction that v has been selected at time $f(s) < t' < f(t)$. Let m be such that $f(m) = t'$, which exists because we removed silent transitions. Since $f(s) < f(m) < f(t)$ and $\{v_m, v_t\} \in E$, we have $m < t$. We similarly obtain $s < m$. Therefore s would not have been the latest time before t where v moved, yielding a contradiction. Therefore we have $C'_{f(s)+1}(v) = C'_{f(t)}(v)$. $\square$

Lemma 4.4. *Let $P = (Q, \mathrm{Run}, \delta_0, Y, N)$ denote a generalised graph protocol deciding a predicate φ, and P' an automaton simulating P. Then there is an automaton P'' simulating P which also decides φ.*

Proof. For P'' we reuse δ_0 as initialisation function. We change P' so that each agent remembers its last non-intermediate state. We define Y' as the set of states where the

28

last non-intermediate state is in Y, and define N' analogously. Then we use Y', N' as accepting/rejecting states for P''. Any run π of P'' starting in an initial configuration has a reordering π_f which is an extension of a run τ of P. If P accepts, then every node v is only finitely often in a state $Q \setminus Y$ in τ, which then also holds for π_f and π. Thus, v will eventually remain in Y', and π accepts as well. Similarly, P'' will reject if P does. $\quad\square$

This lemma also explains why we treat silent transitions separately: To simulate weak broadcasts, we use a three-phase protocol and want to prove that we can reorder the run such that all nodes move to phase 1, then all to phase 2 and so on. However, Lemma B.1 shows that if some node v observes a neighbour who is behind a phase and a neighbour who is ahead, then this would have to be reflected at some point in time in the reordered run, making it impossible for all nodes to be at most one phase apart. However, in this case our protocols have v do nothing, so removing silent transitions resolves the issue.

B.1. Reorderings in three-phase automata

As we want to make a general statement about three-phase protocols, we start by formally introducing the notion. The idea is that each state belongs to one of three phases, that agents may not move directly to the previous phase, and that an agent does nothing, unless all its neighbours are in the same or the next phase. Further, we require that an agent either moves to the next phase or does nothing, if it has a neighbour in the next phase. The last condition is rather technical, it will later allow us construct a reordering which executes the transitions that do not move agents into the next phase first, before executing the other transitions.

Definition B.2. Let $P = (Q, \delta_0, \delta, Y, N)$ denote an automaton with counting bound β. We say that P is a *three-phase automaton* if $Q = Q_0 \cup Q_1 \cup Q_2$, for some pairwise disjoint Q_0, Q_1, Q_2, and for all states $q \in Q_i$ and neighbourhoods $N : Q \to [\beta]$ we have

1. $\delta(q, N) = q$ if $N(r) > 0$ for some $r \in Q_{i-1}$,
2. $\delta(q, N) \in Q_i \cup Q_{i+1}$, and
3. $\delta(q, N) \in \{q\} \cup Q_{i+1}$ if $N(r) > 0$ for some $r \in Q_{i+1}$.

Here, we set $Q_3 := Q_0$ and $Q_{-1} := Q_2$ for convenience. We refer to states in Q_i as *phase-i* states.

As defined above, it is possible for a three-phase automaton to get "stuck" when some agents move to the next phase, but others cannot make progress. Our protocols will not have this problem, so we define the following semantic constraint.

Definition B.3. A three-phase automaton $P = (Q, \delta_0, \delta, Y, N)$ is *nonblocking* if every reachable configuration $C : V \to Q$ which has agents from at least two phases will eventually execute a transition where an agent changes phase.

With these definitions we can now state the main proposition of this section.

Proposition B.4. *Let $P = (Q, \delta)$ denote a nonblocking three-phase automaton and $C_0 : V \to Q_0$ an initial configuration. Then every fair run $\pi = (C_0, C_1, ...)$ of P has a reordering $\pi_f = (C'_0, C'_1, ...)$ which fulfils, for each step i,*

1. *$C'_i(V) \subseteq Q_j \cup Q_{j+1}$ for some j, and at step i an agent moves to its next phase, or*

2. *$C'_i(V) \subseteq Q_j$ for some j.*

The proof will take up the remainder of this section. We now fix such a $P = (Q, \delta)$ and $\pi = (C_0, C_1, ...)$.

It will be convenient to count the total number of phases changes of a node, so we define the *phase count* $\mathsf{pc}(v, i) \in \mathbb{N}$ as the smallest function which is non-decreasing w.r.t. i and has $C_i(v) \in Q_j$ for all i, v and $j := (\mathsf{pc}(i, v) \bmod 3)$. Intuitively, this means that we increment pc whenever a node moves to the next phase. Observe that $\mathsf{pc}(v, i+1) - \mathsf{pc}(v, i) \leq 1$, as a node can move at most one phase per transition.

Lemma B.5. *For all adjacent nodes $u, v \in V$ we have $|\mathsf{pc}(u, i) - \mathsf{pc}(v, i)| \leq 1$ for all i.*

Proof. Assume for contradiction that the statement does not hold and prick appropriate u, v, i where i is minimal and $\mathsf{pc}(u, i) = \mathsf{pc}(v, i) - 1$. Then at step $i - 1$ node v must move to the next phase, but this is prohibited by condition (2) of Definition B.2. $\square$

Lemma B.5 implies that if one node has infinitely many phase changes, then all nodes do. We will now show the stronger statement that if a node has m phase changes, for $m \in \mathbb{N} \cup \{\infty\}$, then all other nodes have as well. Here we use that P is nonblocking, so it is not possible for nodes to become stuck in prior phases.

Lemma B.6. *If pc is bounded, i.e. $\mathsf{pc}(v, i) \leq M$ for some $M \in \mathbb{N}$ and all v, i, then there are $m, i \in \mathbb{N}$ with $\mathsf{pc}(v, j) = m$ for all nodes v and $j \geq i$.*

Proof. Assume that pc is bounded. As pc is non-decreasing we can thus find a i s.t. no node moves to another phase after step i. If C_i has two nodes u, v with different phase counts, i.e. $\mathsf{pc}(u, i) < \mathsf{pc}(v, i)$, then must also be such u, v which are adjacent. Due to Lemma B.5, the phase counts of u and v differ only by 1, therefore we know that $C_i(u) \in Q_j$ and $C_i(v) \in Q_{j+1}$ for some j. As P is nonblocking, eventually an agent will move to its next phase, contradicting our choice of i. So at step i all phase counts must be pairwise equal. $\square$

Lemma B.6 is crucial for the reordering, since in the new run of A all nodes are supposed to perform the same number of phase changes. Now we can define the reordering f. Let $(v_0, v_1, v_2, ...) \in V^\omega$ be the sequence of selections inducing run π.

We define a new ordering on natural numbers by

$$i \leq_f j \Leftrightarrow (\mathsf{pc}(v_i, i), \mathsf{pc}(v_i, i+1), i) \leq_{\text{lex}} (\mathsf{pc}(v_j, j), \mathsf{pc}(v_j, j+1), j)$$

where $\leq_{\text{lex}}$ denotes the lexicographical ordering. The intuition is that we always execute a transition from a node with the lower phase count. Amongst those, we pick one that will not move the node to its next phase, if possible. Finally, from the remaining choices

we pick the one that occurred first in the original run π. Let $I := \{i \in \mathbb{N} : C_i \neq C_{i+1}\}$ denote the indices of non-silent steps of π. The function $f : I \to \mathbb{N}$ can now be defined as $f(i) := |\{j \in I : j \leq_f i\}| - 1$. (We will see shortly that f is indeed well-defined.)

Lemma B.7. π_f *is a reordering.*

Proof. We first check that f is well-defined. For this not to be the case, we would have to have an i with $j \leq_f i$ for infinitely many j. In particular, there must be a node u s.t. there are infinitely many $j \leq_f i$ with $v_j = u$, which implies that $\mathsf{pc}(u, j) < \mathsf{pc}(v_i, i+1)$ for infinitely many j. Therefore pc must be bounded by Lemma B.5, but then Lemma B.6 implies that $\mathsf{pc}(v, j) < \mathsf{pc}(v_i, i + 1)$ can only occur for finitely many j, a contradiction.

The function f is clearly a bijection. In order to show that it induces a reordering, let $i, j \in I$ with $i < j$ and v_i, v_j being adjacent. (Note that the transitions at steps i and j are not silent, by definition of I.) We need to show that $f(i) < f(j)$. Assuming that $f(i) \geq f(j)$ holds, i.e. $i \geq_f j$, there are two possible cases.

Case 1: $\mathsf{pc}(v_i, i) > \mathsf{pc}(v_j, j)$. As $\mathsf{pc}(\cdot, t)$ is non-decreasing in t and $i < j$, we get $\mathsf{pc}(v_i, i) > \mathsf{pc}(v_j, i)$. Further, v_i and v_j are adjacent, so Lemma B.5 implies that $\mathsf{pc}(v_i, i) = \mathsf{pc}(v_j, i) + 1$. But then, by condition (1) of Definition B.2, the transition at step i must be silent, contradicting our assumption.

Case 2: $\mathsf{pc}(v_i, i) = \mathsf{pc}(v_j, j)$ and $\mathsf{pc}(v_i, i + 1) > \mathsf{pc}(v_j, j + 1)$. Again, $\mathsf{pc}(\cdot, t)$ is non-decreasing in t, so $\mathsf{pc}(v_i, j) \geq \mathsf{pc}(v_i, i + 1) > \mathsf{pc}(v_j, j + 1) \geq \mathsf{pc}(v_j, j)$. Using Lemma B.5 we now get $\mathsf{pc}(v_i, j) = \mathsf{pc}(v_j, j) + 1$ and thus $\mathsf{pc}(v_j, j + 1) = \mathsf{pc}(v_j, j)$. So at step j node v_i is one phase ahead of its neighbour v_j and v_j moves neither to the next phase, nor does it perform a silent transition. This contradicts condition (3) of Definition B.2. $\square$

Let $\pi' := \pi_f$ denote the reordered execution and define pc' analogously to pc. To complete the proof of Proposition B.4, it suffices to show that at each step of π' an agent with the smallest phase count will be selected, and amongst those the agents that do not move to the next phase are preferred. Intuitively, this sounds reasonable, as we have defined the reordering f in precisely this manner. We do, however, need to argue briefly that the phase counts pc' of the reordered execution correspond directly to the original phase counts pc.

