arXiv:1110.0334v1 [cs.DC] 3 Oct 2011

A Taxonomy of Daemons in Self-Stabilization

Swan Dubois* Sébastien Tixeuil†

Abstract

We survey existing scheduling hypotheses made in the literature in self-stabilization, commonly referred to under the notion of *daemon*. We show that four main characteristics (distribution, fairness, boundedness, and enabledness) are enough to encapsulate the various differences presented in existing work. Our naming scheme makes it easy to compare daemons of particular classes, and to extend existing possibility or impossibility results to new daemons. We further examine existing daemon transformer schemes and provide the exact transformed characteristics of those transformers in our taxonomy.

Keywords: Self-stabilization, Daemon, Scheduler, Fairness, Centrality, Boundedness, Enabledness, Transformers, Distributed Algorithms, Taxonomy.

1 Introduction

Daemons are one of the most central yet less understood concepts in self-stabilization. Self-stabilization [11, 12, 33] is a versatile approach to enable forward recovery in distributed systems and networks. Intuitively a distributed system is self-stabilizing if it is able to recover proper behavior after being started from an arbitrary initial global state. This permit to withstand any kind of transient fault or attack (*i.e.* transient in the sense that faults stop occurring after a while) as the recovery mechanism does not make any assumption about what caused the initial arbitrary state.

Self-stabilizing protocols have to fight against two main adversaries that are interdependent. The first adversary is the initial global state. The second adversary is the amount of asynchrony amongst participants. In classical fault-tolerant (*e.g.* crash fault tolerant), more asynchrony usually means more impossibilities [16]. In self-stabilization, more synchrony can also be the source of more impossibilities.

Consider for example the mutual exclusion protocol proposed by Herman [25] and depicted in Figure 1. The protocol operates under the assumption that the network has the shape of a unidirectional ring (processes may only obtain information from their predecessor on the ring, and send information on their successor on the ring). Processes may hold tokens depending on their initial state, and the goal of the protocol is to ensure that regardless of the initial state, the network converges to a point where a single token is present and circulates infinitely often thereafter. Informally, the protocol can be described as follows: whenever a process holds a token, it keeps the token with probability p, and sends the token to its immediate successor on the ring with probability $1-p$. If a process holding a token receives a token from its predecessor, the two tokens are merged. This protocol was well studied assuming synchronous scheduling for all processes [15, 17] and convergence to a single token configuration is expected in $\Theta(n^2)$ time units. Now, if process scheduling can be asynchronous, the protocol may not self-stabilize, *i.e.* there may exist an initial state and a particular schedule that prevent tokens from merging. Such an example is presented in Figure 1: Consider that there exists two initial token in a ring of size five at positions A and B. The scheduling is as follows: the process at position A is scheduled for execution until it passes its token (this happens in $O(1)$ expected time), then the process at position B is scheduled for execution until it passes its token (again, this happens in $O(1)$

*UPMC Sorbonne Universités & INRIA, France, swan.dubois@lip6.fr

†UPMC Sorbonne Universités & Institut Universitaire de France, France, sebastien.tixeuil@lip6.fr

expected time). The new configuration is isomorphic to the first one, and the schedule repeats. As a result, the two tokens that are initially present never merge, and the protocol does not stabilize.

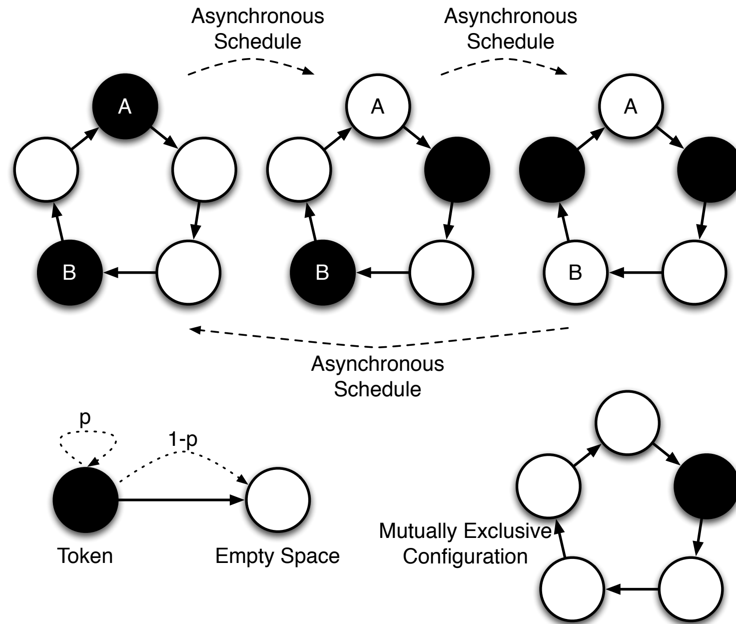

Figure 1: Mutual exclusion *vs.* asynchronous scheduling

Another example is the vertex coloring protocol of Gradinariu *et al.* [23] that is depicted in Figure 2. This protocol operates on arbitrary shaped networks under the assumption that no two neighboring processes are scheduled simultaneously. The protocol colors the graph using $deg(g) + 1$ colors in a greedy manner, where $deg(g)$ denotes the maximum degree of graph g. Whenever a process is scheduled for execution, it checks whether its color conflicts with one of its neighbors (*i.e.* it has the same color as at least one neighbor). If so, it takes the minimal (assuming arbitrary global order on colors) available color to recolor itself. When the scheduling precludes neighbors to be simultaneously activated, the protocol converges to a vertex coloring of the graph. When the scheduling is synchronous, the protocol may not stabilize. Consider the example presented in Figure 2: the initial configuration is *symmetric white*, that is, all processes have white color. If all processes are scheduled for execution in this context, they all choose the minimal available color (here, black) and the system reaches a symmetric black configuration. Again, if all processes are scheduled for execution in this context, they all choose the minimal available color (here, white) and the system reaches a symmetric white configuration. The scheduling repeats and the system never stabilizes.

Those two examples are representatives of the assumptions made to ensure stabilization of particular protocols. They also show that depending on the problem to be solved, depending on the protocol used to solve the problem, the class of scheduling hypotheses made is quite different. It is nevertheless appealing yet difficult to relate those two scheduling assumptions in a common framework (one relates to temporal constraints, while the other relates to spatial constraints). Literature presenting self-stabilizing protocols typically abstract scheduling assumptions under the notion of *daemon*. Intuitively, a daemon is just a predicate on global executions, which could in principle be any possible predicate. If every execution of a particular protocol that satisfies the daemon's predicate converges to a legitimate configuration, the protocol is self-stabilizing under this daemon.

This approach has the advantage of clearly separating the protocol (that is designed to solve a particular problem) and the scheduling assumptions (that can be seen as an adversary of the protocol, hence the term *daemon*). However, the problem of comparing possibly unrelated daemons may occur *e.g.* when choosing

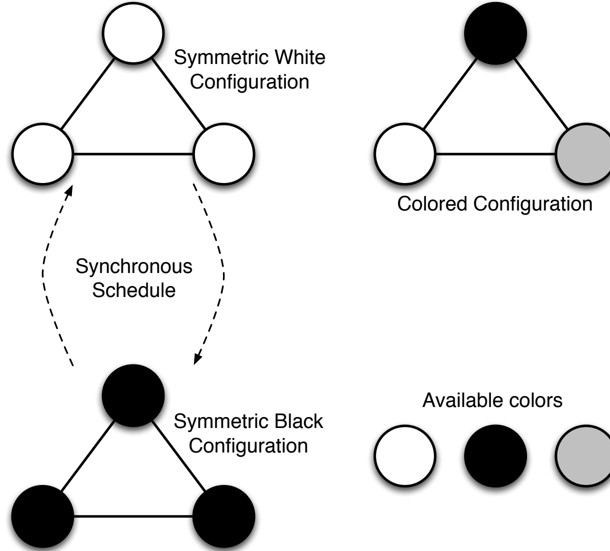

Figure 2: Vertex coloring *vs.* synchronous scheduling

a particular protocol for implementation in a particular environment (*i.e.* assuming a particular daemon). One would generally like to design a protocol for the strongest adversary (that is, the most inclusive defining predicate), while impossibility results should be given for the weakest adversary (that is, the least inclusive defining predicate). Obviously, checking whether a particular solution supports a particular environment (that is, the daemon supported by the solution includes the daemon defining the target environment) or whether a particular problem is solvable in a particular environment (that is, the daemon that makes the problem impossible to solve intersects with the daemon defining the target environment) are important questions a self-stabilizing protocol designer or implementer should be able to answer.

This paper presents a taxonomy for describing daemons having already been used in the self-stabilizing literature. After presenting our model in Section 2, we review in Section 3 the four characteristic traits of daemons existing in the literature. In Section 4, we show how our taxonomy can be used to compare daemons in particular contexts with a "more powerful" relation, and maps classical daemons according to their respective power. Section 5 reviews algorithms transformations for turning a daemon into another and depicts the influence of the transformation with respect to all four characteristic daemon traits. Section 6 provides some concluding remarks.

2 Model and Definitions

Distributed protocol A distributed system consists of a set of processes that form a communication graph. The processes are vertices in this graph and V denotes the set of vertices. The edges of this graph are pairs of processes that can communicate with each other. Such pairs are neighbors and E denotes the set of edges ($E \subseteq V^2$). Hence, $g = (V, E)$ is the communication graph of the distributed system. Each vertex of g has a set of variables, each of them ranges over a fixed domain of values. A state $\gamma(v)$ of a vertex v is the vector of values of all variables of v at a given time. An assignment of values to all variables of the graph is a configuration. The set of configurations of g is denoted by Γ. An action α of g transitions the graph from one configuration to another. The set of actions of g is denoted by A ($A = \{(\gamma, \gamma')|\gamma \in \Gamma, \gamma' \in \Gamma, \gamma \neq \gamma'\}$). A *distributed protocol* π on g is defined as a subset of A that gathers all actions of g allowed by π. The set

of distributed protocols on g is denoted by Π ($\Pi = P(A)$[1]).

Execution Given a graph g, a distributed protocol π on g, an *execution* σ of π on g starting from a given configuration γ_0 is a maximal sequence of actions of π of the following form $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)(\gamma_2, \gamma_3)\ldots$. An execution is *maximal* if it is either infinite or finite but its last configuration is terminal (that is, there exists no actions of π starting from this configuration). The set of all executions of π on g starting from all configurations of Γ is denoted by Σ_π. The set of all executions of all distributed protocols on S starting from all configurations of Γ is denoted by Σ_Π ($\Sigma_\Pi = \{\Sigma_\pi | \pi \in \Pi\}$).

Daemon The asynchrony of executions is captured by an abstraction called *daemon*. Intuitively, a daemon is a restriction on the executions of distributed protocols to be considered possible. A formal definition follows.

Definition 1 (Daemon) *Given a graph g, a daemon d on g is a function that associates to each distributed protocol π on g a subset of executions of π.*

$$\begin{array}{rcrcl} d & : & \Pi & \longrightarrow & P(\Sigma_\Pi) \\ & & \pi & \longmapsto & d(\pi) \in P(\Sigma_\pi) \end{array}$$

The set of all daemons on g is denoted by $\mathcal{D}$.

Given a graph g, a daemon d on g and a distributed protocol π on g, an execution σ of π ($\sigma \in \Sigma_\pi$) is *allowed* by d if and only if $\sigma \in d(\pi)$. Also, given a graph g, a daemon d on g and a distributed protocol π on g, we say that π *runs* on g under d if we consider that only possible executions of π on g are those allowed by d.

Other Notations Given a graph g and a distributed protocol π on g, we introduce the following set of notations. First, n denotes the number of vertices of the graph whereas m denotes the number of edges ($n = |V|$ and $m = |E|$). The distance between two vertices u and v (that is, the length of a shortest path between u and v in g) is denoted by $dist(g, u, v)$. The diameter of g (that is, the maximal distance between two vertices of g) is denoted by $diam(g)$. The maximal degree of g (that is, the maximal number of neighbors of a vertex in g) is denoted by $deg(g)$ (note that $deg^+(g)$ denotes the maximal out-degree when g is oriented).

Each action of g is characterized by the set of vertices that change their state during the action. We define the following function:

$$\begin{array}{rcrcl} Act & : & A & \longrightarrow & P(V) \\ & & \alpha = (\gamma, \gamma') & \longmapsto & \{v \in V | \gamma(v) \neq \gamma'(v)\} \end{array}$$

A vertex v is enabled by π in a configuration γ if and only if

$$\exists \gamma' \in \Gamma, (\gamma, \gamma') \in \pi, \gamma(v) \neq \gamma'(v)$$

Each configuration of g is characterized by the set of vertices enabled by π in this configuration. We define the following function:

$$\begin{array}{rcrcl} Ena & : & \Gamma \times \Pi & \longrightarrow & P(V) \\ & & (\gamma, \pi) & \longmapsto & \{v \in V | v \text{ is enabled by } \pi \text{ in } \gamma\} \end{array}$$

3 Characterization of Daemons

In this section, we review the four characteristic traits of daemons existing in the literature, namely *distribution* (Section 3.1), *fairness* (Section 3.2), *boundedness* (Section 3.3), and *enabledness* (Section 3.4).

