Optimal Memory Requirement for Self-Stabilizing Token Circulation

Lélia Blin, Gabriel Le Bouder, Franck Petit

Optimal Memory Requirement for Self-Stabilizing Token Circulation*

Lélia Blin[§], Gabriel Le Bouder[†], and Franck Petit[‡]
[§]IRIF, CNRS, Université Paris Cité, Paris, France
[†]LMF, CNRS, Université Paris-Saclay, Gif-sur-Yvette, France
[‡]LIP6 CNRS, Sorbonne Université, Paris, France

Abstract

In this paper, we consider networks where every transmitted message is received by all of the transmitter's neighbors. Typical such networks are wireless networks, in which to dedicate a message to a specific neighbor, the sender must specify who the recipient is by specifying the recipient's ID. Adding an identifier has a non-negligible cost, more precisely $O(\log n)$ bits in an n-node graph. Token Circulation (TC) is a fundamental problem that consists in guaranteeing that a single token circulates from one node to another, the token fairly visiting every node infinitely often. TC inherently requires that the token holder selects a unique neighboring node to which to pass the token. This paper proposes a solution that overcomes the communication of identifiers. It achieves optimal space complexity for the token circulation problem.

The contribution of this paper is fourfold. First, we present the first deterministic depth-first token circulation algorithm for rooted wireless networks that uses only $O(\log \log n)$ bits of memory per node. This is an exponential improvement compared to the classical addressing. Our algorithm assumes a Destination-Oriented Directed Acyclic Graph (DODAG) spanning the network. Less popular than spanning trees, DODAGs are nonetheless more general and adaptable spanning structures, since they do not need to distinguish one neighboring node as unique parent. Second, our algorithm has the very desirable property of being self-stabilizing. This means that starting from a configuration where zero or more than one token circulate in the network, the system is guaranteed to eventually work correctly, *i.e.*, a single token eventually fairly visits every node infinitely often, following a depth-first order. Third, it works under the unfair scheduler, that is the most challenging scheduling assumption of the model. Finally, we show that our algorithm is optimal in terms of space complexity. Meaning that, for every n-node network, no TC algorithm can use less than $\Omega(\log \log n)$ bits of memory per node, even given a rooted spanning tree and even without considering self-stabilization.

Keywords: Distributed algorithms, transient faults, self-stabilization, token circulation, memory optimization.

*This work has been partially supported by the ANR projects SKYDATA (ANR-22-CE25-0008) and DREAMY (ANR-21-CE48-0003).

1 Introduction

In recent decades, wireless network technology has gained tremendous importance. It not only more and more replaces so far "wired" network installations, but also gives rise to new applications and with them, new tech and soft stacks enabling specific communications. Many distributed algorithms written for wired networks assume that every node v of the system is equipped with a local set of *labels*, sometimes also referred to as *port labels*. This set maps the set of v's neighbors, that is the set of nodes directly linked to v by bidirectional communication links. By using their respective port sets, two neighboring nodes u and v can directly communicate by exchanging messages with each other without interfering with other nodes, and without any further specific information such as IDs. Furthermore, port labels allow to maintain global distributed spanning data structures such as various types of trees, rings, and so on, which makes solving many problems much easier.

By contrast, in wireless networks, nodes have *a priori* no knowledge of their neighborhood. In particular, without further information (particularly, IDs), nodes are unable to distinguish from which node a message is received, neither to pick out a particular node to whom send a message. Thus, in wireless networks, to differentiate the recipient or the sender of a particular message, classically, the message includes the identifier of the recipient. This identifier will be stored (temporarily or permanently) in the receiver's memory. In n-node networks, the inclusion of the identifier requires $\Omega(\log n)$ bits. The dark side effect of the ID's inclusion is that, depending on the task considered, the message size is only determined by the size of the identifiers. Several problems can be specified by a constant number of bits, like the leader election or the token circulation, or by the number of bits depending of the degree of the network, like the coloration of the nodes. In this type of problem, the space complexity is entirely consumed by the inclusion of the identifier in the messages. Moreover, this is not appropriate to many settings, including sensor networks or swarms of robots, in which the memory of each process must be kept as small as possible, ideally constant.

Token Circulation, also referred to as *token passing*, is a fundamental problem to guarantee that a single token circulates from one node to another, and the token fairly visits every node infinitely often. The most popular way to implement such a mechanism in an arbitrary topology deterministically is the *Depth-First Token Circulation* (DFTC, for short). Briefly, this well-known mechanism takes place by executing successive traversals, each of them initiated by a specific node, called the root. In each traversal, the token is passed among the nodes following a depth-first order. A traversal ends when the root holds the token again after every node has been visited by the token at least once. The overall DFTC mechanism is implemented by restarting traversals one after another by the root. By construction, the DFTC mechanism is fair, since the unvisited nodes become visited only once during each traversal cycle.