Lemma B.8. *Let v denote a node and let $i \in I$ with $v_i = v$. Then $\mathsf{pc}'(v, f(i)) = \mathsf{pc}(v, i)$.*

Proof. This follows from a simple induction on i combined with Lemma B.1. $\square$

This concludes the proof of Proposition B.4.

B.2. Simulating Weak Broadcasts

Lemma 4.7. *Every automaton with weak broadcasts is simulated by some automaton of the same class without weak broadcasts.*

(We repeat the construction from the proof sketch for clarity.)

Let $P = (Q, \delta_0, \delta, Q_B, B)$ denote an automaton with weak broadcasts. We define an automaton $P' = (Q', \delta_0', \delta')$ simulating P. The automaton P' has three phases, called 0,1,

and 2. A node moves to the next phase (modulo 3) only if every neighbour is in the same phase or in the next. The states of P' are $Q' := Q \cup Q \times \{1, 2\} \times Q^Q$. Intuitively, an agent of P' in state $q \in Q$ is in phase 0, and simulates an agent of P in state q; an agent of P' in state $(q, i, f) \in Q \times \{1, 2\} \times Q^Q$ is in phase i, and simulates an agent executing P in state q, and initiating or responding to a broadcast with response function f.

Let β denote the counting bound of P. To specify the transitions, for a neighbourhood $N : Q' \to [\beta]$ we write $N[i] := \sum_{q,f} N((q, i, f))$ for $i \in \{1, 2\}$ and $N[0] := \sum_{q \in Q} N(q)$ to denote the number of adjacent agents in a particular phase, and choose a function $g(N) \in Q^Q \cup \{\square\}$ s.t. $g(N) = f \neq \square$ implies $N((q, 1, f)) > 0$, and $g(N) = \square$ implies $N[1] = 0$. The function g is used to select which broadcast to execute, if there are multiple possibilities. We define the following transitions for δ', for all states $q \in Q$ and neighbourhoods $N : Q \to [\beta]$.

$$
\begin{align}
q, N &\mapsto \delta(q, N) & &\text{if } q \notin Q_B \text{ and } N[0] = |N| \tag{1}\\
q, N &\mapsto (q', 1, f) & &\text{if } q \in Q_B \text{ and } N[0] = |N|, \text{ with } (q', f) := B(q) \tag{2}\\
q, N &\mapsto (f(q), 1, f) & &\text{if } g(N) = f \neq \square \tag{3}\\
(q, 1, f), N &\mapsto (q, 2, f) & &\text{if } N[0] = 0 \tag{4}\\
(q, 2, f), N &\mapsto q & &\text{if } N[1] = 0 \tag{5}
\end{align}
$$

We will use the definitions and results from the previous section, Appendix B.1, where we have constructed a general reordering for three-phase protocols.

Lemma B.9. *The automaton P' is a nonblocking three-phase automaton.*

Proof. Using Q_i to denote the phase i states as defined above, it is easy to check that P' is a three-phase automaton. It remains to show that P' is nonblocking, so let $\pi = (C_0, C_1, ...)$ denote a fair run of P' and $C_i : V \to Q'$ a configuration where two agents are in different phases. We define the phase count pc as for the proof of Proposition B.4, meaning that $\mathsf{pc}(v, j)$ is the number of phase changes of node v until step j.

Let $U := \{u \in V : \mathsf{pc}(u, i) = \min_v \mathsf{pc}(v, i)\}$ denote the set of nodes which have a minimal number of phase changes at step i. We know that C_i has nodes of two different phases, so U is a proper subset of V and we can pick adjacent nodes u, v with $u \in U$ and $v \notin U$. We claim that the next step selecting u will move it to its next phase. This can be seen by a simple case distinction: if u is in phase 0, 1, or 2, then transition (3), (4), or (5) will move it to the next phase, respectively. Selecting $u \in U$ ensures that u has no neighbours in the previous phase, which already suffices to enable (4) and (5), while having u adjacent to v ensures that (3) can be executed. $\square$

Again, let π denote a fair run of P'. Using Proposition B.4 we find a specific reordering $\pi_f = (C_0, C_1, ...)$ of π. In particular, every configuration C_i either has all agents in the same phase, or it has agents in at most two phases and at step i one agent moves to the its next phase. We are now going to show that π_f is an extension of a fair run τ of P, which will complete the proof of Lemma 4.7.

32

First, note that in π_f there are infinitely many configurations C_i where all agents are in the same phase. Due to transitions (4) and (5) it is not possible to perform a silent transition if all agents are in phase 1 or 2, respectively. So the set $I := \{i \in \mathbb{N} : C_i(V) \subseteq Q\}$ of indices i where C_i has only phase 0 agents has infinitely many elements. We define the mapping $g : \mathbb{N} \to \mathbb{N}$ as the unique bijection with $g(\mathbb{N}) = I$ which is strictly increasing, and set $\tau := (K_0, K_1, ...)$ where $K_i := C_{g(i)}$ for all i.

Lemma B.10. *π_f is an extension of τ.*

Proof. Fix any $i, j \in \mathbb{N}$ with $g(i) < j < g(i+1)$. Due to the definition of g we know that $C_{g(i)+1}$ does not contain only phase 0 agents, while $C_{g(i)}$ does. So an agent has moved to the next phase at step $g(i)$ in π_f and the properties of π_f guarantee us that there are t_1, t_2 with $g(i) < t_1, t_2 < g(i+1)$ s.t. C_{t_1} (C_{t_2}) has only agents in phase 1 (phase 2). Moreover, we know that in π_f every step t with $g(i) \leq t < t_1$ or $t_2 \leq t < g(i+1)$ moves an agent to its next phase or is silent. As phases 1 and 2 consist of only intermediate states, this implies that $C_{g(i)} \sim_Q C_j$ if $j < t_1$, $C_{g(i+1)} \sim_Q C_j$ if $j > t_2$, and $C_{g(i)} \sim_Q C_j \sim_Q C_{g(i+1)}$ if $t_1 \leq j \leq t_2$. $\square$

The next lemma is mostly a matter of looking carefully at the definition of our transitions (1)-(5).

Lemma B.11. *τ is a run of P.*

Proof. Fix any $i \in \mathbb{N}$. If $g(i+1) = g(i) + 1$, then step $g(i)$ of π_f has simply executed transition (1), which correctly performs a neighbourhood transitions for an agent not in a broadcast-initiating state. Otherwise, as we argued for Lemma B.10, there is a $g(i) < t_1 < g(i+1)$ s.t. C_{t_1} has only agents in phase 1. Let S denote the set of agents executing transition (2) between steps $g(i)$ and t_1 in π_f. As agent can only move from phase 0 to phase 1 via transitions (2) and (3), and transition (3) is enabled iff a neighbour is already in phase 1, we find that S is both nonempty and an independent set. Additionally, the definition of (2) ensures that S contains only agents in broadcast-initiating states (i.e. $K_i(v) \in Q_B$ for $v \in S$).

Now we simply note that K_{i+1} is the result of executing a weak broadcast transition on K_i on the selection S. Transition (2) correctly perform the local update, while (3) moves the node according to some response function. $\square$

Finally, we have to show that τ is fair.

Lemma B.12. *τ is fair.*

Proof. There are two cases, depending on whether P uses adversarial or pseudo-stochastic scheduling.

We start with the former. Here, π either contains infinitely many transitions where an agent moves to the next phase, in which case τ executes infinitely many broadcasts and is fair by definition, or there is some i with $K_i + j = C_{g(i)+j}$ for all $j \in \mathbb{N}$. The transitions in π_f after step i do not move to new phases, so they are not affected by the reordering f. In particular, as π is fair, so is τ.

Now we consider the case of pseudo-stochastic scheduling. It is well-known that a pseudo-stochastic schedule (i.e. every finite sequence of selections appears infinitely often) implies that every configuration C which can be reach infinitely often will be reached infinitely often. (This follows from there being only finitely many distinct configurations in a run.) That argument can be strengthened to show that, for any finite sequence σ of selections, σ will be executed starting from C infinitely often.

Further note that any configuration C which appears infinitely often in τ and thus in π_f can be reached infinitely often in π. This can be seen by choosing a specific reordering which executes the transitions of π faithfully up to a point, and then only executes only the transitions necessary to reach C in π_f. It is easy to see that this a valid reordering (using that π_f is reordering).

Now let σ denote any finite sequence of selections of P, the automaton with weak broadcasts. We want to show that σ is infinitely often in τ. So we pick any configuration K appearing infinitely often in τ, and therefore can be reached infinitely often π. It would now suffice to show that, in τ, K is followed infinitely often by the selections of σ. As we know that for any sequence σ' of selections of P' we have that K appears infinitely often in π followed by the selections of σ', it now suffices to show that we can pick an appropriate σ' that would lead to σ being executed in τ.

To execute a selection (n, v) at configuration $K : V \to Q$, i.e. a neighbourhood transition of node $v \in V$, we either have $K(v) \notin Q_B$ and select v, thus executing transition (1), or we do nothing. (To be precise, in the latter case we would have to append a copy of K to τ, and modify g s.t. τ is still an extension of π_f.) For a selection (b, S) with $S \subseteq V$ an independent set of broadcast initiating nodes, we either have $S' = \emptyset$ with $S' := S \cap K^{-1}(Q_B)$ and again do nothing, or we select all agents in S' to move them to phase 1 via transition (2), then move all other nodes to phase 1 via transition (3), and then use transitions (4) and (5) to move all nodes to phase 2 and then back to phase 0. $\qquad\square$

B.3. Simulating Weak Absence Detection

Lemma 4.9. *Every `DA$`-automaton with weak absence detection is simulated by some `DAf`-automaton, when restricted to bounded-degree graphs.*

The proof will take up the remainder of this section.