[1] where, for any set S, $P(S)$ denotes the set of parts of S.

3.1 Distribution

Constraints about the spatial scheduling of processes appeared since the seminal paper of Dijkstra [11], as both the central (a single process is scheduled for execution at any given time) and the distributed (any subset of enabled processes may be scheduled for execution at any given time) daemons are presented. Subsequent literature [6, 29, 5] enriched the initial model with intermediate steps. Intuitively a daemon is k-central is no two processes less than k hops away are allowed to be simultaneously scheduled. A formal definition follows.

Definition 2 (k-Centrality) *Given a graph g, a) daemon d is k-central if and only if*

$$\exists k \in \mathbb{N}, \forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall i \in \mathbb{N}, \forall (u,v) \in V^2, \\ [u \neq v \wedge u \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_i, \gamma_{i+1})] \Rightarrow dist(g,u,v) > k$$

The set of k-central daemons is denoted by k-$\mathcal{C}$.

In the literature, a 0-central daemon is often called *distributed*, and a $diam(g)$-central daemon is either called *central* or *sequential*.

Proposition 1 *Given a graph g, the following statement holds:*

$$\forall k \in \{0, \ldots, diam(g) - 1\}, (k+1)\text{-}\mathcal{C} \subsetneq k\text{-}\mathcal{C}$$

Proof Let g be a graph and $k \in \{0, \ldots, diam(g) - 1\}$. We first prove that $(k+1)$-$\mathcal{C} \subseteq k$-$\mathcal{C}$.

Let d be a daemon such that $d \in (k+1)$-$\mathcal{C}$. Then, by definition:

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall i \in \mathbb{N}, \forall (u,v) \in V^2, \\ [u \neq v \wedge u \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_i, \gamma_{i+1})] \Rightarrow dist(g,u,v) > k+1$$

As $k < k+1$, we obtain that: $\forall (u,v) \in V^2, dist(g,u,v) > k+1 \Rightarrow dist(g,u,v) > k$. As a consequence:

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall i \in \mathbb{N}, \forall (u,v) \in V^2, \\ [u \neq v \wedge u \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_i, \gamma_{i+1})] \Rightarrow dist(g,u,v) > k$$

By definition, this implies that $d \in k$-$\mathcal{C}$ and shows us that $(k+1)$-$\mathcal{C} \subseteq k$-$\mathcal{C}$.

There remains to prove that $(k+1)$-$\mathcal{C} \neq k$-$\mathcal{C}$. It is sufficient to construct a daemon d such that: $d \in k$-$\mathcal{C}$ and $d \notin (k+1)$-$\mathcal{C}$.

Let d be a daemon of k-$\mathcal{C}$ that satisfies:

$$\exists \pi \in \Pi, \exists \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \exists i \in \mathbb{N}, \exists (u,v) \in V^2, \\ u \neq v \wedge u \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_i, \gamma_{i+1}) \wedge dist(g,u,v) = k+1$$

Note that d exists since the execution σ is not contradictory with the fact that $d \in k$-$\mathcal{C}$. On the other hand, we can observe that $d \notin (k+1)$-$\mathcal{C}$ since the execution σ cannot satisfy the definition of an execution allowed by a $(k+1)$-central daemon. This completes the proof of the proposition. □

Figure 3 renders Proposition 1 graphically.

3.2 Fairness

The fairness properties of daemons was not discussed in the seminal paper of Dijkstra [11], as "executing and action" was tantamount to "using critical section" in its mutual exclusion schemes. So, only global progress was assumed, *i.e.* any set of enabled processes could be scheduled for execution. This very weak assumption was later referred to as an "unfair" daemon [28, 6, 29, 7], since it may happen that a continuously enabled process is never scheduled for execution. In our taxonomy, this "unfair" property is simply having no assumptions besides "distributed". The notion of *weak fairness* [30, 26] prevent such behaviors, as it mandates continuously enabled processes to eventually be scheduled by the daemon. A formal definition follows.

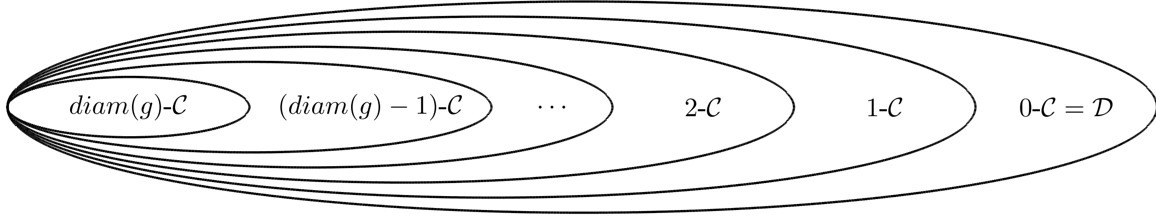

Figure 3: Inclusions of sets of daemons with respect to distribution.

Definition 3 (Weak Fairness) *Given a graph g, a daemon d is weakly fair if and only if*

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots \in \Sigma_\pi, \\ [\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, v \in Ena(\gamma_j, \pi)) \wedge (\forall j \geq i, v \notin Act(\gamma_j, \gamma_{j+1}))] \Rightarrow \sigma \notin d(\pi)$$

A weakly fair daemon is also called a fair daemon. The set of (weakly) fair daemons is denoted by $\mathcal{WF}$ or by $\mathcal{F}$. A daemon that is not fair is called unfair. The set of unfair daemons is denoted by $\bar{\mathcal{F}}$ ($\bar{\mathcal{F}} = \mathcal{D} \setminus \mathcal{F}$).

For some protocols (including protocols involving Byzantine behaviors [13, 14]), weak fairness is not sufficient to guarantee convergence, and the notion of *strong fairness* was defined [32, 30]. Intuitively a daemon is strongly fair if any process that is enabled infinitely often is eventually scheduled for execution by the daemon. A formal definition follows.

Definition 4 (Strong Fairness) *Given a graph g, a daemon d is strongly fair if and only if*

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots \in \Sigma_\pi, \\ [\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, \exists k \geq j, v \in Ena(\gamma_k, \pi)) \wedge (\forall j \geq i, v \notin Act(\gamma_j, \gamma_{j+1}))] \Rightarrow \sigma \notin d(\pi)$$

The set of strongly fair daemons is denoted by $\mathcal{SF}$.

The strongest notion of fairness (in self-stabilizing systems of finite size) is due to Gouda [20]. In short, a weakly stabilizing protocol (*i.e.* a protocol such that from any initial configuration, there exists an execution that leads to a legitimate configuration) is in fact self-stabilizing assuming Gouda's notion of fairness. Intuitively, a daemon is *Gouda fair* if from any configuration that appears infinitely often in an execution, every transition is eventually scheduled for execution. A formal definition follows.

Definition 5 (Gouda Fairness) *Given a graph g, a daemon d is Gouda fair if and only if*

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots \in \Sigma_\pi, \forall (\gamma, \gamma') \in \pi \\ [\exists i \in \mathbb{N}, (\forall j \geq i, \exists k \geq j, \gamma_k = \gamma) \wedge (\forall j \geq i, (\gamma_j, \gamma_{j+1}) \neq (\gamma, \gamma'))] \Rightarrow \sigma \notin d(\pi)$$

The set of Gouda fair daemons is denoted by $\mathcal{GF}$.

Proposition 2 *Given a graph g, the following properties hold:*

$$\begin{array}{rcl} \mathcal{GF} & \subsetneq & \mathcal{SF} \\ \mathcal{SF} & \subsetneq & \mathcal{WF} \\ \mathcal{WF} & \subsetneq & \mathcal{D} \end{array}$$

Proof We first prove that $\mathcal{GF} \subsetneq \mathcal{SF}$. We start by proving that $\mathcal{GF} \subseteq \mathcal{SF}$.

Let d be a daemon of $\mathcal{GF}$. Assume that there exist $\pi \in \Pi$ and $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi$ such that

$$\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, \exists k \geq j, v \in \mathit{Ena}(\gamma_k, \pi)) \wedge (\forall j \geq i, v \notin \mathit{Act}(\gamma_j, \gamma_{j+1}))$$

Since π is a finite subset of actions of g, this property implies the following:

$$\exists (\gamma, \gamma') \in \pi, \exists i \in \mathbb{N}, (\forall j \geq i, \exists k \geq j, \gamma_k = \gamma) \wedge (\forall j \geq i, (\gamma_j, \gamma_{j+1}) \neq (\gamma, \gamma'))$$

As $d \in \mathcal{GF}$, we can deduce that $\sigma \notin d(\pi)$ by definition. Consequently:

$$\begin{aligned}&\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi,\\ &\quad [\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, \exists k \geq j, v \in \mathit{Ena}(\gamma_k, \pi)) \wedge (\forall j \geq i, v \notin \mathit{Act}(\gamma_j, \gamma_{j+1}))] \Rightarrow \sigma \notin d(\pi)\end{aligned}$$

This proves that $d \in \mathcal{SF}$ and hence that $\mathcal{GF} \subseteq \mathcal{SF}$.

It remains to prove that $\mathcal{GF} \neq \mathcal{SF}$. It is sufficient to construct a daemon d such that $d \in \mathcal{SF}$ and $d \notin \mathcal{GF}$.

Let g be a graph and π be a distributed protocol such that:

$$\exists (\gamma, \gamma', \gamma'') \in \Gamma^3, (\gamma, \gamma') \in \pi \wedge (\gamma, \gamma'') \in \pi \wedge \mathit{Act}(\gamma, \gamma') = \mathit{Act}(\gamma, \gamma'')$$

Then, it is possible to define a daemon $d \in \mathcal{SF}$ and an execution $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi$ such that:

$$\exists i \in \mathbb{N}, (\forall j \geq i, \exists k \geq j, \gamma_k = \gamma) \wedge (\forall j \geq i, \gamma_j = \gamma \Rightarrow (\gamma_j, \gamma_{j+1}) = (\gamma, \gamma'') \neq (\gamma, \gamma'))$$

We can conclude that $d \notin \mathcal{GF}$ since the execution σ cannot satisfy the definition of an execution allowed by a Gouda fair daemon. That proves the result (since $d \in \mathcal{SF}$ by assumption).

There remains to prove that $\mathcal{SF} \subsetneq \mathcal{WF}$. We first prove that $\mathcal{SF} \subseteq \mathcal{WF}$.

Let d be a daemon of $\mathcal{SF}$. Assume that there exists $\pi \in \Pi$ and $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi$ such that

$$\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, v \in \mathit{Ena}(\gamma_j, \pi)) \wedge (\forall j \geq i, v \notin \mathit{Act}(\gamma_j, \gamma_{j+1}))$$

This property implies the following:

$$\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, \exists k = j, v \in \mathit{Ena}(\gamma_k, \pi)) \wedge (\forall j \geq i, v \notin \mathit{Act}(\gamma_j, \gamma_{j+1}))$$

As $d \in \mathcal{SF}$, we can deduce that $\sigma \notin d(\pi)$ by definition. Consequently:

$$\begin{aligned}&\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi,\\ &\quad [\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, v \in \mathit{Ena}(\gamma_j, \pi)) \wedge (\forall j \geq i, v \notin \mathit{Act}(\gamma_j, \gamma_{j+1}))] \Rightarrow \sigma \notin d(\pi)\end{aligned}$$

This proves that $d \in \mathcal{WF}$ and hence that $\mathcal{SF} \subseteq \mathcal{WF}$.

It remains to prove that $\mathcal{SF} \neq \mathcal{WF}$. It is sufficient to construct a daemon d such that $d \in \mathcal{WF}$ and $d \notin \mathcal{SF}$.

Let g be a graph, π be a distributed protocol and u, v be two vertices such that:

$$\exists (\gamma, \gamma') \in \Gamma^2, \begin{cases} v \in \mathit{Ena}(\gamma, \pi) \wedge u \in \mathit{Ena}(\gamma, \pi) \wedge v \notin \mathit{Ena}(\gamma', \pi) \wedge u \in \mathit{Ena}(\gamma', \pi) \\ v \notin \mathit{Act}(\gamma, \gamma') \wedge u \in \mathit{Act}(\gamma, \gamma') \wedge v \notin \mathit{Act}(\gamma', \gamma) \wedge u \in \mathit{Act}(\gamma', \gamma) \\ (\gamma, \gamma') \in \pi \wedge (\gamma', \gamma) \in \pi \end{cases}$$

Then, it is possible to define a daemon $d \in \mathcal{WF}$ and an execution $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi$ such that:

$$\sigma \in d(\pi) \wedge (\forall p \in \mathbb{N}, \gamma_{2p} = \gamma \wedge \gamma_{2p+1} = \gamma')$$

We can observe that σ satisfies:

$$\exists i = 0 \in \mathbb{N}, [(\forall j \geq i, (\exists k \geq j, v \in \mathit{Ena}(\gamma_k, \pi)) \wedge (\exists k' \geq j, v \notin \mathit{Ena}(\gamma_{k'}, \pi))) \wedge (\forall j \geq i, v \notin \mathit{Act}(\gamma_j, \gamma_{j+1}))]$$

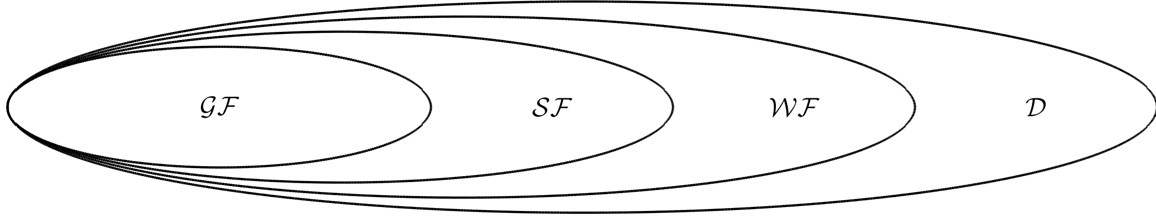

Figure 4: Inclusions of sets of daemons with respect to fairness.