We are interested in *self-stabilizing* [12] token circulation in distributed systems. A self-stabilizing algorithm must return the system to a correct behavior in finite time, whatever its initial configuration. In the context of token circulation, typical examples of initial arbitrary system configurations are scenarios in which either no token or several tokens exist in the network. To satisfy the requirement of self-stabilization, a distributed token circulation algorithm must return in finite time the system in a configuration which contains exactly one token. Actually, it must do more than that: the algorithm must also guarantee that, eventually, this single token fairly circulates among the nodes. Following the seminal work of Dijkstra [12], token circulation has been widely investigated deterministically in the context of self-stabilization, on rings [8, 16], on (spanning) tree networks [7, 15, 20], and on arbitrary shaped networks [17, 13, 18, 11, 9, 20, 10], to quote only a few.

All the above algorithms were written in the well known *(atomic-)state* model [12], in the

sequel denoted $\mathcal{S}$, a high-level model where communication between nodes is abstracted by the ability for a node to atomically read the content of its neighbor registers, and update its own registers. More precisely, all of them were written assuming the ability for each node to locally select at least one of its neighbors: the one to which to pass the token. Such ability is commonly made with variables called *pointers* that maps $N(v)$, the set of neighbors of v. Given two neighbors u and v, by reading p_v, one of v's pointers, u is able to know whether p_v maps the edge vu, *i.e.*, the link leading to itself or not. Pointers are typically implemented in two ways: (i) the nodes use their identifiers, *i.e.*, in the example, $p_v = u$'s ID, (ii) the model assumes *port labels* that maps $N(v)$, the set of neighbors of v. So, if pointers are implemented with node IDs, then they require $\Omega(\log n)$ bits of memory. If pointers are implemented with port labels, then they need $\Omega(\log \Delta)$ bits (Δ refers to the maximum degree of the network), but nodes must be able to recognize which of its neighbor port labels links it back to itself. Note that the use of pointers is particularly helpful for the problem of Token Circulation, where sending the token to exactly one neighbor is crucial to maintain the unicity of the token.

In this paper, we assume a model where no node is able to locally select one of its neighbors, except by its identifier. In $\mathcal{S}$, this assumption makes the design of algorithms much more challenging, which often require the use of node identifiers or the construction of an underlying vertex coloring. Token circulation algorithms for $\mathcal{S}$ are proposed in [2, 1, 4]. All these algorithms work over tree topologies and use $\Theta(\log n)$ bits of memory per node.

Aside from focusing on the algorithmic aspects of the self-stabilizing DFTC problem in $\mathcal{S}$, we also aim to reduce the memory required to solve this problem. Note that memory optimization for the self-stabilizing token circulation has been widely addressed in the literature. As mentioned earlier, all of them assume pointers implemented over local ports and achieving $O(\log \Delta)$ bits of memory per node, namely [12, 8, 16, 7, 15] on rings or chains, [20] on trees, and [11] on arbitrary networks. Except [8] which assumes a uniform prime size ring, all the above algorithms require a root, *i.e.*, a node with a particular local algorithm with respect to the other nodes.

In $\mathcal{S}$, it has been shown that many standard "one-shot" tasks like leader election and spanning tree construction can be constructed in a self-stabilizing manner with memory $O(\log \log n + \log \Delta)$ bits in any n-node network with maximum degree Δ [5, 6]. This indicates that some tasks can be achieved in $\mathcal{S}$ with only sublogarithmic memory. As suggested by the results established in [3], it is much harder to reach $o(\log \log n)$ memory. No previous work addresses the self-stabilizing DFTC in $\mathcal{S}$ without pointers.

1.1 Contributions

We present the first deterministic self-stabilizing depth-first token circulation algorithm in $\mathcal{S}$ that uses only $O(\log \log n)$ bits of memory per node. This is an exponential improvement compared to the classical addressing. Our algorithm does not use any pointer variable. However, like many works in the fields of self-stabilization and networks, our algorithm assumes a specific underlying spanning distributed structure called *Destination-Oriented Directed Acyclic Graph*, DODAG for short [19]. DODAGs are bi-directional spanning data structure in which each edge is given an orientation, so that the resulting graph contains no loop, and has exactly one root: a node with only incoming edges from its neighbors. Less popular than other spanning data structures like virtual rings or spanning trees, DODAGs are nonetheless more general and adaptable than trees, since they do not need to distinguish one neighboring node as unique parent. Furthermore, they are particularly desirable in wireless networks because they allow multiple communication paths from every node to one single sink, thus limiting congestion and supporting link failures, as opposed to rooted spanning trees. As a matter of fact, DODAGs

are the basic structures used in practice by RPL (Routing Protocol for Low power and lossy networks [22]), which is the standard routing protocol for IPv6-based multi-hop wireless sensor networks [21]. By contrast with rooted spanning trees, DODAGs allow the nodes to have more than one single parent, which drastically increases the difficulty to implement the DFTC.