As mentioned in the main paper, we combine a three-phase protocol with a distance-labelling, the latter allowing us to propagate information about the states that have been seen back to the agents initiating the absence detection. Before we define the necessary transitions, we briefly characterise the distance labelling we are going to use.

Definition B.13. Let $k \in \mathbb{N}$. We use D to denote a set of *(distance) labels*, where $D := \mathbb{Z}_{2k+1} \cup \{\text{root}\}$. We define increment on D by using the usual arithmetic (modulo $2k + 1$) for elements in $\mathbb{Z}_{2k+1}$, and setting $\text{root} + 1 := 1 \in \mathbb{Z}_{2k+1}$. We refer to root as *root label*. For all $d \in D$ we say that $d + 1$ is the *child label* of d.

Nodes initiating the absence detection use the root label. Each other node will pick a child label d of one of its neighbours, taking care that no neighbour holds a child label

of d. At this point we will use the bound on the maximum degree of the graph, which makes it easy to see that this is always possible.

Lemma B.14. *Let $S \subset D$ with $0 < |S| \leq k$. Then there is a label $d \in D$ s.t. $d + 1 \notin S$ and there is some label $d' \in S$ with $d' + 1 = d$.*

Proof. Note that the statement can be simplified to there being a $d \in S$ with $d + 2 \notin S$. Pick any $d \in S$. As $2k + 1$ is odd, the sequence $d, d+2, d+4, ..., d+2k$ is pairwise distinct. Additionally, it contains $2k + 1 > |S|$ elements, and thus at least one element not in S. Moreover, we can thus find two subsequent elements $d', d' + 2$ with $d' \in S$, $d' + 2 \notin S$ in that sequence. $\square$

We will now formally define our construction. Let $P = (Q, \delta, Q_A, A)$ denote the automaton we want to simulate, k the maximum degree of our graph. We will construct a `DAf`-automation $P' = (Q', \delta')$ simulating P. As states we use $Q' := Q_0 \cup Q_1 \cup Q_2$, where Q_i contains the phase i states. In particular, we set $Q_0 := Q$, $Q_1 := Q^2 \times D$ and $Q_2 := Q \times 2^Q$. For $(q, r, i) \in Q_1$, an agent v carries its phase 0 state r and a distance label $i \in D$. In phase 2 an agent stores the set of states that it has seen so far, which will be propagated to its parents.

To define the transitions δ', we introduce some notation. For any neighbourhood $N : Q' \to \mathbb{N}$ we write $N(S) := \sum_{q \in S} N(q)$ for $S \subseteq Q'$. We set $\mathrm{old}(N) := N'$, where $N'(q) := N(q) + N(Q \times \{q\} \times \{0, ..., 2k\})$ is the number of agents that were in $q \in Q$ in phase 0. We will use $\mathrm{old}(N)$ to determine which neighbourhood transition of P to execute.

We also define a unique child label $\mathrm{child}(N)$ for each N with $N(Q_1) > 0$. For this, let $S := \{d \in D : N(Q^2 \times \{d\}) > 0\}$ denote the set of distance labels appearing in N. As we have at most k neighbours, Lemma B.14 yields a suitable choice $d =: \mathrm{child}(N)$. Intuitively, this means that d is the child label of a neighbour, but no neighbour is a child of d, which ensures that we never create cycles.

Finally, we write $\mathrm{union}(N) := \bigcup \{S' : (q', S') \in Q_2, N((q', S')) > 0\}$ for the union of all states indicated by phase 2 neighbours. Our transitions δ' are now defined as follows, for all q, N.

$$q, N \mapsto (q', q, \mathrm{root}) \qquad \text{if } N(Q_2) = 0 \text{ and } q' := \delta(q, \mathrm{old}(N)) \in Q_A \quad (1)$$

$$q, N \mapsto (q', q, \mathrm{child}(N)) \qquad \begin{aligned} &\text{if } N(Q_2) = 0 \text{ and } q' := \delta(q, \mathrm{old}(N)) \notin Q_A \\ &\text{and } N(Q_1) > 0 \end{aligned} \quad (2)$$

$$(q, r, i), N \mapsto (q, \mathrm{union}(N) \cup \{q\}) \quad \text{if } N(Q_0) = 0 \text{ and } N(Q^2 \times \{i+1\}) = 0 \quad (3)$$

$$(q, S), N \mapsto A(q, S) \qquad \text{if } N(Q_1) = 0 \text{ and } q \in Q_A \quad (4)$$

$$(q, S), N \mapsto q \qquad \text{if } N(Q_1) = 0 \text{ and } q \notin Q_A \quad (5)$$

Transitions (1) and (2) move the agents from phase 0 to phase 1, executing a neighbourhood transition of δ in the process (synchronously). The move is initiated by agents in Q_A, which pick root as distance label, while the others wait for a neighbour to enter phase 1, at which point they become a child of that neighbour. In phase 1, each node

waits until all children have entered phase 2 (and thus indicate the set of states they have observed), and then executes (3) to move to phase 2, indicating the union of all sets of its children. Finally, the absence-detection initiating nodes move to phase 0 by executing the absence-detection via (4), moving into the appropriate state, while (5) simply moves the other agents to phase 0 without changing their states.

It is crucial that the distance-labels assigned by transition (2) never form a cycle; else we would get a deadlock. Our choice of child ensures that this is the case.

Lemma B.15. *The automaton P' cannot reach a configuration C with a cycle of nodes $(v_0, ..., v_l)$, i.e. $v_0 = v_l$ and v_i is adjacent to v_{i+1} for all i, where each v_i has label $(i \bmod 2k + 1) \in \mathbb{Z}_{2k+1}$.*

Proof. It is only possible for a node to receive a label $d \in \mathbb{Z}_{2k+1}$ via transition (2). (Note that (1) only assigns label root, which cannot be part of a cycle.) However, transition (2) will never close such a cycle, due to the definition of child. $\square$

We will proceed in a similar manner as in the previous section. We want to use Proposition B.4 to construct our reordering, will show that P' is a nonblocking three-phase automaton. Afterwards, we argue that the given reordering is an extension of a run of P.

Lemma B.16. *The automaton P' is a nonblocking three-phase automaton.*

Proof. Again, it is easy to check that P' is a three-phase automaton by inspecting transitions (2)-(5). To show that P is nonblocking the proof is similar to the proof of Lemma B.9.

Let $\pi = (C_0, C_1, ...)$ denote a fair run of P' and $C_i : V \to Q'$ a configuration where two agents are in different phases. We define the phase count pc as for the proof of Proposition B.4, meaning that $\mathsf{pc}(v, j)$ is the number of phase changes of node v until step j.

Let $U := \{u \in V : \mathsf{pc}(u, i) = \min_v \mathsf{pc}(v, i)\}$ denote the set of nodes which have a minimal number of phase changes at step i. If all nodes in U are in phase 0 or phase 2, then selecting any node in U will move it to the next phase via transitions (1), (2) or (4), (5), respectively. Otherwise, we pick any node $u \in U$ and write d for the distance label of u. As u is in phase 1, the only non-silent transition it could perform is (3). If (3) is enabled, then executing it moves u to the next phase, and we are done. If that is not the case, there must be a node u adjacent to v, s.t. v is also in phase 1 and has label $d + 1$. We now set $u := v$ and repeat this process. There are only finitely many nodes and the distance labels form no cycles (due to Lemma B.15), so this must terminate. $\square$

As for weak broadcasts, let π denote a fair run of P'. Using Proposition B.4 we find a specific reordering $\pi_f = (C_0, C_1, ...)$ of π. In particular, every configuration C_i either has all agents in the same phase, or it has agents in at most two phases and at step i one agent moves to the its next phase. We are now going to show that π_f is an extension of a run τ of P.

Again, note that in π_f there are infinitely many configurations C_i where all agents are in the same phase. However, to construct g and the extension τ we will now have to argue slightly differently. If all agents are in phase 1 then there must be at least one agent where transition (3) is enabled. This follows from the same argument as used in the proof of Lemma B.16. If all agents are in phase 2, then either transition (4) or (5) will be enabled for each agent.

The set $I := \{i \in \mathbb{N} : C_i(V) \subseteq Q\}$ of indices i where C_i has only phase 0 agents thus has infinitely many elements, as before. We then define $I' := I \setminus \{i : C_i = C_{i-1}, \exists j > i : C_j \neq C_i\}$ by removing all steps which perform a silent transition and are followed by a non-silent transition from I.

We define the mapping $g : \mathbb{N} \to \mathbb{N}$ as the unique bijection with $g(\mathbb{N}) = I'$ which is strictly increasing, and set $\tau := (K_0, K_1, ...)$ where $K_i := C_{g(i)}$ for all i.

Lemma B.17. π_f *is an extension of* τ.

Proof. The proof is analogous to Lemma B.10, as the removal of silent transitions does not affect the notion of extension. $\qquad\square$

Finally, we argue that τ is a run of P. There is no need to argue that τ is fair, as all runs of P are fair.

Lemma B.18. τ *is a run of* P.

Proof. Fix any $i \in \mathbb{N}$. If $g(i+1) = g(i)+1$ then must have executed a silent transition at step $g(i)$ of π_f, as else it is impossible to remain in phase 0. However, we have removed silent transitions followed by non-silent transition from I' and therefore g, so step i is followed by an infinite sequence of silent transitions $C_{g(i)} = C_{g(i)+1} = ...$. We know that π and thus π_f are fair (w.r.t. an adversarial scheduler), so this means that transition (1) is not enabled for any node. Therefore $K_i(V) \cap Q_A = \emptyset$ and Definition 4.8 states that P hangs in this case, so $K_{i+1} = K_i$ should hold, which is what we have.