We can conclude that $d \notin \mathcal{SF}$ since the execution σ cannot satisfy the definition of an execution allowed by a strongly fair daemon. That proves the result (since $d \in \mathcal{WF}$ by assumption).

Finally, we prove that $\mathcal{WF} \subsetneq \mathcal{D}$. As the definition implies that $\mathcal{WF} \subseteq \mathcal{D}$, it remains to prove that $\mathcal{WF} \neq \mathcal{D}$. It is sufficient to construct a daemon d such that $d \in \mathcal{D}$ and $d \notin \mathcal{WF}$.

Let g be a graph and π be a distributed protocol such that there exists $v \in V$ satisfying:

$$\forall(\gamma, \gamma') \in \pi, v \in Ena(\gamma, \pi) \Rightarrow |Ena(\gamma, \pi)| \geq 2$$

Then, it is possible to define a daemon d and an execution $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi$ such that:

$$\forall i \in \mathbb{N}, v \notin Act(\gamma_i, \gamma_{i+1}) \wedge \sigma \in d(\pi)$$

We can conclude that $d \notin \mathcal{WF}$ since the execution σ cannot satisfy the definition of an execution allowed by a weakly fair daemon. That proves the result (since $d \in \mathcal{D}$ by definition). $\square$

Figure 4 renders Proposition 2 graphically. Devismes *et al.* [10] observe that in infinite systems, Gouda fairness is not the strongest form of fairness.

3.3 Boundedness

Boundedness was first presented in [6] as a property achieved by a daemon transformer (see also Section 5) and was also used as a benchmark to evaluate the performance of self-stabilizing protocols under various kinds of daemons [1, 2]. Intuitively a daemon is k-bounded if no process can be scheduled more than k times between any two schedulings of any other process. Note that this does not imply that there exists a bound on the "speed" ratio between any two processes: in particular if a process is never scheduled in a particular execution, another process may be scheduled more than k times in the execution sequel without breaking the k-boundedness constraint. As a matter of fact, a daemon can be both k-bounded and unfair. A formal definition follows.

Definition 6 (k-Boundedness) *Given a graph g, a daemon d is k-bounded if and only if*

$$\begin{array}{l}\exists k \in \mathbb{N}^*, \forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V, \\ \quad \big[[v \in Act(\gamma_i, \gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\ \quad \Rightarrow \forall u \in V \setminus \{v\}, |\{l \in \mathbb{N} | l < i \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| \leq k\big] \wedge \\ \quad \big[[i < j \wedge v \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_j, \gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\ \quad \Rightarrow \forall u \in V \setminus \{v\}, |\{l \in \mathbb{N} | i \leq l < j \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| \leq k\big]\end{array}$$

The set of k-bounded daemons is denoted by k-$\mathcal{B}$. The set of bounded daemons is denoted by $\mathcal{B}$ ($\mathcal{B} = \bigcup_{k \in \mathbb{N}^*} k\text{-}\mathcal{B}$). A daemon that is not k-bounded for any $k \in \mathbb{N}^*$ is called unbounded. The set of unbounded daemons is denoted by $\bar{\mathcal{B}}$ ($\bar{\mathcal{B}} = \mathcal{D} \setminus \mathcal{B}$).

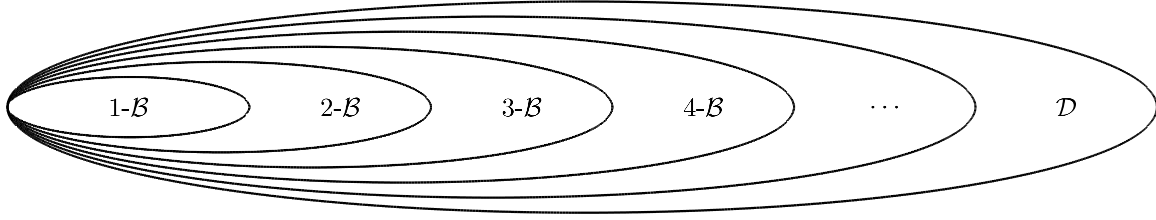

Figure 5: Inclusions of sets of daemons with respect to boundedness.

Proposition 3 *Given a graph g, the following statements hold:*

$$\forall k \in \mathbb{N}^*, \begin{cases} k\text{-}\mathcal{B} \subsetneq (k+1)\text{-}\mathcal{B} \\ k\text{-}\mathcal{B} \subsetneq \mathcal{D} \end{cases}$$

Proof Let g be a graph and $k \in \mathbb{N}^*$. We first prove that k-$\mathcal{B} \subseteq (k+1)$-$\mathcal{B}$.

Let d be a daemon such that $d \in k$-$\mathcal{B}$. Then, by definition:

$$\begin{aligned}
&\exists k \in \mathbb{N}^*, \forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V, \\
&\qquad \big[[v \in Act(\gamma_i, \gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\
&\qquad \Rightarrow \forall u \in V \setminus \{v\}, |\{l \in \mathbb{N} | l < i \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| \leq k\big] \wedge \\
&\qquad \big[[i < j \wedge v \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_j, \gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\
&\qquad \Rightarrow \forall u \in V \setminus \{v\}, |\{l \in \mathbb{N} | i \leq l < j \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| \leq k\big]
\end{aligned}$$

As $k < k+1$, we obtain that:

$$\begin{aligned}
&\exists k \in \mathbb{N}^*, \forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V, \\
&\qquad \big[[v \in Act(\gamma_i, \gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\
&\qquad \Rightarrow \forall u \in V \setminus \{v\}, |\{l \in \mathbb{N} | l < i \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| \leq k+1\big] \wedge \\
&\qquad \big[[i < j \wedge v \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_j, \gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\
&\qquad \Rightarrow \forall u \in V \setminus \{v\}, |\{l \in \mathbb{N} | i \leq l < j \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| \leq k+1\big]
\end{aligned}$$

By definition, this implies that $d \in (k+1)$-$\mathcal{B}$ and shows us that k-$\mathcal{B} \subseteq (k+1)$-$\mathcal{B}$.

There remains to prove that k-$\mathcal{B} \neq (k+1)$-$\mathcal{B}$. It is sufficient to construct a daemon d such that: $d \in (k+1)$-$\mathcal{B}$ and $d \notin k$-$\mathcal{B}$.

Let d be a daemon of $(k+1)$-$\mathcal{B}$ that satisfies:

$$\begin{aligned}
&\exists \pi \in \Pi, \exists \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \exists (i,j) \in \mathbb{N}^2, \exists v \in V, \\
&\qquad i < j \wedge v \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_j, \gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1})) \\
&\qquad \wedge (\exists u \in V \setminus \{v\}, |\{l \in \mathbb{N} | i \leq l < j \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| = k+1)
\end{aligned}$$

Note that d exists since the execution σ is not contradictory with the fact that $d \in (k+1)$-$\mathcal{B}$. On the other hand, we can observe that $d \notin k$-$\mathcal{B}$ since the execution σ cannot satisfy the definition of an execution allowed by a k-bounded daemon. This completes the proof of the first property.

Finally, we prove that k-$\mathcal{B} \subsetneq \mathcal{D}$. By definition, k-$\mathcal{B} \subseteq \mathcal{D}$. There remains to prove that k-$\mathcal{B} \neq \mathcal{D}$. By the first property, there exists a daemon d such that $d \in (k+1)$-$\mathcal{B}$ and $d \notin k$-$\mathcal{B}$. By definition, $(k+1)$-$\mathcal{B} \subseteq \mathcal{D}$ holds. Hence, the claimed result. □

Figure 5 renders Proposition 3 graphically.

3.4 Enabledness

Enabledness is a characterization of daemon properties that is introduced in this paper. It is defined to be related to the intuitive notion that the ratio between the "speed" of the fastest process and that of the slowest process is bounded. In an asynchronous setting where we use configurations and time-independent transitions between configurations, k-enabledness intuitively means that a particular process can not be enabled more than k times before being activated. A formal definition follows.

Definition 7 (k-Enabledness) *Given a graph g, a daemon d is k-enabled if and only if*

$$\begin{aligned}\exists k \in \mathbb{N}, \forall \pi \in \Pi, &\forall \sigma = (\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V,\\ &\big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | l < i \wedge v \in Ena(\gamma_l,\pi)\}| \leq k\big] \wedge\\ &\big[[i < j \wedge v \in Act(\gamma_i,\gamma_{i+1}) \wedge v \in Act(\gamma_j,\gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | i < l < j \wedge v \in Ena(\gamma_l,\pi)\}| \leq k\big] \wedge\\ &\big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l > i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | l > i \wedge v \in Ena(\gamma_l,\pi)\}| \leq k\big]\end{aligned}$$

The set of k-enabled daemons is denoted by k-$\mathcal{E}$. The set of daemons of bounded enabledness is denoted by $\mathcal{E}$ ($\mathcal{E} = \bigcup_{k\in\mathbb{N}} k\text{-}\mathcal{E}$). A daemon that is not k-enabled for any $k \in \mathbb{N}$ has an unbounded enabledness. The set of daemons of unbounded enabledness is denoted by $\bar{\mathcal{E}}$ ($\bar{\mathcal{E}} = \mathcal{D} \setminus \mathcal{E}$).

Proposition 4 *Given a graph g, the following statements hold:*

$$\forall k \in \mathbb{N}, \begin{cases} k\text{-}\mathcal{E} \subsetneq (k+1)\text{-}\mathcal{E} \\ k\text{-}\mathcal{E} \subsetneq \mathcal{D} \end{cases}$$

Proof Let g be a graph and $k \in \mathbb{N}$. We first prove that k-$\mathcal{E} \subseteq (k+1)$-$\mathcal{E}$.

Let d be a daemon such that $d \in k$-$\mathcal{E}$. Then, by definition:

$$\begin{aligned}\exists k \in \mathbb{N}, \forall \pi \in \Pi, &\forall \sigma = (\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V,\\ &\big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | l < i \wedge v \in Ena(\gamma_l,\pi)\}| \leq k\big] \wedge\\ &\big[[i < j \wedge v \in Act(\gamma_i,\gamma_{i+1}) \wedge v \in Act(\gamma_j,\gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | i < l < j \wedge v \in Ena(\gamma_l,\pi)\}| \leq k\big] \wedge\\ &\big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l > i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | l > i \wedge v \in Ena(\gamma_l,\pi)\}| \leq k\big]\end{aligned}$$

As $k < k+1$, we obtain that:

$$\begin{aligned}\exists k \in \mathbb{N}, \forall \pi \in \Pi, &\forall \sigma = (\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V,\\ &\big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | l < i \wedge v \in Ena(\gamma_l,\pi)\}| \leq k+1\big] \wedge\\ &\big[[i < j \wedge v \in Act(\gamma_i,\gamma_{i+1}) \wedge v \in Act(\gamma_j,\gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | i < l < j \wedge v \in Ena(\gamma_l,\pi)\}| \leq k+1\big] \wedge\\ &\big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l > i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))]\\ &\Rightarrow |\{l \in \mathbb{N} | l > i \wedge v \in Ena(\gamma_l,\pi)\}| \leq k+1\big]\end{aligned}$$

By definition, this implies that $d \in (k+1)$-$\mathcal{E}$ and shows us that k-$\mathcal{E} \subseteq (k+1)$-$\mathcal{E}$.

There remains to prove that k-$\mathcal{E} \neq (k+1)$-$\mathcal{E}$. It is sufficient to construct a daemon d such that: $d \in (k+1)$-$\mathcal{E}$ and $d \notin k$-$\mathcal{E}$.