Another quality of our algorithm is that is works under the *unfair* scheduler, the most powerful distributed scheduler. Roughly speaking, a scheduler is considered as an adversary which tries to prevent the protocol to behave as expected. The more powerful the scheduler is, the more it can prevent the algorithm to work correctly. The nuisance power of the scheduler is modeled by the notion of *fairness*. The scheduler is considered to be *fair* if it cannot prevent forever a node to execute an action (if it has one to execute). By contrast, the unfair scheduler can prevent forever a node to execute an action, unless if it is the only node able to execute an action.

Finally, we show that our algorithm is *optimal* in terms of space complexity. Meaning that, for every n-node network, no token passing algorithm can use less than $\Omega(\log \log n)$ bits of memory per node, even given a rooted spanning tree and even without considering self-stabilization.

1.2 Outline of the paper.

In Section 2, we describe the distributed systems and the model in which our self-stabilizing DFTC algorithm is written. Section 3 describes our algorithm. Due to the lack of space, the details of the correctness of our algorithm and the proof of space optimality are postponed in the appendix. However, both are sketched in Section 4. Finally, we make concluding remarks in Section 5.

2 Model and definitions

2.1 Algorithm syntax and semantics

A distributed system is formalized by a n-nodes graph. When two nodes u and v are able to communicate with each other, this bi-directional communication is represented by an edge between u and v. The nodes u and v are said to be *neighbors*. A distributed system is *semi-uniform* if only one node is designated to have a particular role, in this paper we call this node *root*. In a distributed system, all the nodes have the same code, in semi-uniform system the root can have a different code. Let us formalise the semi-uniform distributed system, by an n-node graph $G = (V, E, r)$, where V is the set of nodes, E is the set of edges, and r is the root. We consider an *id-based* system, where every node has a distinct identifier. Note that, like in a classical distributed system, the identifiers are not consecutive between 1 and n, but are taken in $[1, n^c]$ where $c > 1$ is a constant. A system is said anonymous if nodes do not have identifiers.

Space-complexity in self-stabilization considers only *mutable memory*, the memory whose content changes during the execution of the algorithm. Mutable memory includes the space allocated for algorithm variables. On the other hand, the content of the *non-mutable memory* does *not* change during the execution. It typically stores the code, the constants (including the port label set, if any), and the node identifier. Therefore, IDs size is not a part of the space-complexity.

Each node contains variables and rules. Variable ranges over a domain of values. The variable var_v denotes the variable var located at node v. A rule is of the form [12]:

$$\langle label \rangle : \langle guard \rangle \longrightarrow \langle command \rangle$$

A *guard* is a boolean predicate over node variables. A *command* is a set of variable-assignments. A command of a node u can only update its own variables. On the other hand, u can read the variables of its neighbors. This classical communication model is called the *state model*.

An assignment of values to all variables in the system is called a *configuration*. Let Γ be the set of system configurations. A rule whose guard is **true** on one node u in some system configuration $\gamma \in \Gamma$ is said to be *enabled* on u in γ. Similarly, u is said to be enabled in γ. The atomic execution of a subset of enabled rules (at most one rule per node) results in a transition of the system from one configuration to another. This transition is called a *step*. A *run* of a distributed system is a maximal alternating sequence of configurations and steps. Maximality means that the execution is either infinite or its final configuration has no rule enabled.

2.2 Schedulers

The asynchronism of the system is modeled by an adversary (a.k.a. *scheduler*) which chooses, at each step, the subset of enabled nodes that are allowed to execute one of their rules during this step. Those schedulers can be classified according to their characteristics (like fairness, distribution, ...), and a taxonomy was presented in [14]. In this paper, we assume an *unfair asynchronous scheduler*. *Unfairness* means that even if a node u is continuously enabled, then u may never be chosen by the scheduler, unless u is the only enabled node. *Asynchronous* implies that during a computation step, if one or more nodes are enabled, then the scheduler chooses at least one (possibly more) of these enabled nodes to execute an action. This scheduler is the most challenging since no assumption is made on the subset of enabled nodes chosen by the scheduler at each step.