Otherwise, there is a $g(i) < t_1, t_2 < g(i+1)$ s.t. C_{t_1} (C_{t_2}) has only agents in phase 1 (phase 2). First, every node executes either (1) or (2), effectively moving to configuration C' with $(C'(v), \cdot, \cdot) = C_{t_2}(v)$ for $v \in V$. In particular, transitions (1) and (2) use the phase 0 state of each agent, so this executes a synchronous neighbourhood transition (i.e. one with selection V).

Let S denote the set of agents executing transition (1) between steps $g(i)$ and t_1 in π_f. A brief look at transition (1) reveals that $S = (C')^{-1}(Q_A)$ contains precisely the agents which are in absence-detection initiating states after executing the synchronous neighbourhood transition.

Now we simply note that K_{i+1} is the result of executing a weak absence-detection transition on C' using selection S. Here, we observe that every node $v \notin S$ must pick a child label of a neighbour u in transition (2). Agent u will only execute transition (3) once v is in phase 2, so the information of v will be propagated to u, then to a parent of u, and so on, until it reaches an agent in S. The agents in S then perform transition (4) and move according to the weak absence-detection transition, while all other nodes execute (5) and remain in their original state. $\qquad\square$

B.4. Simulating Rendez-vous Transitions

Definition B.19. A *graph population protocol* is a tuple (Q, δ), where Q is a finite set of *states*, and $\delta : Q^2 \to Q^2$ is a set of *transitions* that describes the rendez-vous interactions between two adjacent nodes. In particular, if $\delta(p, q) = (p', q')$, then we write $p, q \mapsto p', q'$. Further, let $\delta_1(p, q) = p'$ and $\delta_2(p, q) = q'$ be functions for the first and second component of δ. The definitions of configurations and runs are equivalent to the ones of distributed machines. Selections are ordered pairs of adjacent nodes, i.e. the set of possible selections is $\{(u, v) : \{u, v\} \in E\}$. If $(u, v) \in V^2$ is the selection in some configuration C, then the successor configuration is C' with $C'(u) := \delta_1(C(u), C(v))$, $C'(v) := \delta_2(C(u), C(v))$ and $C'(x) := C(x)$ for all $x \in V \setminus \{u, v\}$. We require the schedules the be pseudo-stochastic, so every finite sequence of selections has to appear infinitely often in a schedule.

Lemma 4.10. *Every graph population protocol is simulated by some* `DAF`*-automaton.*

Proof. Let $P = (Q, \delta)$ be a population protocol on graphs. We define a `DAF`-automaton $M = (Q', \delta')$ that simulates P. We set the counting bound $\beta := 2$. Let $Q_{\overline{\Xi}} = Q$, $Q_{\mathbf{Q}} = Q \times \{\mathbf{Q}\}$, $Q_{\mathbf{\mathfrak{H}}} = Q \times \{\mathbf{\mathfrak{H}}\}$ and $Q_{\checkmark} = Q \times \{\checkmark\} \times Q$. We define $Q' := Q_{\overline{\Xi}} \cup Q_{\mathbf{Q}} \cup Q_{\mathbf{\mathfrak{H}}} \cup Q_{\checkmark}$. Intuitively, each node stores its state in the original protocol in the first component and has a status that helps to simulate rendez-vous transitions. The status can be "waiting" ($\overline{\Xi}$), "searching" ($\mathbf{Q}$), "answering" ($\mathbf{\mathfrak{H}}$) or "confirming" ($\checkmark$) and initially every node is waiting. Additionally, a confirming node stores the state it would have after the rendez-vous transition is completed. This is necessary, because the other node will performs its part of the rendez-vous interaction first.

Now will define the transition function δ' of M, for states $q, q', q'' \in Q$ and neighbourhood $N \in [\beta]^{Q'}$. Let $N(\overline{\Xi}) := \sum_{q \in Q} N(q)$ be the number of detectable waiting neighbours. Further, we use the auxiliary function $f(N)$ to denote the unique non-waiting neighbour, if any, i.e. we set $f(N) := x$ if $N(\overline{\Xi}) = |N| - 1$ and $N(x) = 1$. If no such neighbour exists, we set $f(N) := \overline{\Xi}$ if $N[\overline{\Xi}] = |N|$ (all neighbours are waiting), and $f(N) := \bot$ otherwise. Figure 4 contains the formal definition of δ and a diagram that visualises how nodes change their status.

Intuitively, the simulation of a rendez-vous transition $p, q \mapsto p', q'$ starts with a waiting ($\overline{\Xi}$) agent with original state p that only sees waiting nodes. This agent searches for a partner by changing its status to $\mathbf{Q}$. Then, its waiting neighbours can answer by changing to $\mathbf{\mathfrak{H}}$ if they detect exactly one search. If the searching agent detects exactly one answer, it confirms by changing to $\checkmark$ while remembering the state it would have after the rendez-vous with the answering node. If the answering node with original state q sees exactly one confirmation, it applies the state change (q to q') and waits. Then, the confirming node detects that the answering node is now waiting and applies the state change it remembered (p to p'). However, once a node detects an irregularity in the simulation (e.g. more than one non-waiting neighbour) it cancels the interaction by changing its state to $\overline{\Xi}$.

We still have to show that M simulates P. We call a change to the original state of a node a *state change*. In other words, neighbourhood transitions that change the first component of a nodes state perform a state change. We will now argue, that state

$$q, N \mapsto (q, \mathbf{Q}) \qquad \text{for } f(N) = \boxed{\mathbf{Z}}$$
$$q, N \mapsto (q, \circlearrowleft) \qquad \text{for } f(N) = (q', \mathbf{Q})$$
$$(q, \mathbf{Q}), N \mapsto (q, \checkmark, \delta_1(q, q')) \qquad \text{for } f(N) = (q', \circlearrowleft)$$
$$(q, \circlearrowleft), N \mapsto \delta_2(q', q) \qquad \text{for } f(N) = (q', \checkmark, q'')$$
$$(q, \checkmark, q'), N \mapsto q' \qquad \text{for } f(N) = \boxed{\mathbf{Z}}$$

Right side diagram: states shown with status symbols (confirming $\mathbf{Q}$, waiting $\boxed{\mathbf{Z}}$, answering $\checkmark$, and $\circlearrowleft$). Edges labelled: "$1\times\circlearrowleft$, rest $\boxed{\mathbf{Z}}$", "all $\boxed{\mathbf{Z}}$", "$1\times\checkmark$, rest $\boxed{\mathbf{Z}}$", "all $\boxed{\mathbf{Z}}$", "$1\times\mathbf{Q}$, rest $\boxed{\mathbf{Z}}$".
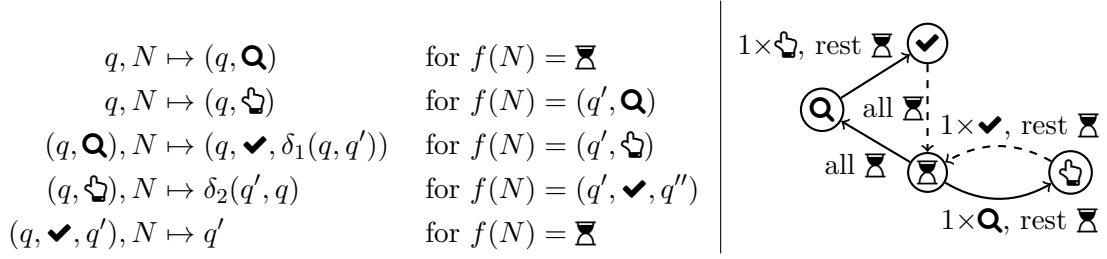
Figure 4: Neighbourhood transitions that simulate rendez-vous interactions. The left side show the formal definition of the transition function δ'. For all inputs x, N where $\delta'(x, N)$ is undefined, the status is set to waiting ($\boxed{\mathbf{Z}}$) by changing to the original state saved in the first component of x. The right side visualises the neighbourhood transitions as a graph. States in the diagram only show the status of a node. A node only change its status by following an edges in the diagram, if its neighbourhood satisfies the condition on the edge. If a node is selected and no edge can be followed, it instead changes its status to waiting ($\boxed{\mathbf{Z}}$). The edges that apply the rendez-vous transition δ are drawn dashed.

changes only occur in pairs and that they simulate the rendez-vous transitions in δ. First note, that from a configuration C where two nodes u, v and their neighbours are waiting, scheduling the sequence u, v, u, v, u correctly applies the state changes $\delta_1(C(u), C(v))$ for u and $\delta_2(C(u), C(v))$ for v. For a node u to enter the confirming state, it must have exactly one answering neighbour v and all other neighbours must be waiting. u cannot perform its state change before v because it needs to wait until all nodes are waiting. Once the answering agent v performs its state change, all of u's neighbours are waiting and cannot change their status because they see that u is confirming. Thus, the next time u is scheduled, it must perform its state change. Further, v can only perform the state change if it sees exactly one confirming state and all other of v's neighbours are waiting. Thus, once one of nodes in the rendez-vous interaction performs the state change, the full rendez-vous interaction will be performed. Further, it is impossible for more than two nodes to interact simultaneously, because selection is exclusive and whenever a node detects more than one non-waiting neighbour, it cancels the interaction and waits.

Next, we need to reorder a given run π' of M such that it is an extension of some run π in P. For this, we make sure that after an answering node performs the state change, the corresponding confirming node is scheduled immediately so that it can perform its state change. Intuitively, the reordering makes sure that the state changes in the simulation of two different rendez-vous transitions are not executed in an interleaving manner. Thus, the reordered run is indeed an extension of a run in P where the state changes of rendez-vous interactions happen atomically. The reordering is valid, because after the answering node performs the state change, the nodes in the neighbourhood of the confirming node are all waiting and they cannot change their status because they see a confirming node. Thus, scheduling the confirming agent earlier in the reordered run does not interfere with the neighbourhood transitions that were executed between the

two state changes in π'.