Let d be a daemon of $(k+1)$-$\mathcal{E}$ that satisfies:

$$\begin{aligned}\exists \pi \in \Pi, \exists \sigma = &(\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in d(\pi), \exists (i,j) \in \mathbb{N}^2, \exists v \in V,\\ &i < j \wedge v \in Act(\gamma_i,\gamma_{i+1}) \wedge v \in Act(\gamma_j,\gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))\\ &\wedge |\{l \in \mathbb{N} | i < l < j \wedge v \in Ena(\gamma_l,\pi)\}| = k+1\end{aligned}$$

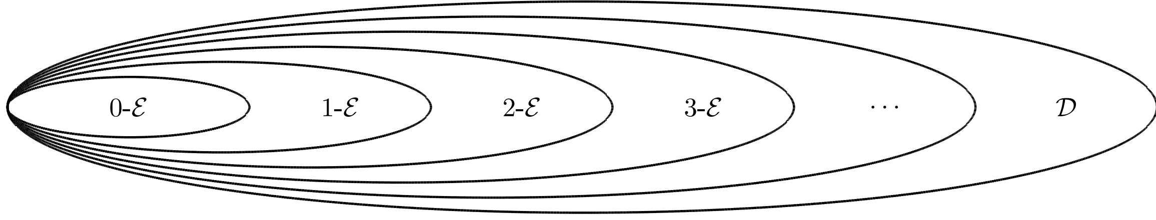

Figure 6: Inclusions of sets of daemons with respect to enabledness.

Note that d exists since the execution σ is not contradictory with the fact that $d \in (k+1)$-$\mathcal{E}$. On the other hand, we can observe that $d \notin k$-$\mathcal{E}$ since the execution σ cannot satisfy the definition of an execution allowed by a k-enabled daemon. This completes the proof of the first property.

Finally, we prove that k-$\mathcal{E} \subsetneq \mathcal{D}$. By definition, k-$\mathcal{E} \subseteq \mathcal{D}$ holds. There remains to prove that k-$\mathcal{E} \neq \mathcal{D}$. By the first property, there exists a daemon d such that $d \in (k+1)$-$\mathcal{E}$ and $d \notin k$-$\mathcal{E}$. By definition, $(k+1)$-$\mathcal{E} \subseteq \mathcal{D}$ holds. Hence the claimed result. □

Figure 6 renders Proposition 4 graphically. Unlike previous characteristic properties of daemons, enabledness is not completely independent from others. Relationship between enabledness and fairness and boundedness are depicted in the sequel (Sections 3.4.1 and 3.4.2).

3.4.1 Relationship between Fairness and Enabledness

Daemons with bounded enabledness cannot ignore scheduling processes more than k times, implying that the overall schedule is at least weakly fair. Nevertheless, the following proposition shows that the converse is not true (*i.e.* there exist daemons that are weakly fair but do not have bounded enabledness, furthermore those daemons are not strongly fair either). There also exist daemons that are strongly fair or Gouda fair, yet do not have finite enabledness.

Proposition 5 *For any given graph g, the following statements hold:*

$$\begin{array}{c}\forall d \in \mathcal{D}, d \in \mathcal{E} \Rightarrow d \in \mathcal{WF} \\ \exists d \in \mathcal{WF} \setminus (\mathcal{E} \cup \mathcal{SF}) \\ \exists d \in \mathcal{SF} \setminus (\mathcal{E} \cup \mathcal{GF}) \\ \exists d \in \mathcal{GF} \setminus \mathcal{E}\end{array}$$

Proof Let g be a graph. Let d be a daemon such that $d \in \mathcal{E}$. Then, there exists $k \in \mathbb{N}$ such that $d \in k$-$\mathcal{E}$. We are going to prove that $d \in \mathcal{WF}$.

Assume that π is a distributed protocol and $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots$ is an execution of $d(\pi)$ satisfying:

$$\exists i \in \mathbb{N}^*, \exists v \in V, v \in Act(\gamma_{i-1}, \gamma_i) \wedge (\forall j \geq i, v \in Ena(\gamma_j, \pi)) \wedge (\forall j \geq i, v \notin Act(\gamma_j, \gamma_{j+1}))$$

Then, we obtain:

$$[v \in Act(\gamma_{i-1}, \gamma_i) \wedge (\forall l \in \mathbb{N}, l > i \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \wedge |\{l \in \mathbb{N} | l > i \wedge v \in Ena(\gamma_l, \pi)\}| = \infty > k$$

This property is contradictory with $\sigma \in d(\pi)$ and $d \in k$-$\mathcal{E}$. hence, we deduct that:

$$\begin{array}{l}\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots \in \Sigma_\pi, \\ \qquad [\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, v \in Ena(\gamma_j, \pi)) \wedge (\forall j \geq i, v \notin Act(\gamma_j, \gamma_{j+1}))] \Rightarrow \sigma \notin d(\pi)\end{array}$$

That means that $d \in \mathcal{WF}$ and then, we proved that: $\forall d \in \mathcal{D}, d \in \mathcal{E} \Rightarrow d \in \mathcal{WF}$.

There remains to prove that $\exists d \in \mathcal{GF} \setminus \mathcal{E}$. Consider a daemon d such that $d \in \mathcal{GF}$ and a distributed protocol π_1 such that:

$$\exists (\gamma_0, \gamma_1, \gamma_2) \in \Gamma^3, \exists v \in V, \begin{cases} (\gamma_0, \gamma_1) \in \pi_1 \wedge v \in Ena(\gamma_0, \pi_1) \wedge v \in Act(\gamma_0, \gamma_1) \\ (\gamma_1, \gamma_2) \in \pi_1 \wedge v \in Ena(\gamma_1, \pi_1) \wedge v \notin Act(\gamma_1, \gamma_2) \\ (\gamma_2, \gamma_1) \in \pi_1 \wedge v \in Ena(\gamma_2, \pi_1) \wedge v \notin Act(\gamma_2, \gamma_1) \end{cases}$$

Let σ be an execution of $d(\pi_1)$ starting from γ_2. Now, we define the following set of executions of π_1 (where the product operator denotes the concatenation of portions of executions):

$$\forall k \in \mathbb{N}, \sigma_k = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2).\left[(\gamma_2, \gamma_1)(\gamma_1, \gamma_2)\right]^k.\sigma$$

We can define a daemon d' in the following way:

$$\begin{cases} \forall \pi \in \Pi \setminus \{\pi_1\}, d'(\pi) = d(\pi) \\ d'(\pi_1) = d(\pi_1) \cup \{\sigma_k | k \in \mathbb{N}\} \end{cases}$$

Then, we can observe that $d' \in \mathcal{GF}$ by construction and that, for any $k \in \mathbb{N}$, the execution $\sigma_k \in d'(\pi_1)$ does not satisfy the definition of k-enabledness. Consequently, we prove that: $d' \in \mathcal{GF} \setminus \mathcal{E}$. If we follow the same reasoning starting from a daemon d in $\mathcal{WF} \setminus \mathcal{SF}$ (respectively in $\mathcal{SF} \setminus \mathcal{GF}$), we prove that $d' \in \mathcal{WF} \setminus (\mathcal{E} \cup \mathcal{SF})$ (respectively that $d' \in \mathcal{SF} \setminus (\mathcal{E} \cup \mathcal{GF})$), which ends the proof. □

3.4.2 Relationship between Boundedness and Enabledness

As previously mentioned, there is not relationship between boundedness and fairness. In this section, we prove that there is a connexion between (finite) enabledness and (finite) boundedness. In particular, if a daemon is both k-enabled and k'-bounded (for some particular integers k and k'), then $k \leq (n-1) \times k'$ (where n denotes the number of processes in the system). However, there exist daemons that are k-enabled (for some integer k) but do not have finite boundedness, and daemons that are k'-bounded (for some integer k') but do not have finite enabledness.

Proposition 6 *For any given graph g, the following statements hold:*

$$\forall d \in \mathcal{D}, \forall (k, k') \in \mathbb{N} \times \mathbb{N}^*, (d \in k\text{-}\mathcal{E} \wedge d \in k'\text{-}\mathcal{B}) \Rightarrow k \leq (n-1) \times k'$$
$$\forall k \in \mathbb{N}, \exists d \in k\text{-}\mathcal{E} \setminus \mathcal{B}$$
$$\forall k \in \mathbb{N}^*, \exists d \in k\text{-}\mathcal{B} \setminus \mathcal{E}$$

Proof Firstly, we prove that $\forall d \in \mathcal{D}, \forall (k, k') \in \mathbb{N} \times \mathbb{N}^*, (d \in k\text{-}\mathcal{E} \wedge d \in k'\text{-}\mathcal{B}) \Rightarrow k \leq (n-1) \times k'$. Consider a daemon d such that $d \in k\text{-}\mathcal{E}$ and $d \in k'\text{-}\mathcal{B}$ for two given $(k, k') \in \mathbb{N} \times \mathbb{N}^*$.

As d is k'-bounded, between two consecutive actions of any vertex v, any vertex u such that $u \neq v$ takes at most k' actions. This implies that there exists at most $(n-1) \times k'$ actions between two consecutive actions of v (since the daemon must ensure the progress). This implies that, between two consecutive actions of v, there exists at most $(n-1) \times k'$ configurations where v is enabled (without being activated by construction). As d has a bounded enabledness k, we can deduce that $k \leq (n-1) \times k'$, which proves the result.

Secondly, we prove that $\forall k \in \mathbb{N}, \exists d \in k\text{-}\mathcal{E} \setminus \mathcal{B}$. Consider $k \in \mathbb{N}$, a daemon d such that $d \in k\text{-}\mathcal{E}$ and a distributed protocol π_1 such that:

$$\forall \ell \in \mathbb{N}^*, \exists (\gamma_{\ell+1}, \gamma_\ell, \dots, \gamma_1, \gamma_0) \in \Gamma^{\ell+2}, \exists v \in V, \begin{cases} \forall i \in \{0, \dots, \ell\}, (\gamma_{i+1}, \gamma_i) \in \pi_1 \\ \forall i \in \{0, \dots, \ell\}, Act(\gamma_{i+1}, \gamma_i) = Ena(\gamma_{i+1}, \pi_1) = V \end{cases}$$

Let σ be an execution of $d(\pi_1)$ starting from γ_0. Now, we define the following set of executions of π_1 (where the product operator denotes the concatenation of portions of executions):

$$\forall k' \in \mathbb{N}^*, \sigma_{k'} = (\gamma_{k'+1}, \gamma_{k'})(\gamma_{k'}, \gamma_{k'-1}) \dots (\gamma_2, \gamma_1)(\gamma_1, \gamma_0).\sigma$$

Note that, for any $k' \in \mathbb{N}^*$, the portion of execution $(\gamma_{k'+1}, \gamma_{k'})(\gamma_{k'}, \gamma_{k'-1}) \dots (\gamma_2, \gamma_1)(\gamma_1, \gamma_0)$ is 0-enabled. Hence, any execution of $\{\sigma_{k'} | k' \in \mathbb{N}^*\}$ is k-enabled.

We can define a daemon d' in the following way:

$$\begin{cases} \forall \pi \in \Pi \setminus \{\pi_1\}, d'(\pi) = d(\pi) \\ d'(\pi_1) = d(\pi_1) \cup \{\sigma_{k'} | k' \in \mathbb{N}^*\} \end{cases}$$

Then, we can observe that $d' \in k\text{-}\mathcal{E}$ by construction and that, for any $k' \in \mathbb{N}^*$, the execution $\sigma_{k'} \in d'(\pi_1)$ does not satisfy the definition of k'-boundedness. Consequently, we prove that: $d' \in k\text{-}\mathcal{E} \setminus \bigcup_{k' \in \mathbb{N}^*} k'\text{-}\mathcal{B} = k\text{-}\mathcal{E} \setminus \mathcal{B}$.

Finally, we prove that $\forall k \in \mathbb{N}^*, \exists d \in k\text{-}\mathcal{B} \setminus \mathcal{E}$. Consider $k \in \mathbb{N}^*$, a daemon d such that $d \in k\text{-}\mathcal{B}$ and a distributed protocol π_1 such that:

$$\exists (\gamma_0, \gamma_1, \gamma_2) \in \Gamma^3, \exists v \in V, \begin{cases} (\gamma_0, \gamma_1) \in \pi_1 \wedge v \in Ena(\gamma_0, \pi_1) \wedge Act(\gamma_0, \gamma_1) = \{v\} \\ (\gamma_1, \gamma_2) \in \pi_1 \wedge v \in Ena(\gamma_1, \pi_1) \wedge v \notin Act(\gamma_1, \gamma_2) \\ (\gamma_2, \gamma_1) \in \pi_1 \wedge v \in Ena(\gamma_2, \pi_1) \wedge v \notin Act(\gamma_2, \gamma_1) \\ Act(\gamma_1, \gamma_2) = Act(\gamma_2, \gamma_1) \end{cases}$$

Let σ be an execution of $d(\pi_1)$ starting from γ_2. Now, we define the following set of executions of π_1 (where the product operator denotes the concatenation of portions of executions):

$$\forall k' \in \mathbb{N}, \sigma_{k'} = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2).\big[(\gamma_2, \gamma_1)(\gamma_1, \gamma_2)\big]^{k'}.e$$

Note that, for any $k' \in \mathbb{N}$, the portion of execution $(\gamma_0, \gamma_1)(\gamma_1, \gamma_2).\big[(\gamma_2, \gamma_1)(\gamma_1, \gamma_2)\big]^{k'}$ is 1-bounded. Hence, any execution of $\{\sigma_{k'} | k' \in \mathbb{N}\}$ is k-bounded.

We can define a daemon d' in the following way:

$$\begin{cases} \forall \pi \in \Pi \setminus \{\pi_1\}, d'(\pi) = d(\pi) \\ d'(\pi_1) = d(\pi_1) \cup \{\sigma_{k'} | k' \in \mathbb{N}\} \end{cases}$$

Then, we can observe that $d' \in k\text{-}\mathcal{B}$ by construction and that, for any $k' \in \mathbb{N}$, the execution $\sigma_{k'} \in d'(\pi_1)$ does not satisfy the definition of k'-enabledness. Consequently, we prove that: $d' \in k\text{-}\mathcal{B} \setminus \bigcup_{k' \in \mathbb{N}} k'\text{-}\mathcal{E} = k\text{-}\mathcal{B} \setminus \mathcal{E}$.

□

4 Comparing Daemons

The four main characteristics presented in Section 3 provide a convenient way to define a particular class of daemons: this class simply combines the four characteristic properties. A formal definition follows.

Definition 8 (Daemon class) *Given a graph g and four sets of daemons*

$$\begin{cases} C & \in & \{k\text{-}\mathcal{C} | k \in \{0, \dots, diam(g)\}\} \\ B & \in & \{\mathcal{D}, k\text{-}\mathcal{B} | k \in \mathbb{N}^*\} \\ E & \in & \{\mathcal{D}, k\text{-}\mathcal{E} | k \in \mathbb{N}\} \\ F & \in & \{\mathcal{D}, \mathcal{WF}, \mathcal{SF}, \mathcal{GF}\} \end{cases},$$

the class *of daemons $\mathcal{D}(C, B, E, F)$ is defined by $\mathcal{D}(C, B, E, F) = C \cap B \cap E \cap F$.*

4.1 Comparing daemon classes

Now, each particular daemon instance d may belongs to several classes (those that include all possible executions under d). It is convenient to refer to the *minimal class* of d as the set of characteristics that strictly define d. A formal definition follows.

Definition 9 (Minimal class) *Given a graph g and a daemon d, the* minimal class *of d is the class of daemons $\mathcal{D}(C,B,E,F)$ such that:*

$$\begin{cases} d \in \mathcal{D}(C,B,E,F) \\ \forall \mathcal{D}(C',B',E',F') \subsetneq \mathcal{D}(C,B,E,F), d \notin \mathcal{D}(C',B',E',F') \end{cases}$$

In any particular class, the canonical daemon of this class is a representative element of that class such that for any daemon d in the class, any execution allowed by d is also allowed by the canonical daemon. Simply put, the canonical daemon of a class is the largest element of this class with respect to allowed executions. A formal definition follows.

Definition 10 (Canonical Daemon) *For a given graph g and a class of daemons $\mathcal{D}(C,B,E,F)$, the canonical daemon $d(C,B,E,F)$ of $\mathcal{D}(C,B,E,F)$ is the daemon defined by:*

$$\begin{cases} d(C,B,E,F) \in \mathcal{D}(C,B,E,F) \\ \forall d \in \mathcal{D}(C,B,E,F), \forall \pi \in \Pi, \forall \sigma \in \Sigma_\pi, \sigma \in d(\pi) \Rightarrow \sigma \in d(C,B,E,F)(\pi) \end{cases}$$

This way of viewing daemons as a set of possible executions (for a particular graph g) drives a natural "more powerful" relation definition. For a particular graph g, a daemon d is more powerful than another daemon d' if all executions allowed by d' are also allowed by d. Overall, d has more scheduling choices than d'. A formal definition follows.

Definition 11 (More powerful relation) *For a given graph g, we define the following binary relation $\preccurlyeq$ on $\mathcal{D}$:*

$$\forall (d,d') \in \mathcal{D}, d \preccurlyeq d' \Leftrightarrow (\forall \pi \in \Pi, d(\pi) \subseteq d'(\pi))$$

If two daemons d and d' satisfy $d \preccurlyeq d'$, we say that d' is more powerful than d.

As with set inclusions, this "more powerful" relation induces a partial order, which is formally presented in the sequel.

Proposition 7 *For any graph g, the binary relation $\preccurlyeq$ is a partial order on $\mathcal{D}$.*

Proof Let g be a graph. We are going to prove that the binary relation $\preccurlyeq$ is reflexive, antisymmetric and transitive. Then we show that this order is not total (*i.e.* that there exists some incomparable elements by $\preccurlyeq$ in $\mathcal{D}$).

For any daemon $d \in \mathcal{D}$, we have $\forall \pi \in \Pi, d(\pi) \subseteq d(\pi)$, which proves that $\forall d \in \mathcal{D}, d \preccurlyeq d$ (reflexivity of the binary relation $\preccurlyeq$).

Let d and d' be two daemons such that $d \preccurlyeq d'$ and $d' \preccurlyeq d$. Then, by definition:

$$\left.\begin{array}{l} \forall \pi \in \Pi, d(\pi) \subseteq d'(\pi) \\ \forall \pi \in \Pi, d'(\pi) \subseteq d(\pi) \end{array}\right\} \Rightarrow \forall \pi \in \Pi, d(\pi) = d'(\pi)$$

In other words, $d = d'$ (antisymmetry of the binary relation $\preccurlyeq$).

Let d, d' and d'' be three daemons such that $d \preccurlyeq d'$ and $d' \preccurlyeq d''$. Then, by definition:

$$\left.\begin{array}{l} \forall \pi \in \Pi, d(\pi) \subseteq d'(\pi) \\ \forall \pi \in \Pi, d'(\pi) \subseteq d''(\pi) \end{array}\right\} \Rightarrow \forall \pi \in \Pi, d(\pi) \subseteq d''(\pi)$$

In other words, $d \preccurlyeq d''$ (transitivity of the binary relation $\preccurlyeq$).

Let d be a daemon, π_1 and π_2 be two distributed protocols and σ_1 and σ_2 be two executions such that:

$$\begin{cases} \pi_1 \neq \pi_2 \\ \sigma_1 \notin d(\pi_1) \\ \sigma_2 \notin d(\pi_2) \end{cases}$$

Then, we can construct two daemons d_1 and d_2 in the following way:

$$\left\{ \begin{array}{c} \forall \pi \in \Pi \setminus \{\pi_1\}, d_1(\pi) = d(\pi) \\ d_1(\pi_1) = d(\pi_1) \cup \{\sigma_1\} \end{array} \right. \text{, and } \left\{ \begin{array}{c} \forall \pi \in \Pi \setminus \{\pi_2\}, d_2(\pi) = d(\pi) \\ d_2(\pi_2) = d(\pi_2) \cup \{\sigma_2\} \end{array} \right.$$

Then, we can deduce that $d_2(\pi_1) \subsetneq d_1(\pi_1)$ and $d_1(\pi_2) \subsetneq d_2(\pi_2)$, which proves that d_1 and d_2 are not comparable using the binary relation $\preccurlyeq$. □

Another natural intuition is that if d is more powerful than d' and d belong to a particular daemon class, then d' also belongs to this class. This is formally demonstrated in the following.

Proposition 8 *For a given graph g, for any daemons d and d' and for any class of daemons $\mathcal{D}(C, B, E, F)$, the following statements hold:*

$$\left. \begin{array}{c} d' \preccurlyeq d \\ d \in \mathcal{D}(C, B, E, F) \end{array} \right\} \Rightarrow d' \in \mathcal{D}(C, B, E, F)$$

Proof Let g be a graph, d and d' be two daemons and $\mathcal{D}(C, B, E, F)$ be a class of daemons such that: $d' \preccurlyeq d$ and $d \in \mathcal{D}(C, B, E, F)$.

Assume that $C = k\text{-}\mathcal{C}$ with $k \in \{0, \ldots, diam(g)\}$. As $d \in \mathcal{D}(C, B, E, F) = C \cap B \cap E \cap F$, $d \in k\text{-}\mathcal{C}$. By definition:

$$\begin{array}{l} \forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall i \in \mathbb{N}, \forall (u, v) \in V^2, \\ \qquad [u \neq v \wedge u \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_i, \gamma_{i+1})] \Rightarrow dist(g, u, v) > k \end{array}$$

As $d' \preccurlyeq d$, $\forall \pi \in \Pi, d'(\pi) \subseteq d(\pi)$. Then, we obtain:

$$\begin{array}{l} \forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d'(\pi) \subseteq d(\pi), \forall i \in \mathbb{N}, \forall (u, v) \in V^2, \\ \qquad [u \neq v \wedge u \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_i, \gamma_{i+1})] \Rightarrow dist(g, u, v) > k \end{array}$$

This implies that $d' \in k\text{-}\mathcal{C} = C$. We can prove in a similar way that $d' \in B$, $d' \in E$ and $d' \in F$. Consequently, we obtain that $d' \in C \cap B \cap E \cap F = \mathcal{D}(C, B, E, F)$, which proves the result. □

4.2 Preserving execution properties

Meaningful distributed protocols provide non-trivial properties when operated. A property can be defined as a predicate on computations, valued with *true* when the predicate is satisfied and *false* otherwise. A distributed protocol satisfies a property if its every executions satisfy the corresponding predicate. Conversely, a property is impossible to satisfy if no protocol is such that any of its executions satisfies the corresponding predicate. Formal definitions follow.

Definition 12 (Execution property) *For a given graph g, a property of execution p is a function that associates to each execution a Boolean value.*

$$\begin{array}{rcrcl} p & : & \Sigma_\Pi & \longrightarrow & \{true, false\} \\ & & \sigma & \longmapsto & p(\sigma) \in \{true, false\} \end{array}$$

Definition 13 (Property satisfaction) *For a given graph g, a distributed protocol π satisfies a property of execution p under a daemon d (denoted by $\pi \overset{d}{\models} p$) if and only if $\forall \sigma \in d(\pi), p(\sigma) = true$.*

Definition 14 (Property impossibility) *For a given graph g, it is impossible to satisfy a property of execution p under a daemon d (denoted by $d \not\models p$) if and only if $\forall \pi \in \Pi, \exists \sigma \in d(\pi), p(\sigma) = false$.*

The "more powerful" meaning that is associated to the $\preccurlyeq$ relation permits to intuitively understand the two following theorems. If a property is guaranteed by a protocol under a daemon d, it is also guaranteed using the same protocol under any "less powerful" daemon d' (the executions allowed by d' are a – possibly strict – subset of those allowed by d). Similarly, if a property cannot be guaranteed by any protocol under a daemon d, it is also impossible to guarantee this property under a "more powerful" daemon d' (the executions that falsifies the property in these allowed by d are also present in those allowed by d'). A formal treatment follows.

Theorem 1 *For a given graph g, let p be a property of execution satisfied by a distributed protocol π under a daemon d. Then,*

$$\forall d' \in \mathcal{D}, d' \preccurlyeq d \Rightarrow \pi \overset{d'}{\models} p$$

Proof Let g be a graph, p be a property of execution satisfied by a distributed protocol π_1 under a daemon d. By definition:

$$\forall \sigma \in d(\pi_1), p(\sigma) = true$$

Assume now that d' is a daemon such that $d' \preccurlyeq d$. By definition:

$$\forall \pi \in \Pi, d'(\pi) \subseteq d(\pi)$$

Consequently, we get:

$$\forall \sigma \in \Sigma_{\pi_1}, \sigma \in d'(\pi_1) \Rightarrow \sigma \in d(\pi_1) \Rightarrow p(\sigma) = true$$

By definition, we obtain that: $\pi_1 \overset{d'}{\models} p$, which proves the theorem. $\Box$

Theorem 2 *For a given graph g, let p be a property of execution impossible under a daemon d. Then,*

$$\forall d' \in \mathcal{D}, d \preccurlyeq d' \Rightarrow d' \not\models p$$

Proof Let g be a graph, p be a property of execution impossible under a daemon d. By definition:

$$\forall \pi \in \Pi, \exists \sigma \in d(\pi), p(\sigma) = false$$

Assume now that d' is a daemon such that $d \preccurlyeq d'$. By definition:

$$\forall \pi \in \Pi, d(\pi) \subseteq d'(\pi)$$

Consequently, we obtain:

$$\forall \pi \in \Pi, \exists \sigma \in d(\pi) \subseteq d'(\pi), p(\sigma) = false$$

By definition, we obtain that: $d' \not\models p$, which proves the theorem. $\Box$

A less obvious result shows that dealing with canonical daemons (rather than with the classes they represent) is sufficient for comparison purposes. The two derived corollaries demonstrate that using characteristic daemons is also valid for proving properties (or lack hereof) executions. This is formalized in the sequel.

Theorem 3 *For a given graph g, let $d(C, B, E, F)$ and $d(C', B', E', F')$ be two canonical daemons. Then,*

$$d(C, B, E, F) \preccurlyeq d(C', B', E', F') \Leftrightarrow \begin{cases} C \subseteq C' \\ B \subseteq B' \\ E \subseteq E' \\ F \subseteq F' \end{cases}$$

Proof We first prove the "$\Leftarrow$" part of the theorem.