Lastly, we need to argue about fairness. Let π be the simulated run of P for some fair run π' of M. Let C be some configuration that is visited infinitely often in π. Further, let $S = (u_1, v_1), \cdots, (u_k, v_k) \in (V \times V)^*$ be a finite sequence of selections such that scheduling S in C leads to come configuration C_f. As C is visited infinitely often, there are infinitely many configurations $C'_1, C'_2, \cdots \in \pi'$ with $C \sim_Q C'_i$ for all $i > 0$. Because there are only finitely many different configurations for a given graph, there is at least one configuration C' that is visited infinitely often in π' such that $C \sim_Q C'$. C' can reach a configuration $C'' \sim_Q C'$ where all nodes are waiting by scheduling all confirming nodes, then all answering nodes and lastly all searching nodes. C'' can reach a configuration $C'_f \sim_q C_f$ by simulating all selections (u_i, v_i) of S one after the other by scheduling u_i, v_i, u_i, v_i, u_i. Because π' is fair, C'_f is visited infinitely often in π'. Therefore, C_f is visited infinitely often in π and π is fair. $\qquad\square$

C. Proofs of Section 5

As mentioned in the introduction, we are interested in labelling properties.

Definition C.1. For every labelled graph $G = (V, E, \lambda)$ over the finite set of Labels $\mathcal{L}$ we write L_G for the multiset of labels occurring in G, i.e. $L_G : \mathcal{L} \to \mathbb{N}, L_G(x) = |\{v \in V : \lambda(v) = x\}|$ for all labels x. We call L_G the label count of G.

A graph property φ is called a labelling property if for all labelled graphs G, G' with $L_G = L_{G'}$ we have $\varphi(G) = \varphi(G')$. In such a case we also write $\varphi(L_G)$ instead of $\varphi(G)$.

In this section, we will use L for multisets of labels.

C.1. `DaF` only decides trivial properties

Proposition C.2. *Let φ be a labelling property decided by a `DaF`-automaton in the unrestricted set of graphs or in the set of k-degree-bounded graphs. Then φ is trivial, i.e. either always false or always true.*

Proof. Assume that φ is not always false, i.e. $\varphi(L) = 1$ for some L. We have to prove that φ is always true, i.e. $\varphi(L') = 1$ for all labelling multisets L'. Let L' be any labelling multiset. By our general assumption, network graphs have at least 3 nodes, i.e. $|L|, |L'| \geq 3$. Since φ is a labelling property, we can choose the underlying graph. Let G be the cycle with $|L|$ nodes labelled with L, and let G' be the cycle with $|L'|$ nodes labelled with L'. By Lemma 3.1, we cannot distinguish G and G' and therefore have $\varphi(L') = \varphi(L) = 1$ as claimed. This also holds in the k-degree-bounded case since the graph constructed in the proof of Lemma 3.1 is k-degree-bounded, if both G and G' were. $\qquad\square$

C.2. `DAf` decides at most $\mathsf{Cutoff}(1)$

Proposition C.3. *Let φ be a labelling property decided by a `DAf`-automaton. Then $\varphi \in \mathsf{Cutoff}(1)$.*

Proof. Let A be a `DAf`-automaton with $\varphi_A = \varphi$. Let $K = \beta + 1$ be as in Lemma 3.4, i.e. the natural number such that $\varphi(L) = \varphi(\lceil L \rceil_K)$ for all labelling multisets L. In addition, we know that φ is closed under scalar multiplication by Corollary 3.3. We use this corollary with $\lambda = K$ to scale up L with the factor K, then cut if off at K and scale down again.

Formally, we start by proving $\lceil \lambda \cdot L \rceil_\lambda = \lambda \cdot \lceil L \rceil_1 = \lceil \lambda \cdot \lceil L \rceil_1 \rceil_\lambda$ for all $\lambda \in \mathbb{N}$ by case distinction, namely if some label x occurs, then all those three functions equal λ, and otherwise they all equal 0.

We use this to obtain the following chain of equalities:

$$\varphi(L) \stackrel{\text{C3.3}}{=} \varphi(K \cdot L) \stackrel{\text{L3.4}}{=} \varphi(\lceil K \cdot L \rceil_K) = \varphi(K \cdot \lceil L \rceil_1) = \varphi(\lceil K \cdot \lceil L \rceil_1 \rceil_K)$$
$$\stackrel{\text{L3.4}}{=} \varphi(K \cdot \lceil L \rceil_1) \stackrel{\text{C3.3}}{=} \varphi(\lceil L \rceil_1)$$

$\square$

C.3. `dAf` can decide $\mathsf{Cutoff}(1)$

Proposition C.4. *`dAf`-automata can decide all labelling properties $\varphi \in \mathsf{Cutoff}(1)$.*

Proof. Let $\varphi \in \mathsf{Cutoff}(1)$. Let $x_1, ..., x_n$ be the variables occurring in φ. Then φ corresponds to a subset M in $\{0,1\}^{\{1,...,n\}}$, describing whether we accept if exactly the variables with indices $i \mapsto 1$ occur. By [16, Proposition 12], `dAf`-automata can decide the language B of graphs with a black node, i.e. the labelling predicate $\varphi(x, y) \Leftrightarrow x \geq 1$. On the level of subsets M, this means the set

$$M_i := \{f : \{1, ..., n\} \to \{0,1\} : f(i) = 1\}$$

of all functions with $i \mapsto 1$ can be decided. We can write every subset M via unions, complements and intersections of sets M_i. This corresponds to writing φ as a boolean combination of $x_i \geq 1$, which can be decided. $\square$

C.4. `dAF` can decide exactly Cutoff

Lemma C.5. *For every property $\varphi : \mathbb{N}^l \to \{0,1\}$ with $\varphi(x, y_1, ..., y_{l-1}) \Leftrightarrow x \geq k$ for some $k \in \mathbb{N}$ there is a `dAF`-automaton deciding φ.*

Proof. We construct a `dAF`-automaton $P = (Q, \emptyset, I, O)$, which we will augment with weak broadcast transitions. As states we use $Q := \{0, 1, ..., k\}$, the input mapping is given by $I(x) := 1$ and $I(y_1) = ... = I(y_{l-1}) = 0$, and the set of accepting states is $O := \{k\}$. We add the following broadcasts, with $i = 1, ..., k-1$.

$$i \mapsto i, \{i \mapsto i + 1\} \qquad\qquad\qquad \langle\text{level}\rangle$$
$$k \mapsto k, \{q \mapsto k : q \in Q\} \qquad\qquad\qquad \langle\text{accept}\rangle$$

Using Lemma 4.7 we get an equivalent `dAF`-automaton.

Let C_0 denote an initial configuration with $c := |C_0^{-1}(1)|$ set to the number of agents starting in state 1. It is easy to see that C_0 is accepting iff it can reach a configuration C with $k \in C(V)$, i.e. at least one agent in state k. Of course, $\langle\mathsf{accept}\rangle$ cannot be used to reach k as the initiator is already in state k, so we now consider only configurations C reachable by $\langle\mathsf{level}\rangle$.

It is only possible for an agent to go from state i to $i+1$ by receiving broadcast $\langle\mathsf{level}\rangle$ initiated by an agent in state i, for $i = 1, ..., k-1$. The initiator remains in state i, so we have that $i+1 \in C(V)$ implies $i \in C(V)$. Therefore $k \in C(V)$ implies $\{1, 2, ..., k\} \subseteq C(V)$ and thus at least k agents have started in state 1, i.e. $c \geq k$, as it is not possible to leave state 0 via $\langle\mathsf{level}\rangle$.

To summarise, the protocol accepts only initial configuration which should be accepted. It remains to show that the converse holds as well, so we require $c \geq k$ and set C to an arbitrary configuration reachable from C_0 with only $\langle\mathsf{level}\rangle$. We have pseudo-stochastic fairness, so it is enough to show that C can reach a configuration C' with $k \in C'(V)$.

Let $m_i := |C^{-1}(i)|$ denote the number of agents in state i, for $i = 1, ..., k$. We define an ordering on the set of configurations by ordering the tuples $(m_k, m_{k-1}, ..., m_1)$ lexicographically. If C does not have a node in state k, then it has $c \geq k$ occupying states 1 to $k-1$, i.e. $m_1 + ... + m_{k-1} = k$ and, by pigeonhole principle, there is some $1 \leq j < k$ with $m_j \geq 2$. By executing transition $\langle\mathsf{level}\rangle$ on one of those agents exclusively, at least one agent moves to state $j+1$. Hence the resulting configuration is strictly larger w.r.t. our ordering. There are only finitely many configurations with n agents, so we can repeat this procedure until at least one agent has state k, thereby proving that C reaches some accepting configuration. □

Proposition C.6. *The set of labelling properties decided by* `dAF`*-automata is precisely* Cutoff.

Proof. By Lemma 3.5, the expressive power is contained in Cutoff.

Now let $\varphi \in$ Cutoff. Let $K \in \mathbb{N}$ be as in the definition of Cutoff. Let $x_1, ..., x_n$ be the variables occurring in φ. Then φ corresponds to a $M \in [K]^{\{1, ..., n\}}$ of accepted cutoffs. If we can decide all formulas corresponding to 1-element subsets of $\{0, 1, ..., K\}^{\{1, ..., n\}}$, then we can decide φ, since φ can be written as a disjunction of such formulas. Let M be such a 1-element subset, write this element as $f : \{1, ..., n\} \to \{0, 1, ..., K\}$. Let $S \subseteq \{1, ..., n\}$ be the set of indices i with $f(i) = K$. The formula corresponding to M is

$$\bigwedge_{i \notin S} (x_i \geq f(i) \land \neg(x_i \geq f(i) + 1)) \land \bigwedge_{i \in S} (x_i \geq f(i)),$$

which can be decided since by Lemma C.5, we can compute $x_i \geq f(i)$, and the set of decidable properties is closed under boolean combinations. □

C.5. `DAF` can decide exactly the labelling properties in NL

Lemma 5.1. *`DAF`-automata decide exactly the labelling properties in* NL.