Assume that there exist a graph g and two canonical daemons $d(C, B, E, F)$ and $d(C', B', E', F')$ such that:

$$\begin{cases} C \subseteq C' \\ B \subseteq B' \\ E \subseteq E' \\ F \subseteq F' \end{cases}$$

We can deduce that $C \cap B \cap E \cap F \subseteq C' \cap B' \cap E' \cap F'$. Then, by the definition of a class of daemons:

$$\mathcal{D}(C, B, E, F) \subseteq \mathcal{D}(C', B', E', F')$$

By the definition of a canonical daemon, $d(C, B, E, F) \in \mathcal{D}(C, B, E, F)$. Hence:

$$d(C, B, E, F) \in \mathcal{D}(C', B', E', F')$$

As $d(C', B', E', F')$ is the canonical daemon of the class $\mathcal{D}(C', B', E', F')$, by definition:

$$\forall \pi \in \Pi, \forall \sigma \in \Sigma_\pi, \sigma \in d(C, B, E, F)(\pi) \Rightarrow \sigma \in d(C', B', E', F')(\pi)$$

In other words,

$$\forall \pi \in \Pi, d(C, B, E, F)(\pi) \subseteq d(C', B', E', F')(\pi)$$

This means that: $d(C, B, E, F) \preccurlyeq d(C', B', E', F')$, which ends the first part of the proof.

Then, we prove the "$\Rightarrow$" part of the theorem.

Assume that there exist a graph g and two canonical daemons $d(C, B, E, F)$ and $d(C', B', E', F')$ such that: $d(C, B, E, F) \preccurlyeq d(C', B', E', F')$.

By definition of the $\preccurlyeq$ relation,

$$\forall \pi \in \Pi, d(C, B, E, F)(\pi) \subseteq d(C', B', E', F')(\pi)$$

Let d be a daemon of $\mathcal{D}(C, B, E, F)$. As $d(C, B, E, F)$ is the canonical daemon of the class of daemons $\mathcal{D}(C, B, E, F)$,

$$\begin{aligned} \forall \pi \in \Pi, \forall \sigma \in \Sigma_\pi, \sigma \in d(\pi) &\Rightarrow \sigma \in d(C, B, E, F)(\pi) \\ &\Rightarrow \sigma \in d(C', B', E', F')(\pi) \end{aligned}$$

In other words, $\forall \pi \in \Pi, d(\pi) \subseteq d(C', B', E', F')(\pi)$. By the definition of the $\preccurlyeq$ relation, this implies that:

$$\forall d \in \mathcal{D}(C, B, E, F), d \preccurlyeq d(C', B', E', F')$$

As $d(C', B', E', F')$ is the canonical daemon of the class of daemons $\mathcal{D}(C', B', E', F')$, $d(C', B', E', F') \in \mathcal{D}(C', B', E', F')$ and Proposition 8 implies

$$\forall d \in \mathcal{D}(C, B, E, F), d \in \mathcal{D}(C', B', E', F')$$

In other words, $C \cap B \cap E \cap F = \mathcal{D}(C, B, E, F) \subseteq \mathcal{D}(C', B', E', F') = C' \cap B' \cap E' \cap F'$.

Assume by contradiction that $C' \subsetneq C$. By the properties of boundedness, enabledness and fairness (see propositions of Section 3), $(C \setminus C') \cap B \cap E \cap F \neq \emptyset$. So, there exists a daemon d such that $d \in C \cap B \cap E \cap F$ and $d \notin C'$. Then, we can deduce that $d \notin C' \cap B' \cap E' \cap F'$, which contradicts $C \cap B \cap E \cap F \subseteq C' \cap B' \cap E' \cap F'$.

By the same way, we can prove that:

$$\begin{cases} C \subseteq C' \\ B \subseteq B' \\ E \subseteq E' \\ F \subseteq F' \end{cases}$$

This result ends the proof. $\square$

Corollary 1 *For a given graph g, let $d(C,B,E,F)$ and $d(C',B',E',F')$ be two canonical daemons. Then, for any property of execution p satisfied by a distributed protocol π under $d(C,B,E,F)$, the following statements hold:*

$$\left.\begin{array}{l} C' \subseteq C \\ B' \subseteq B \\ E' \subseteq E \\ F' \subseteq F \end{array}\right\} \Rightarrow \pi \overset{d(C',B',E',F')}{\models} p$$

Proof This result is a direct corollary from Theorems 1 and 3. □

Corollary 2 *For a given graph S, let $d(C,B,E,F)$ and $d(C',B',E',F')$ be two canonical daemons. Then, for any property of execution p impossible under $d(C,B,E,F)$, the following statements hold:*

$$\left.\begin{array}{l} C \subseteq C' \\ B \subseteq B' \\ E \subseteq E' \\ F \subseteq F' \end{array}\right\} \Rightarrow d(C',B',E',F') \not\models p$$

Proof This result is a direct corollary from Theorems 2 and 3. □

4.3 The Case of the Synchronous Daemon

Although we did not describe it in the previous sections, the *synchronous* daemon play a very important part in the self-stabilization literature. First introduced by Herman [25] to enable analytical tractability of probabilistic self-stabilizing protocols, it was later used in a number of works, either to demonstrate impossibility results (due to initial symmetry [23]) or to enable efficient solution to existing problems (due to the single scheduling generated [15]). A synchronous daemon simply executes every enabled process at every step. A formal definition follows.

Definition 15 (Synchronous Daemon) *Given a graph g, the synchronous daemon (denoted by sd) is defined by:*

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in sd(\pi), \forall i \in \mathbb{N}, \forall v \in V, v \in Ena(\gamma_i,\pi) \Rightarrow v \in Act((\gamma_i,\gamma_{i+1}))$$

We first show that there is a connection between enabledness and synchrony. Indeed a synchronous daemon cannot prevent an enabled process from being activated, even for a single step.

Proposition 9 *For any given graph g, $0\text{-}\mathcal{E} = \{sd\}$.*

Proof Let g be a graph and d be a daemon such that $d \in 0\text{-}\mathcal{E}$. We now prove that $d = sd$.

By definition:

$$\begin{array}{l} \exists k \in \mathbb{N}, \forall \pi \in \Pi, \forall \sigma = (\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in d(\pi), \forall (i,j) \in \mathbb{N}^2, \forall v \in V, \\ \qquad \big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l < i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))] \\ \qquad \Rightarrow |\{l \in \mathbb{N} | l < i \wedge v \in Ena(\gamma_l,\pi)\}| = 0\big] \wedge \\ \qquad \big[[i < j \wedge v \in Act(\gamma_i,\gamma_{i+1}) \wedge v \in Act(\gamma_j,\gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))] \\ \qquad \Rightarrow |\{l \in \mathbb{N} | i < l < j \wedge v \in Ena(\gamma_l,\pi)\}| = 0\big] \wedge \\ \qquad \big[[v \in Act(\gamma_i,\gamma_{i+1}) \wedge (\forall l \in \mathbb{N}, l > i \Rightarrow v \notin Act(\gamma_l,\gamma_{l+1}))] \\ \qquad \Rightarrow |\{l \in \mathbb{N} | l > i \wedge v \in Ena(\gamma_l,\pi)\}| = 0\big] \end{array}$$

In other words, no action (γ,γ') of any execution of $d(\pi)$ for any distributed protocol π can satisfy: $\exists v \in V, v \in Ena(\gamma,\pi) \wedge v \notin Act(\gamma,\gamma')$. Hence:

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0,\gamma_1)(\gamma_1,\gamma_2)\ldots \in d(\pi), \forall i \in \mathbb{N}, \forall v \in V, v \notin Ena(\gamma_i,\pi) \vee v \in Act(\gamma_i,\gamma_{i+1})$$

As $v \in Act(\gamma_i, \gamma_{i+1})$ implies that $v \in Ena(\gamma_i, \pi)$, this property is equivalent to the following:

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \forall i \in \mathbb{N}, \forall v \in V, v \in Ena(\gamma_i, \pi) \Rightarrow v \in Act(\gamma_i, \gamma_{i+1})$$

By the definition of the synchronous daemon, this means that $d = sd$, which ends the proof. □

It may first come to a surprise that boundedness is absolutely not related to synchrony, but as we pointed out previously, boundedness is also not related to fairness. The exact characteristics of the synchronous daemon are captured by the following proposition.

Proposition 10 *Given a graph g, $\mathcal{D}(0\text{-}\mathcal{C}, \mathcal{D}, 0\text{-}\mathcal{E}, \mathcal{SF})$ is the minimal class of SD. Moreover, $SD = d(0\text{-}\mathcal{C}, \mathcal{D}, 0\text{-}\mathcal{E}, \mathcal{SF})$.*

Proof First, we prove that $sd \in 0\text{-}\mathcal{C} \setminus 1\text{-}\mathcal{C}$. By definition, $sd \in 0\text{-}\mathcal{C} = \mathcal{D}$. By contradiction, assume that $sd \in 1\text{-}\mathcal{C}$. Let $\pi \in \Pi$ be a distributed protocol such that:

$$\exists (\gamma, \gamma') \in \pi, Ena(\gamma, \pi) = V$$

Then, by definition of the synchronous daemon, the first action of any execution $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in sd(\pi)$ starting from $\gamma_0 = \gamma$ satisfies: $Act(\gamma_0, \gamma_1) = V$. Consequently, σ does not satisfy the property of executions allowed by a 1-central daemon, which contradicts $sd \in 1\text{-}\mathcal{C}$ and proves the result.

Second, we prove that $sd \in \bar{\mathcal{B}}$. As $sd \in \mathcal{D}$, assume for the purpose of contradiction that there exists $k \in \mathbb{N}^*$ such that $sd \in k\text{-}\mathcal{B}$. Then, consider a distributed protocol π such that:

$$\exists (v, u) \in V^2, \exists (\gamma_0, \ldots, \gamma_{k+3}) \in \Gamma^{k+4}, \begin{cases} (\gamma_0, \gamma_1) \in \pi \wedge Ena(\gamma_0, \pi) = \{v\} \\ \forall i \in \{1, \ldots, k+1\}, (\gamma_i, \gamma_{i+1}) \in \pi \wedge Ena(\gamma_i, \pi) = \{u\} \\ (\gamma_{k+2}, \gamma_{k+3}) \in \pi \wedge Ena(\gamma_{k+2}, \pi) = \{v\} \\ Ena(\gamma_{k+3}, \pi) = \emptyset \end{cases}$$

We can observe that the execution σ defined by $\sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots (\gamma_{k+2}, \gamma_{k+3})$ satisfies $\sigma \in sd(\pi)$. But, on the other hand, the following holds:

$$\begin{aligned} &\exists \pi \in \Pi, \exists \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in d(\pi), \exists (i = 0, j = k+2) \in \mathbb{N}^2, \exists v \in V, \\ &\quad [i < j \wedge v \in Act(\gamma_i, \gamma_{i+1}) \wedge v \in Act(\gamma_j, \gamma_{j+1}) \wedge (\forall l \in \mathbb{N}, i < l < j \Rightarrow v \notin Act(\gamma_l, \gamma_{l+1}))] \\ &\quad \wedge \exists u \in V \setminus \{v\}, |\{l \in \mathbb{N} | i \leq l < j \wedge u \in Act(\gamma_l, \gamma_{l+1})\}| = k+1 \end{aligned}$$

By the definition of a k-bounded daemon, this implies that $sd \notin k\text{-}\mathcal{B}$.

We now prove that $sd \in 0\text{-}\mathcal{E}$. By Proposition 8, $0\text{-}\mathcal{E} = \{sd\}$. This implies that $sd \in 0\text{-}\mathcal{E}$.

Next, we prove that $sd \in \mathcal{SF} \setminus \mathcal{GF}$. We start by proving that $sd \in \mathcal{SF}$. By the definition of the synchronous daemon:

$$\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi, \forall v \in V, (\exists i \in \mathbb{N}, v \in Ena(\gamma_i, \pi)) \Rightarrow v \in Act(\gamma_j, \gamma_{j+1})$$

Consequently,

$$\begin{aligned} &\forall \pi \in \Pi, \forall \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2) \ldots \in \Sigma_\pi, \\ &\quad [\exists i \in \mathbb{N}, \exists v \in V, (\forall j \geq i, \exists k \geq j, v \in Ena(\gamma_k, \pi)) \wedge (\forall j \geq i, v \notin Act(\gamma_j, \gamma_{j+1}))] \Rightarrow \sigma \notin sd(\pi) \end{aligned}$$

By the definition of a strongly fair daemon, this implies that $sd \in \mathcal{SF}$. Now, we prove that $sd \notin \mathcal{GF}$. Consider a distributed protocol π such that:

$$\exists (\gamma, \gamma', \gamma'') \in \Gamma^3, \begin{cases} (\gamma, \gamma') \in \pi \wedge Act(\gamma, \gamma') \subsetneq Ena(\gamma, \pi) \\ (\gamma, \gamma'') \in \pi \wedge Act(\gamma, \gamma'') = Ena(\gamma, \pi) \\ (\gamma'', \gamma) \in \pi \wedge Act(\gamma'', \gamma) = Ena(\gamma'', \pi) \end{cases}$$

We can construct an execution σ of π starting from γ in the following way: $\sigma = (\gamma, \gamma'')(\gamma'', \gamma)(\gamma, \gamma'')\ldots$. We can observe that $\sigma \in sd(\pi)$ (since at each action, any enabled vertex is activated). Consequently,

$$\exists \pi \in \Pi, \exists \sigma = (\gamma_0, \gamma_1)(\gamma_1, \gamma_2)\ldots \in sd(\pi), \exists (\gamma, \gamma') \in \pi,$$
$$\exists i = 0 \in \mathbb{N}, (\forall j \geq i, \exists k = 2j \geq j, \gamma_k = \gamma) \wedge (\forall j \geq i, (\gamma_j, \gamma_{j+1}) \neq (\gamma, \gamma'))$$

By the definition of a Gouda fair daemon, this implies that $sd \notin \mathcal{GF}$.

The four previous results imply that $\mathcal{D}(0\text{-}\mathcal{C}, \mathcal{D}, 0\text{-}\mathcal{E}, \mathcal{SF})$ is the minimal class of sd. As $\mathcal{D}(0\text{-}\mathcal{C}, \mathcal{D}, 0\text{-}\mathcal{E}, \mathcal{SF}) \subseteq 0\text{-}\mathcal{E}$ by definition and $0\text{-}\mathcal{E} = \{sd\}$ by Proposition 9, we can deduce that $\mathcal{D}(0\text{-}\mathcal{C}, \mathcal{D}, 0\text{-}\mathcal{E}, \mathcal{SF}) = \{sd\}$. Then, the definition of a canonical daemon implies that $sd = d(0\text{-}\mathcal{C}, \mathcal{D}, 0\text{-}\mathcal{E}, \mathcal{SF})$, which completes the proof. □

4.4 A map of classical daemons

We are now ready to present our map for "classical" daemons (*i.e.* daemons most frequently used in the literature. Using our taxonomy, these daemons can be defined as follows.

Definition 16 (Classical daemons) *Given a graph g, the classical daemons of the literature are defined as follows:*

- *The unfair daemon (denoted by ufd) is $d(\mathcal{D}, \mathcal{D}, \mathcal{D}, \mathcal{D})$.*
- *The weakly fair daemon (denoted by wfd) is $d(\mathcal{D}, \mathcal{D}, \mathcal{D}, \mathcal{WF})$.*
- *The strongly fair daemon (denoted by sfd) is $d(\mathcal{D}, \mathcal{D}, \mathcal{D}, \mathcal{SF})$.*
- *The Gouda fair daemon (denoted by gfd) is $d(\mathcal{D}, \mathcal{D}, \mathcal{D}, \mathcal{GF})$.*
- *The locally central unfair daemon (denoted by $1\text{-}ufd$) is $d(1\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{D})$.*
- *The locally central weakly fair daemon (denoted by $1\text{-}wfd$) is $d(1\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{WF})$.*
- *The locally central strongly fair daemon (denoted by $1\text{-}sfd$) is $d(1\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{SF})$.*
- *The locally central Gouda fair daemon (denoted by $1\text{-}gfd$) is $d(1\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{GF})$.*
- *The central unfair daemon (denoted by $0\text{-}ufd$) is $d(0\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{D})$.*
- *The central weakly fair daemon (denoted by $0\text{-}wfd$) is $d(0\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{WF})$.*
- *The central strongly fair daemon (denoted by $0\text{-}sfd$) is $d(0\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{SF})$.*
- *The central Gouda fair daemon (denoted by $0\text{-}gfd$) is $d(0\text{-}\mathcal{C}, \mathcal{D}, \mathcal{D}, \mathcal{GF})$.*

Now, our main theorem (Theorem 3) permits to map the relationships between all classical daemons in the literature is a rather compact format. For any given graph g, Figure 7 depicts graphically those relationships.

5 Daemon Transformers

As it is easier to write distributed protocols under daemons providing strong properties (that is, under weak daemons that allow only a limited set of possible executions, such as a central or a bounded daemon), many authors provide protocols to simulate the operation of a weak daemon under a strong one. Such protocols are called *daemon transformers*. Note that several works in the area of self-stabilization may be used as daemon transformers although they were not initially designed with this goal in mind (*e.g.* a self-stabilizing token circulation protocol that performs under the unfair distributed daemon can easily be turned into a daemon transformer that provides a central daemon out of an unfair distributed one).

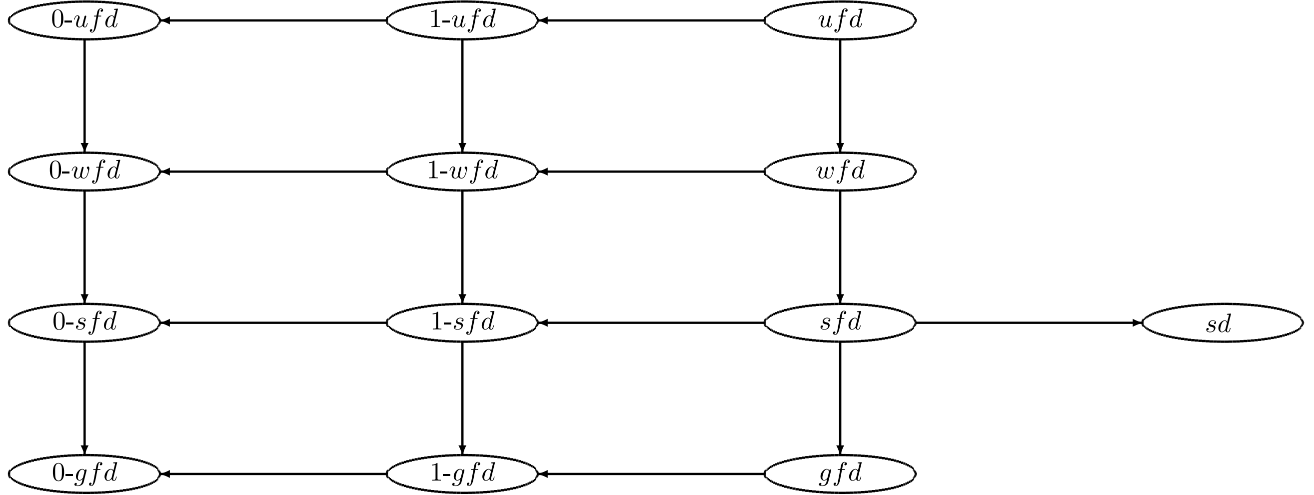

Figure 7: Relationship between classical daemons (an arrow from a daemon d to a daemon d' means that $d' \preccurlyeq d$, note that we remove all arrows obtained by transitivity).

In the following, we propose a survey of the main daemon transformers that also preserve the property of self-stabilization. That is, the protocol transforming the daemon is a self-stabilizing one. Figure 8 summarizes this survey and maps for each daemon transformer the initial daemon and the simulated one. We restrict ourselves to deterministic daemon transformers in order to be able to exactly compute the characteristics of the simulated daemon. Note that features of the emulated daemon (centrality, fairness, boundedness, and enabledness) provided in the following are satisfied only after the stabilization of the daemon transformer. In the sequel, we use the notation $d \longmapsto d'$ to denote that a daemon transformer simulates d' while operating under d.

Alternator-based daemon transformers. In 1997, Gouda and Haddix [21] introduced the alternator problem. Roughly speaking, the aim is to design a protocol such that no neighbors are enabled simultaneously yet ensures that some fairness property holds (namely, between any two steps of a particular process, any of its neighbors may execute at most one step). They claim that this protocol is useful to simulate a locally central daemon under a distributed one. Actually, this protocol ensures the following daemon transformation: $ufd \longmapsto d(1\text{-}\mathcal{C}, \mathcal{WF}, n^2\text{-}\mathcal{B}, n^2\text{-}\mathcal{E})$ and works on chain topologies only. Johnen *et al.* [27] later designed an alternator for any oriented tree but require the initial daemon to be weakly fair. In other words, they provide the following daemon transformer: $wfd \longmapsto d(1\text{-}\mathcal{C}, \mathcal{WF}, n^2\text{-}\mathcal{B}, n^2\text{-}\mathcal{E})$. Finally, Gouda and Haddix [22] provided an alternator for an arbitrary underlying communication graph that provides the following daemon transformation: $wfd \longmapsto d(1\text{-}\mathcal{C}, \mathcal{WF}, diam(g)\text{-}\mathcal{B}, (n \times diam(g))\text{-}\mathcal{E})$. This last transformer makes the following assumption: the graph is identified (that is, every vertex has a unique identifier) and each vertex knows the cyclic distance of the graph (the cyclic distance is defined as the number of edges of the longest simple cycle if the graph has cycles, and two otherwise).

Mutual exclusion-based daemon transformers. The classical mutual exclusion problem requires that no two vertices are simultaneously in critical section and that every vertex infinitely often enters critical section. So, any self-stabilizing mutual exclusion protocol may be turned into a daemon transformer that provides a central weakly fair daemon. In his seminal work on self-stabilization [11], Dijkstra proposed a self-stabilizing mutual exclusion protocol for ring topologies (using a token circulation) under a distributed unfair daemon. His protocol needs however a distinguished vertex (that is, one vertex executes a protocol that is different from every other). Formally, we can derive the following daemon transformation from this

protocol: $ufd \longmapsto d(diam(g)\text{-}\mathcal{C}, \mathcal{WF}, 1\text{-}\mathcal{B}, n\text{-}\mathcal{E})$. From this first protocol, several works later revisited the mutual exclusion problem. From a daemon transformation viewpoint, the most interesting ones follow. Using a token circulation, Beauquier *et al.* ([3]) provide a $ufd \longmapsto d(diam(g)\text{-}\mathcal{C}, \mathcal{WF}, deg^+(g)\text{-}\mathcal{B}, n \times deg^+(g)\text{-}\mathcal{E})$ daemon transformation on oriented graph whenever the graph is strongly connected. Still on graphs with a distinguished vertex, Datta *et al.* provided [8] a self-stabilizing depth-first token circulation that perform the following daemon transformation: $wfd \longmapsto d(diam(g)\text{-}\mathcal{C}, \mathcal{WF}, deg(g)\text{-}\mathcal{B}, 2m\text{-}\mathcal{E})$. Finally, Datta *et al.* [7] improved this result enabling the same daemon transformation but starting from an unfair daemon (more formally, they achieve the following daemon transformation: $ufd \longmapsto d(diam(g)\text{-}\mathcal{C}, \mathcal{WF}, deg(g)\text{-}\mathcal{B}, 2m\text{-}\mathcal{E})$) and they do not require the existence of a distinguished vertex.

Local mutual exclusion-based daemon transformers. Local mutual exclusion refines mutual exclusion since it requires the same exclusion and liveness properties but only within a vicinity around each vertex (and not for the whole graph as for the – global – mutual exclusion problem). In other words, a k-local mutual exclusion protocol ensures that no two vertices are simultaneously in critical section if their distance is less than k and that any vertex enters infinitely often in critical section. Hence, we can easily design a daemon transformer providing a weakly fair k-central daemon from such a protocol. Note that the aforementioned alternator protocols solve a particular instance of 1-local mutual exclusion.

A classical solution to 1-local mutual exclusion has been proposed by Beauquier *et al.* [1] using unbounded memory at each vertex. This protocol ensures the following daemon transformation: $ufd \longmapsto d(1\text{-}\mathcal{C}, \mathcal{WF}, (n-1)\text{-}\mathcal{B}, \frac{n(n-1)}{2}\text{-}\mathcal{E})$. Using only a bounded memory, Gairing *et al.* provided[18] a 2-local mutual exclusion that can be turned into a $ufd \longmapsto d(2\text{-}\mathcal{C}, \mathcal{WF}, m \times n^2\text{-}\mathcal{B}, m \times n^2\text{-}\mathcal{E})$ daemon transformer.

Several works give more general solutions dealing with k-local mutual exclusion for any integer k. For example, Goddart *et al.* generalize [19] the work of Gairing *et al.* [18]. Their solution performs the $ufd \longmapsto d(k\text{-}\mathcal{C}, \mathcal{WF}, O(n^2)\text{-}\mathcal{B}, O(n^2)\text{-}\mathcal{E})$ daemon transformation. Using a local clock synchronization, Boulinier and Petit provide [4] a wavelets protocol that can be used for k-local mutual exclusion. Hence, their protocol gives the following daemon transformation: $ufd \longmapsto d(k\text{-}\mathcal{C}, \mathcal{WF}, \left\lceil \frac{diam(g)}{k} \right\rceil\text{-}\mathcal{B}, \left\lceil \frac{n-1}{k} \right\rceil\text{-}\mathcal{E})$. Danturi *et al.* [5] deal with dining philosophers with generic conflicts under a distributed weakly fair daemon. The main idea is to clearly distinguish the communication graph from the conflict graph. If we consider that two vertices are in conflict if they are at distance less than k from each other, this protocol ensures k-local mutual exclusion. This protocol provides the $wfd \longmapsto d(k\text{-}\mathcal{C}, \mathcal{WF}, \mathcal{D}, deg(g)^k\text{-}\mathcal{E})$ daemon transformation but requires each vertex to be the root of a tree spanning its k-neighborhood.