Proof. As argued in the main paper, `DAF`-automata can decide at most the predicates in `NL`. We now restate the construction from the proof sketch for clarity.

It is known that strong broadcast protocols decide exactly the predicates in `NL` [11, Theorem 15]. Therefore, it suffices to show that for every strong broadcast protocol there is an equivalent `DAF`-automaton. By Lemma 4.7 `DAF`-automata can simulate weak broadcasts, and so, loosely speaking, the task is to simulate strong broadcasts with weak ones.

Let $P = (Q, \delta, I, O)$ be a strong broadcast protocol. We start with a graph population protocol $P_{\text{token}} := (Q_{\text{token}}, \delta_{\text{token}})$, with states $Q_{\text{token}} := \{0, L, L', \bot\}$ and rendez-vous transitions δ_{token} given by

$$(L, L) \mapsto (0, \bot), \quad (0, L) \mapsto (L, 0), \quad (L, 0) \mapsto (L', 0) \qquad \langle\text{token}\rangle$$

Now we construct a `DAF`-automaton $P'_{\text{token}} = (Q'_{\text{token}}, \delta'_{\text{token}})$ simulating P_{token} using Lemma 4.10, and combine it with P by setting $P_{\text{step}} := P'_{\text{token}} \times Q + \langle\text{step}\rangle$, where $\langle\text{step}\rangle$ is a weak broadcast defined as

$$(L', q) \mapsto (L, q'), \{(t, r) \mapsto (t, f(r)) : (t, r) \in Q'_{\text{token}} \times Q\} \qquad \langle\text{step}\rangle$$

for each broadcast $q \mapsto q', f$ in δ. Finally, let $P'_{\text{step}} = (Q'_{\text{step}}, \delta'_{\text{step}})$ be a `DAF`-automaton simulating P_{step}, which exists by Lemma 4.7.

Intuitively, agents in states L, L' have a *token*. If we could ensure that initially there is only one token in L, L', then we would be done. Indeed, in this case at each moment only the agent with the token can move; if in L, it initiates a (simulated) rendez-vous transition, and if in L', a weak broadcast. Since no other agent is executing a weak broadcast at the same time, the weak broadcast is received by all agents, and has the same effect as a strong broadcast.

We cannot ensure that initially there is only one token, but if the computation starts with more than one, then two tokens eventually meet using transition $\langle\text{token}\rangle$ and an agent moves into the *error state* $\bot$. We design a mechanism to restart the computation after this occurs, now with fewer agents in state $(L, \cdot)$, guaranteeing that eventually the computation is restarted with only one token. For this we again add an additional component to each state and consider the protocol $P_{\text{reset}} := P'_{\text{step}} \times Q + \langle\text{reset}\rangle$, where $\langle\text{reset}\rangle$ are the following broadcast transitions, for each $q, q_0 \in Q$.

$$((\bot, q), q_0) \mapsto ((L, q_0), q_0), \{(r, r_0) \mapsto ((0, r_0), r_0) : r \in Q'_{\text{step}}, r_0 \in Q\} \qquad \langle\text{reset}\rangle$$

For P_{reset} we define the input mapping $I_{\text{reset}}(x) := ((L, I(x)), I(x))$ and the set of accepting states $O_{\text{reset}} := \{((r, q), q_0) : q \in O, q_0 \in Q, r \in \{0, L\}\}$. Using Lemma 4.7 (and Lemma 4.4) we get a `DAF`-Automaton equivalent to P_{reset}, so it suffices to show that P_{reset} is equivalent to P.

As a technical aide to state the proof, we introduce another graph population protocol $P^*_{\text{token}} := (Q_{\text{token}}, \delta^*_{\text{token}})$, where $\delta^*_{\text{token}} := \{(L, L) \mapsto (0, \bot), (0, L) \mapsto (L, 0)\}$. Essentially, we want to ignore the difference between L and L'. For this, we consider the mapping $g : Q'_{\text{token}} \to Q'_{\text{token}}$, which maps $g(L') := L$ and all other states to themselves. We

extend g to Q_{step} by applying it only to the first component, i.e. $g((q, r)) := g(q)$ for $(q, r) \in Q_{\text{step}}$, and then to configurations $C : V \to Q_{\text{step}}$ and runs π of P_{step} in the obvious manner.

Let π denote a fair run of P_{step}. If we only consider the first component of the states in π, this is essentially a run of P'_{token}, except that at some steps an agent transitions from L' to L using $\langle\text{step}\rangle$. It is not guaranteed that a simulation continues to work under these conditions, but the construction of Lemma 4.10 does not rely on the non-intermediate states remaining unchanged between transitions. This means that $g(\pi)$ has a reordering which is an extension of a fair run of P^*_{token}.

Let $C_0^* : V \to Q$ denote an initial configuration of P, and let $\pi := (C_0, C_1, ...)$ denote a run of P_{step}, starting in a configuration C_0 with $C_0(v) = (0, C_0^*(v))$ or $C_0(v) = (L, C_0^*(v))$ for all nodes v.

If C_0 has $k > 1$ tokens, then, we claim, π will reach a configuration with an agent in an error state, and the set $S := \bigcup_i C_i^{-1}(Q \times \bot)$ of agents to ever reach an error state has size at most $k - 1$. We will refer to this property as $A(\pi)$. Crucially, if $A(\pi)$ holds for any run π, then it also holds for any reordering of π, and it also holds for any extension of π.

As we argued before, $g(\pi)$ is a reordering of an extension of a fair run of P^*_{token}. Moreover, this projection does not affect A. So it is sufficient to show $A(\tau)$ for all fair runs τ of P^*_{token}, which follows immediately from its definitions.

Let $\pi' := (C'_0, C'_1, ...)$ denote a fair run of P_{reset} with initial configuration C'_0 defined similar to C_0, so $C'_0(v) = ((0, C_0^*(v)), C_0^*(v))$ or $C'_0(v) = ((L, C_0^*(v)), C_0^*(v))$ for all nodes v, and the latter holds for exactly $k > 1$ nodes. (Note that initial configurations of P_{reset} always have $k = n$.) As $A(\pi)$ holds for all fair runs π of P_{step} as defined above, we find that a fair run π' where $\langle\text{reset}\rangle$ is never executed will necessarily enable it once. But, as per Definition 4.5, all states $((\bot, \cdot), \cdot)$ are broadcast-initiating, so they cannot execute a neighbourhood transition and can only change their state via $\langle\text{reset}\rangle$. This has to occur eventually, so let step i denote the first execution of $\langle\text{reset}\rangle$.

Again, due to A, before step i the number of agents in a state in $(\{\bot\} \times Q) \times Q$ is at most $k - 1$. So we get $C'_i = ((0, C_0(v)), C_0(v))$ or $C'_i(v) = ((L, C_0(v)), C_0(v))$ for all nodes v, and the latter holds for at most $k - 1$ (but not zero) nodes. (Recall that a weak broadcast might be executed simultaneously by multiple agents, so it is possible to end up with more than one token after executing $\langle\text{reset}\rangle$.) By induction, we eventually find a suffix of the run starting in a configuration C'_0 as defined above with $k = 1$, i.e. exactly one agent is holding a token.

As we argued for A, no agent can ever reach an error state from such an initial configuration, so transition $\langle\text{reset}\rangle$ will never be executed and it suffices to show that any fair run $\pi = (C_0, C_1, ...)$ of P_{step} stabilises to the correct consensus, where $C_0(v) = (L, C_0^*(v))$ for some node v and $C_0(u) = (0, C_0^*(u))$ for all other nodes $u \neq v$.

First, we argue that there is always at most one agent holding a token. Again, applying g yields a reordering of an extension of a run of P^*_{token}, and it is clear that the property holds for any run of P^*_{token} and any extension $\tau = (K_0, K_1, ...)$ of such a run (with initial configuration K_0 defined s.t. $C_0(v) \in \{K_0(v)\} \times Q$ for all v). However, we still need to show that any reordering τ_f of τ also fulfils the property. It is only possible for a node v to receive a token by moving from state 0 or an intermediate state to L. If this happens,

say, at step i in τ, then v or an adjacent node u must have left $\{L, L'\}$ at a step j directly before i, i.e. a step $j < i$ s.t. in configurations $K_{j+1}, K_{j+2}, ..., K_{i-1}$ there are no agents with a token. For the reordering we then have $f(j) < f(j)$, so the token leaves u before entering v as well in τ_f.

This means that transition $\langle\text{step}\rangle$ cannot be executed by multiple agents simultaneously and it thus updates the states in the same manner as in P. Finally, it remains to show that π does so in a pseudo-stochastic manner, for which it is sufficient to prove that any configuration C_i after executing $\langle\text{step}\rangle$ can, for any node v, reach a configuration C' where v has the token without executing $\langle\text{step}\rangle$.

Intuitively, this clearly holds, based on transitions $\langle\text{token}\rangle$. To make this formally precise, we reference the specific construction of Lemma 4.10. Starting with C_i we repeatedly select agents in intermediate states (and execute the corresponding transition) until none are left. This will never select the (unique) node v with $C_i(v) = (L, \cdot)$, and it will terminate, due to the transitions of Lemma 4.10. Thus we reach a configuration C' with $C'(v) = (L, \cdot)$ and all other agents $u \neq v$ have $C'(u) = (0, \cdot)$. (We have already argued that it is not possible to reach a state with more than one token.) As C' contains no intermediate states, it is easy to see that there is a sequence of neighbourhood transitions to move the token to any node. $\qquad\square$

D. Proofs of Section 6

D.1. `dAf` can only decide $\mathsf{Cutoff}(1)$

Proposition D.1. *The set of labelling properties decided by `dAf`-automata in the k-degree-bounded case for $k \geq 3$ is precisely $\mathsf{Cutoff}(1)$.*

Proof. We know that $\mathsf{Cutoff}(1)$ is contained in the expressive power of `dAF` for k-degree-bounded graphs, since we can compute predicates in $\mathsf{Cutoff}(1)$ even in the unrestricted set of graphs.