Finally, Potop-Butucaru and Tixeuil introduced in [24] a weaker version of 1-local mutual exclusion by replacing the fairness property by a progress property. This new problem was called a conflict manager and leads to the $ufd \longmapsto 1\text{-}ufd$ daemon transformation. To our knowledge, this daemon transformer is the only one to perform a transformation according to a single identifier daemon characteristic.

Note that all solutions presented in this paragraph require the graph to be identified.

Other daemon transformers. Even if they transform several characteristics of daemons (with the notable exception of [24]), all previously mentioned daemon transformers are designed for transforming only the *distribution* of daemons. Indeed, only a few works dealt with transforming other daemon characteristics.

Regarding fairness transformation, Karaata [30] provided a daemon transformer to perform strong fairness under weak fairness. More formally, this protocol is a $1\text{-}wfd \longmapsto d(2\text{-}\mathcal{C}, \mathcal{SF}, n \times deg(g)^2\text{-}\mathcal{B}, deg(g)^2\text{-}\mathcal{E})$ daemon transformer. This protocol needs the graph to be identified and each vertex to have an unbounded memory. Karaata later refined [31] the protocol to perform exactly the same daemon transformation but requiring only an identified graph.

Using cross-over composition, Beauquier *et al.*[2] gave a generic transformer for enabledness. More precisely, they design a $ufd \longmapsto d(\mathcal{D}, \mathcal{WF}, k\text{-}\mathcal{B}, n \times k\text{-}\mathcal{E})$ daemon transformer whenever a transformer that provides k-boundedness is available.

6 Conclusion

We surveyed existing scheduling hypotheses made in the literature in self-stabilization, commonly referred to under the notion of *daemon*. We showed that four main characteristics (distribution, fairness, boundedness, and enabledness) are enough to encapsulate the various differences presented in existing work. Our naming scheme makes it easy to compare daemons of particular classes, and to extend existing possibility or impossibility results to new daemons. We further examined existing daemon transformer schemes and provided the exact transformed characteristics of those transformers in our taxonomy.

Two obvious extensions of this work are to include system hypotheses that are not related to scheduling (*e.g.* atomicity) and to further refine the taxonomy to include recently introduced randomized scheduling [10].

References

[1] Joffroy Beauquier, Ajoy Kumar Datta, Maria Gradinariu, and Frédéric Magniette. Self-stabilizing local mutual exclusion and daemon refinement. *Chicago J. Theor. Comput. Sci.*, 2002, 2002.

[2] Joffroy Beauquier, Maria Gradinariu, and Colette Johnen. Cross-over composition - enforcement of fairness under unfair adversary. In Datta and Herman [9], pages 19–34.

[3] Joffroy Beauquier, Maria Gradinariu, Colette Johnen, and Jérôme Olivier Durand-Lose. Token-based self-stabilizing uniform algorithms. *J. Parallel Distrib. Comput.*, 62(5):899–921, 2002.

[4] Christian Boulinier and Franck Petit. Self-stabilizing wavelets and rho-hops coordination. In *22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS08)*, pages 1–8, 2008.

[5] Praveen Danturi, Mikhail Nesterenko, and Sébastien Tixeuil. Self-stabilizing philosophers with generic conflicts. *ACM Transactions of Adaptive and Autonomous Systems (TAAS)*, 4(1), January 2009.

[6] Ajoy K Datta, Maria Gradinariu, and Sébastien Tixeuil. Self-stabilizing mutual exclusion using unfair distributed scheduler. In *IEEE International Parallel and Distributed Processing SYmposium (IPDPS'2000)*, pages 465–470, Cancun, Mexico, May 2000. IEEE Press.

[7] Ajoy K. Datta, Maria Gradinariu, and Sébastien Tixeuil. Self-stabilizing mutual exclusion with arbitrary scheduler. *The Computer Journal*, 47(3):289–298, October 2004.

[8] Ajoy K. Datta, Colette Johnen, Franck Petit, and Vincent Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13:207–218, 2000. 10.1007/PL00008919.

[9] Ajoy Kumar Datta and Ted Herman, editors. *Self-Stabilizing Systems, 5th International Workshop, WSS 2001, Lisbon, Portugal, October 1-2, 2001, Proceedings*, volume 2194 of *Lecture Notes in Computer Science*. Springer, 2001.

[10] Stéphane Devismes, Sébastien Tixeuil, and Masafumi Yamashita. Weak vs. self vs. probabilistic stabilization. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS 2008)*, Beijin, China, June 2008.

[11] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[12] Shlomi. Dolev. *Self-stabilization.* MIT Press, March 2000.

[13] Swan Dubois, Maria Potop-Butucaru, Mikhail Nesterenko, and Sébastien Tixeuil. Self-stabilizing byzantine asynchronous unison. In *Proceedings of OPODIS 2010*, Lecture Notes in Computer Science, Tozeur, Tunisia, December 2010. Springer Berlin / Heidelberg.

[14] Swan Dubois, Maria Potop-Butucaru, and Sébastien Tixeuil. Dynamic ftss in asynchronous systems: the case of unison. *Theoretical Computer Science (TCS)*, 412(29):3418–3439, July 2011.

[15] Philippe Duchon, Nicolas Hanusse, and Sébastien Tixeuil. Optimal randomized self-stabilizing mutual exclusion in synchronous rings. In *Proceedings of the 18th Symposium on Distributed Computing (DISC 2004)*, number 3274 in Lecture Notes in Computer Science, pages 216–229, Amsterdam, The Nederlands, October 2004. Springer Verlag.

[16] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[17] Laurent Fribourg, Stéphane Messika, and Claudine Picaronny. Coupling and self-stabilization. *Distributed Computing*, 18(3):221–232, 2006.

[18] Martin Gairing, Wayne Goddard, Stephen T. Hedetniemi, Petter Kristiansen, and Alice A. McRae. Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14(3-4):387–398, 2004.

[19] Wayne Goddard, Stephen T. Hedetniemi, David Pokrass Jacobs, and Vilmar Trevisan. Distance- k knowledge in self-stabilizing algorithms. *Theor. Comput. Sci.*, 399(1-2):118–127, 2008.

[20] Mohamed G. Gouda. The theory of weak stabilization. In Datta and Herman [9], pages 114–123.

[21] Mohamed G. Gouda and F. Furman Haddix. The linear alternator. In Sukumar Ghosh and Ted Herman, editors, *WSS*, pages 31–47. Carleton University Press, 1997.

[22] Mohamed G. Gouda and F. Furman Haddix. The alternator. *Distributed Computing*, 20(1):21–28, 2007.

[23] Maria Gradinariu and Sébastien Tixeuil. Self-stabilizing vertex coloring of arbitrary graphs. In *International Conference on Principles of Distributed Systems (OPODIS'2000)*, pages 55–70, Paris, France, December 2000.

[24] Maria Gradinariu and Sébastien Tixeuil. Conflict managers for self-stabilization without fairness assumption. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2007)*, page 46. IEEE, June 2007.

[25] Ted Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

[26] Tetz C. Huang, Ji-Cherng Lin, Chih-Yuan Chen, and Cheng-Pin Wang. The worst-case stabilization time of a self-stabilizing algorithm under the weakly fair daemon model. *IJALR*, 1(3):45–52, 2010.

[27] Colette Johnen, Luc Alima, Ajoy K. Datta, and Sébastien Tixeuil. Optimal snap-stabilizing neighborhood synchronizer in tree networks. *Parallel Processing Letters (PPL)*, 12(3-4):327–340, 2002.

[28] H. Kakugawa and M. Yamashita. Uniform and self-stabilizing token rings allowing unfair daemon. *ieeetpds*, 8(2):154–162, 1997.

[29] Hirotsugu Kakugawa and Masafumi Yamashita. Uniform and self-stabilizing fair mutual exclusion on unidirectional rings under unfair distributed daemon. *Journal of Parallel and Distributed Computing*, 62(5):885–898, May 2002.

[30] Mehmet Hakan Karaata. Self-stabilizing strong fairness under weak fairness. *IEEE Trans. Parallel Distrib. Syst.*, 12(4):337–345, 2001.

[31] Mehmet Hakan Karaata. An optimal self-stabilizing strarvation-free alternator. *J. Comput. Syst. Sci.*, 71(4):480–494, 2005.

[32] Mehmet Hakan Karaata and Pranay Chaudhuri. A self-stabilizing algorithm for strong fairness. *Computing*, 60(3):217–228, 1998.

[33] Sébastien Tixeuil. *Algorithms and Theory of Computation Handbook, Second Edition*, chapter Self-stabilizing Algorithms, pages 26.1–26.45. Chapman & Hall/CRC Applied Algorithms and Data Structures. CRC Press, Taylor & Francis Group, November 2009.

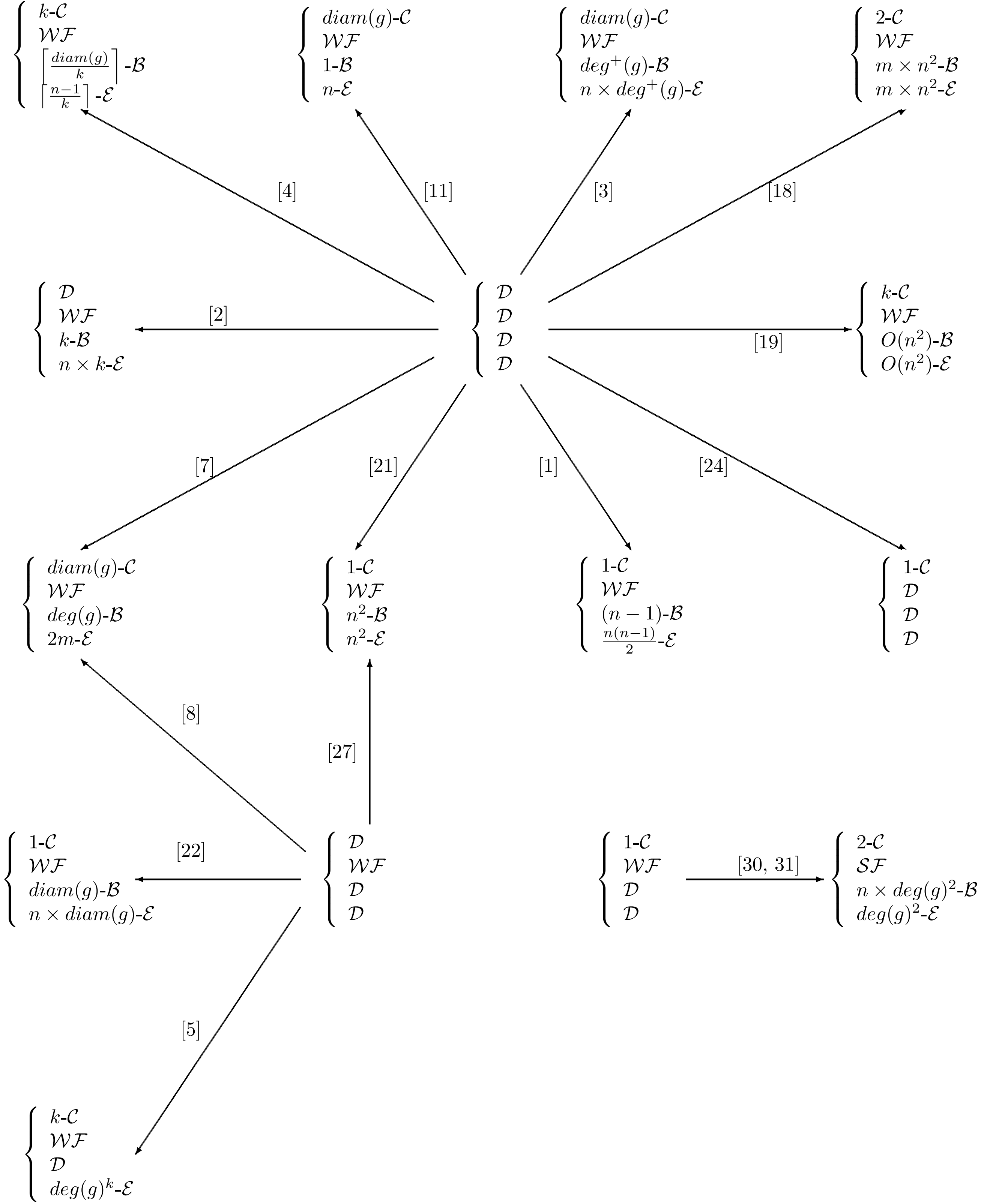

Figure 8: Summary of existing daemon transformers.