Now let φ be a property decided by some `dAf`-automaton M. We claim that for every multiset L and every label x with $L(x) \geq 1$ we have $\varphi(L) = \varphi(L + x)$.

Proof of claim: since φ is a labelling property, we can choose the underlying graph. Let $G = (V, E, \lambda)$ be a line labelled with the set L and the label x on the first end. Define the graph $G' = (V', E', \lambda')$ by copying G and adding a extra node, which is labelled with x and connected to the second node only. Since M is consistent, M accepts G if and only if the synchronous run ρ on G is accepting and it accepts G' if and only if the synchronous run ρ' on G' is accepting. It follows by induction that every node of the graph G is always in the same state in both runs, and that the extra node is always in the same state as the first end.

This shows that ρ is accepting if and only if ρ' is accepting. Therefore G is accepted if and only if G' is accepted, proving the claim.

Now we use the claim to prove the proposition. Let L be some multiset. We have to prove that $\varphi(L) = \varphi(\lceil L \rceil_1)$. For this, we write $L = \lceil L \rceil_1 + x_1 + ... + x_n$ with $x_i(\lceil L \rceil_1) \geq 1$ and use the claim a total of n times. $\qquad\square$

D.2. `dAF` and `DAF` decide exactly the labelling properties in $\mathsf{NSPACE}(n)$

Proposition D.2. *A labelling property φ can be decided by a `dAF`-automaton in the k-degree-bounded case if and only if $\varphi \in \mathsf{NSPACE}(n)$.*

A labelling property φ can be decided by a `DAF`-automaton in the k-degree-bounded case if and only if $\varphi \in \mathsf{NSPACE}(n)$.

Proof. By [16, Proposition 22], the expressive power of `dAF`-automata is equal to the expressive power of `DAF`-automata in the k-degree-bounded case. It is therefore enough to consider `DAF`.

We start by proving that labelling properties $\varphi \in \mathsf{NSPACE}(n)$ can be decided. By [12], when restricting to k-degree-bounded graphs, graph population protocols can decide all symmetric properties $\varphi \in \mathsf{NSPACE}(n)$, in particular all labelling properties $\varphi \in \mathsf{NSPACE}(n)$, since they are by definition invariant under rearranging the labels. By Lemma 4.10, all properties decidable by graph population protocols can also be decided by `DAF`-automata.

Now let φ be a labelling property decided by a `DAF`-automaton M with counting bound β. We have to prove that φ can be decided by a non-deterministic Turing machine with linear space. Since every node uses constant space and we have a linear number of nodes, a Turing machine with linear space can save configurations of our automaton M. We claim that checking whether two configurations C, C' fulfil $C \to C'$ can be checked in $\mathsf{NSPACE}(n)$. For this, the Turing machine guesses for each node whether it has to be selected or not, and then checks for every node v whether

$$C(v) = C'(v) \qquad\qquad \text{if } v \notin S$$
$$\delta(C(v), \lceil C(N(v)) \rceil_\beta) = C'(v) \qquad\qquad \text{if } v \in S,$$

i.e. the definition of the semantics. Since $C \to C'$ can be checked in $\mathsf{NSPACE}(n)$, $C \to^* C'$ also can. For this, the Turing Machine does the following $(|Q|)^{|V|}$ times (upper bound on number of configurations): guess a configuration C'' and check $C \to C''$. Overwrite C with C''. If $C'' = C'$ accept, if we finish the loop without this occurring reject.

Now we use Immerman–Szelepcsényi theorem in the general version to obtain that $C \not\to^* C'$ can also be checked in $\mathsf{NSPACE}(n)$. Due to the automaton M using pseudo-stochastic fairness, we accept from some initial configuration C_0 if and only if there exists a configuration C fulfilling the following three conditions:

1. $C_0 \to^* C$.

2. C is accepting.

3. For all non-accepting configurations C', we have $C \not\to^* C'$.

We can check this in $\mathsf{NSPACE}(n)$ by guessing the configuration C and checking the reachability conditions as described above. $\qquad\square$

D.3. `DAf` can decide majority

The proof of Lemma 6.1 will use the following lemma, which encapsulates the main argument.

Lemma D.3. *Let $\pi = (C_0, C_1, ...)$ denote a fair run of P_{cancel}. There are only finitely many C_i with $C_i(V) \cap \{k+1, ..., E\} \neq \emptyset$ and $C_i(V) \cap \{-E, ..., 0\} \neq \emptyset$.*

Proof. First, we note $C_i(V) \subseteq S \Rightarrow C_{i+1}(V) \subseteq S$ for $S = \{0, ..., E\}$, $S = \{1, ..., E\}$, and $S = \{-E, ..., k\}$. In particular, the latter two imply that it suffices to show that there exists an i with $C_i(V) \cap \{k+1, ..., E\} = \emptyset$ or $C_i(V) \cap \{-E, ..., 0\} = \emptyset$.

Our proof will proceed by first showing that $C_i(V) \cap \{k+1, ..., E\} \neq \emptyset$ and $C_i(V) \cap \{-E, ..., -1\} \neq \emptyset$ cannot both hold for all i. Afterwards, we will argue that $C_i(V) \cap \{k+1, ..., E\} \neq \emptyset$ and $0 \in C_i(V)$ also cannot always hold, thus completing the proof.

Assume $C_i(V) \cap \{k+1, ..., E\} \neq \emptyset$ and $C_i(V) \cap \{-E, ..., -1\} \neq \emptyset$ for all i. We fix an i, and let $S_0(C_i) := C_i^{-1}(\{-E, ..., 0\})$ denote the set of agents with nonpositive contribution in C_i. Due to our assumption, $S_0(C_i)$ is nonempty. We write $S_d(C_i)$ for the set of nodes with distance d to $S_0(C_i)$, for $d = 1, ..., n$, and define $\lambda_d(C_i) := \sum_{v \in S_d(C_i)} C_i(v)$ as the sum of contributions of S_d. Finally, we set $\lambda(C_i) := (\lambda_0(C_i), ..., \lambda_n(C_i))$.

We now claim that $\lambda(C_i) < \lambda(C_{i+1})$ for each i, using lexicographical ordering, which is a contradiction, as there are only finitely many different configurations. To show the claim, we split the transition from C_i to C_{i+1} into a set of pairwise transactions $U \subseteq V \times V$, s.t. $C_{i+1}(v) = C_i(v) - |U \cap \{v\} \times V| + |U \cap V \times \{v\}|$ for each node v, and $C_i(u) > k \wedge C_i(v) \leq k$ or $C_i(u) \geq -k \wedge C_i(v) < -k$ for all $(u, v) \in U$. Intuitively, $(u, v) \in U$ means that u sends one unit to v.

We always have $C_i(u) > C_i(v)$ for $(u, v) \in U$, so $\lambda_0(C_i) \leq \lambda_0(C_{i+1})$. If there exist adjacent nodes $u, v \in V$ with $C_i(u) < 0 < C_i(v)$ and $(v, u) \in U$ then we have $\lambda_0(C_i) < \lambda_0(C_{i+1})$ and our claim follows. Hence we will now exclude this case. In particular, we thereby exclude the possibility of a node v leaving S_0, i.e. $v \in S_0(C_i) \setminus S_0(C_{i+1})$.

Let $U^+ := \{(u, v) \in U : u, v \notin S_0(C_i)\}$ denote the set of transitions where neither node has negative contribution. We pick d, u where $u \in S_d$ and $C_i(u) > k$ s.t. d is minimal. It is clear that $d > 1$ holds, as else there would be a transaction from u to a node in S_0. There is some node v adjacent to u in S_{d-1} which, by choice of u, fulfils $C_i(u) \leq k$. Therefore $(u, v) \in U^+$. In particular, u sends one unit to v, thereby increasing S_{d-1}.

Moreover, all transactions $(u', v') \in U^+$ have $C_i(u') > k$, so it is not possible for any such transaction to decrease any $S_{d'}$ with $d' < d$. Neither can such a transaction change S_0. Therefore we find that the transactions in U^+ strictly increase λ, without affecting S_0.

Let C denote the configuration where the transaction in U^+ have been executed, i.e. $C(v) := C_i(v) - |U^+ \cap \{v\} \times V| + |U^+ \cap V \times \{v\}|$. From the above considerations we get $\lambda(C) > \lambda(C_i)$ and $S_0(C) = S_0(C_i)$. The transactions in $U \cap S_0(C) \times S_0(C)$ do not change λ_0 and do not affect S_0 (a node could go from $-k - 1$ to -1, but not further), so we now set C' to the configuration after executing those.

Finally, consider a transaction $(u, v) \in U$ with $v \in S_0(C) \subseteq S_0(C')$ and $u \notin S_0(C)$. If $C'(u) > 0$, then the transactions would strictly increase λ_0, without changing S_0. If $C'(u) < 0$, then neither λ nor S_0 would change. Otherwise, the contribution of u becomes -1, in which case u would enter S_0, and λ_0 would remain unchanged. As a consequence of u entering S_0, the distance between some other nodes and S_0 might decrease, but that can only increase λ, as all nodes outside of S_0 have nonnegative contribution. We can

now proceed inductively, by updating C' corresponding to (u, v).

This concludes the first part of the proof. It remains to argue that $C_i(V) \cap \{k + 1, ..., E\} \neq \emptyset$ and $0 \in C_i(V)$ cannot hold for all i. We argue analogously to before and assume the contrary. Then we set $S_0(C_i) := C_i^{-1}(0)$ and define S_d, λ_d and λ as before. It is not possible for a node to enter S_0, so a node can leave S_0 only finitely often. Choosing an i large enough, the set S_0 thus does not change. Finally, we again pick d, u where $u \in S_d$ and $C_i(u) > k$ s.t. d is minimal, and see that λ_{d-1} and thus λ must increase at each step, which is a contradiction. $\qquad \square$

Lemma 6.1. *Let $\pi = (C_0, C_1, ...)$ denote a run of P_{cancel} with $\sum_v C_0(v) < 0$. Then there exists $i \geq 0$ such that either all configurations $C_i, C_{i+1}, ...$ only have states in $\{-E, ..., -1\}$, or they only have states $\{-k, ..., k\}$.*

Proof. First, note that it suffices to show the claim for a single i, as $C_i(V) \subseteq \{-E, ..., -1\}$ implies $C_{i+1}(V) \subseteq \{-E, ..., -1\}$, and $C_i(V) \subseteq \{-k, ..., k\}$ even implies $C_{i+1} = C_i$.

This then follows from Lemma D.3 together with the following observation: $\langle \mathsf{cancel} \rangle$ is symmetric w.r.t. negation of all contributions, hence we could flip all signs, apply Lemma D.3, and derive the statement that there are only finitely many C_i with $C_i(V) \cap \{-E, ..., -k - 1\} \neq \emptyset$ and $C_i(V) \cap \{0, ..., E\} \neq \emptyset$.

As $0 > \sum_v C_0(v) = \sum_v C_1(v) = ...$, it is impossible that $C_i(V) \cap \{-E, ..., 0\}$ is empty, for any i. Hence Lemma D.3 yields that we eventually have $C_i(V) \cap \{k + 1, ..., E\} = \emptyset$ for all sufficiently large i. Combining this with the above observation we get the desired statement. $\qquad \square$

Lemma 6.2. *Assuming that no agent enters state $\bot$, π is accepting iff $\varphi(L_G) = 1$. Additionally, π cannot reach a configuration with all leaders in state $\bot$.*

We split the proof into two parts, Lemmata D.4 and D.5.

Lemma D.4. *Assuming that no agents enters state $\bot$, π is accepting iff $\varphi(L_G) = 1$.*

Proof. If no weak broadcast is executed in π, then the computation is necessarily accepting ($\square$ is only reachable via $\langle \mathsf{reject} \rangle$), so we can assume that $\varphi(C_\varphi) = 0$. Additionally, we know that π is a run of P'_{detect} as well, which simulates P_{detect}, so there is a run τ of P_{detect} s.t. π is a reordering of an extension of τ. As $\langle \mathsf{detect} \rangle$ does not affect the first component, we get a run $\sigma = (K_0, K_1, ...)$ of P_{cancel} by projecting τ onto the first component. Due to $\varphi(C_\varphi) = 0$, Lemma 6.1 implies that any run of P_{cancel} starting at K_0 would eventually have only states in $\{-k, ..., k\}$, or only states in $\{-E, ..., -1\}$. In both cases, executing $\langle \mathsf{detect} \rangle$ would move a leader from L to L_{double} or $L_\square$.

In run π, it is not possible for a leader to leave state L_{double} or $L_\square$, as these states are broadcast initiating. This contradicts the weak fairness condition, as then either $\langle \mathsf{double} \rangle$ or $\langle \mathsf{reject} \rangle$ must be executed eventually.

Therefore, let i denote the first step at which a weak broadcast is executed in π (i.e. $\langle \mathsf{double} \rangle$ and/or $\langle \mathsf{reject} \rangle$), and $M \subseteq V$ the set of its initiators. If there is a leader $v \notin M$, then it cannot be in state $\bot$, due to our assumption, nor can it be in $\square$, as $\langle \mathsf{reject} \rangle$ has

not been executed before step i. But then v would move to state $\bot$ in step i, which cannot happen by assumption. Hence M is precisely the set of leaders.

If both $\langle\mathsf{double}\rangle$ and $\langle\mathsf{reject}\rangle$ are executed at step i, i.e. $(\,\cdot\,, L_{\mathrm{double}}), (\,\cdot\,, L_{\square}) \in C_i(M)$, then C_{i+1} has all leaders in state L or $\square$, with at least one in each. Additionally, C_{i+1} is a valid input configuration of P_{detect} (it does not contain any intermediate states added in P'_{detect}). Any fair run τ of P_{detect} starting in C_{i+1} has one leader v which starts in state L and moves to $\bot$ upon the first execution of $\langle\mathsf{detect}\rangle$ as there is an agent in $\square$. So v enters neither L_{double} nor $L_{\square}$ in τ. Therefore, until the second broadcast is executed at step $j > i$ in π, we have $C(v) \notin Q \times \{L_{\mathrm{double}}, L_{\square}\}$ for any configuration $C \in \{C_{i+1}, ..., C_j\}$. If $j = \infty$, then v moves eventually to $\bot$ in π, as it does in τ, otherwise either $\langle\mathsf{double}\rangle$ or $\langle\mathsf{reject}\rangle$ move v immediately to $\bot$. In both cases, our assumption is violated, so at step i we cannot execute both $\langle\mathsf{double}\rangle$ and $\langle\mathsf{reject}\rangle$.

Now there are two cases. If we execute only $\langle\mathsf{reject}\rangle$ at step i of π, we know that $C_i(v) = (\,\cdot\,, L_{\square})$ for any leader v. This is only possible if $\langle\mathsf{detect}\rangle$ moves all leaders to $L_{\square}$ at once, so at some point a configuration in π had only states in $\mathsf{last}^{-1}(\{-E, ..., -1\} \times Q_L)$, which neither $\langle\mathsf{detect}\rangle$ nor a transition of P_{cancel} can change.[2] In particular, this means $C_i(V) \subseteq \mathsf{last}^{-1}(\{-E, ..., -1\} \times Q_L)$, so $\langle\mathsf{reject}\rangle$ would move all agents (including the leaders) to $\square$. At that point, no further transitions can be performed and the protocol moves into a stable 0-consensus. This is correct, as it is only possible for P_{cancel} to move all agents to states $\{-E, ..., -1\}$ if the sum of all contributions in C_0 is negative.

The second case is executing only $\langle\mathsf{double}\rangle$ at step i of π. Similarly, this is only possible if all leaders move to L_{double} at once using $\langle\mathsf{detect}\rangle$. For that to happen, all agents must be in states $\{-k, ..., k\} \times \{0, L\}$ before executing $\langle\mathsf{detect}\rangle$, moving the leaders to L_{double}. It is not possible to execute $\langle\mathsf{detect}\rangle$ or any transition of P_{detect} with only these states, so we get $C_i(V) \subseteq \{-k, ..., k\} \times \{0, L_{\mathrm{double}}\}$ as well and $\langle\mathsf{double}\rangle$ moves the agents back to states $Q \times \{0, L\}$ by doubling their contributions. Doubling every contribution does not change whether the sum is negative, so our claim follows inductively in this case, by considering the suffix $C_{i+1}, C_{i+2}, ...$. (Note that $C_0, ..., C_i$ do not contain state $\square$. So if this case happens infinitely often, which occurs only if the sum of contributions is zero, π is accepting.) $\qquad\square$

Lemma D.5. *The run π cannot reach a configuration with all leaders in state $\bot$.*

Proof. If $\langle\mathsf{reject}\rangle$ is ever executed, then a leader (its initiator) enters state $\square$, from which it cannot enter $\bot$. Otherwise, it is not possible for any agent to enter $\square$, thus $\langle\mathsf{detect}\rangle$ cannot move an agent to $\bot$. Only $\langle\mathsf{double}\rangle$ remains, but it also leaves the leader initiating the broadcast in state L. $\qquad\square$

Proposition 6.3. *For every predicate $\varphi : \mathbb{N}^l \to \{0, 1\}$ such that $\varphi(x_1, ..., x_l) \Leftrightarrow a_1 x_1 + ... + a_l x_l \geq 0$ with $a_1, ..., a_l \in \mathbb{Z}$ there is a bounded-degree `DAf`-automaton computing φ.*

---

[2]It is clear that this holds for some reordering of π. To be entirely precise we would have to argue that it is impossible to reorder the steps at which the leaders enter $L_{\square}$ to before the steps where the other agents enter $\{-E, ..., -1\} \times Q_L$.

Proof. Let $\pi = C_0 C_1 \ldots$ denote a fair run of P_{reset} starting in a configuration C_0 with $C_0(V) \subseteq Q_{\text{cancel}} \times \{0, L\}$. Note that all valid initial configurations have this form. As before, we refer to agents starting in $((\,\cdot\,, L), \cdot\,)$ as *leaders*. If no agent ever enters a state $((\bot, \cdot\,), \cdot\,)$, then $\langle\text{reset}\rangle$ is never executed and Lemma D.4 implies that we reach a correct consensus. If $\langle\text{reset}\rangle$ is executed at some step i, we move to a configuration C_{i+1} with only states $C_{i+1}(V) \subseteq Q_{\text{cancel}} \times \{0, L\}$, i.e. a valid choice for C_0. Let $\pi' := C_{i+1} C_{i+2} \ldots$ denote the suffix of π starting at $i + 1$. Due to Lemma D.5 we know that at least one leader is not in state $\bot$ when executing $\langle\text{reset}\rangle$, so π' has strictly fewer leaders than π, but at least one (the latter follows directly from the definition of $\langle\text{reset}\rangle$). Hence, we conclude that $\langle\text{reset}\rangle$ is executed only finitely often.

It still remains to show that an agent entering $((\,\cdot\,, \bot), \cdot\,)$ at some point implies that $\langle\text{reset}\rangle$ will be executed. This follows immediately, as all such states are broadcast-initiating and thus can only execute $\langle\text{reset}\rangle$. $\qquad\square$