2.3 Self-Stabilization

A predicate is a boolean function over the set of configuration Γ. A configuration $\gamma \in \Gamma$ *satisfies* some predicate R, if R evaluates to **true** for γ. Otherwise, γ *violates* R. We define a special predicate true as follows: for any $\gamma \in \Gamma$, γ satisfies true. We use the terms *closure* and *attractor* in the definition of stabilization. Given a predicate R, provided that the system starts from a configuration in Γ satisfying R, R is said to be *closed* for a certain run r, if every configuration of that run satisfies R. Given two predicates R_1 and R_2 over Γ, R_2 is an *attractor* for R_1 if every run that starts from a configuration that satisfies R_1 contains a configuration that satisfies R_2.

The specification SP_P of a problem P is a predicate over runs of the system, which describes a specific behavior of the system. A distributed algorithm $\mathcal{A}$ solves a problem P under a certain scheduler if every execution of $\mathcal{A}$ under that scheduler satisfies the specification of P.

Definition 1 (*Self-stabilization*)

A distributed algorithm A is self-stabilizing for a specification SP_P if there exists a predicate R for P such that:

- *(Convergence) R is an attractor for* true,
- *(Closure) Any run of A starting from a configuration satisfying R satisfies R.*

2.4 DODAGs

We assume that each edge $\{u, v\} \in E$ is a bidirectional communication link provided with an orientation, from u to v, or from v to u, so that the resulting directed graph forms a

Destination-Oriented Directed Acyclic Graph [19] (DODAG), *i.e.*, a Directed Acyclic Graph (DAG) with a unique root (see Fig. 1). Such orientation assumption is similar the one that assumes a consistent global orientation of a (virtual) ring—each node locally knows how to distinguish between its right and its left neighbor—, or of a spanning tree—each node knows locally how to distinguish between neighbors that do and do not belong to the spanning tree, and among those that do belong to the tree, between its descendants and a single parent.

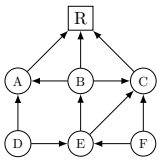

Figure 1: A DODAG.

Consider an oriented edge from node u to node v of the DODAG, we said v is the parent of u and u is the child of v. Thus each node v has two sets of neighbors, the set of parents and the set of children. Note that unlike in tree topologies a node can have several parents—refer to Appendix A, Subsection A.1 for the formal definitions.

2.5 Bit-by-Bit Communication of Identifier

In $\mathcal{S}$, nodes cannot communicate information to one single neighbor without using the fact that identifiers are, at least locally, unique (this is formally proven in Section C.4). In other words, nodes must communicate their identifier stored in the immutable memory to their neighbors to address information to a unique neighbor. The size of an identifier is $\Theta(\log n)$ bits, while we aim at designing a sub-logarithmic algorithm. To reach a $\Theta(\log \log n)$ memory, [5] make the nodes communicate their identifier by smaller pieces, more precisely bit by bit. They define a function called $\texttt{Bit}_v(i)$, which returns the position of the i^{th} most significant bit equal to 1 in $\texttt{id}_v$. In this paper, to achieve $\Theta(\log \log n)$ bits of memory per node, we use a similar $\texttt{Bit}_v(i)$ function as [5] (Appendix A.2).

3 Algorithm

Our token circulation algorithm is based on a perpetual Depth-First traversal from the root r to the other nodes of the network. The classical DFS algorithm can be summarized as follows: first r has the token, then, the token is given to one unvisited neighbor v of r, which becomes r's child and r becomes v's parent. Node v then sends the token to one of its unvisited neighbors, and the token keeps being given from node to node until it reaches a node w such that w has no unvisited neighbors. At this point, node w sends the token back to its parent, which resumes the process of selecting unvisited neighbors. This traversal continues until all nodes in the network have been visited and the token is returned to the root node r. We focus on DODAGs, so a node v with the token selects the next node in the DODAG traversal among its children, rather than among all of its neighbors. From a local point of view, the tasks for a node v are as follows: receive the token from a node p, check whether it has any unvisited children, and if so, pass the token to one of them; if not, send the token back to its parent p.

So the self-stabilizing implementation of the DFS token circulation relies on several key concepts, including the presence or absence of the token, the identification of unvisited and visited neighbors, the selection of an unvisited neighbors, and the recording of the neighbor from which the token was received (i.e., the parent).

In order to prevent token loss or duplication, the circulation of unique token cannot occur in a single step and requires acknowledgment messages. A node v must inform node u that it is sending the token, and once u has confirmed receipt, v must delete the token. If v sends the token but does not verify that u has received it, the token may be lost. If u receives the token but does not verify that v has deleted the token, the token may be duplicated.

In this work, we consider the problem of token circulation in networks with sub-logarithmic memory available in each node. Storing identifiers in variables is not feasible, as a result, selecting the next unvisited child to receive the token becomes a non-trivial task that cannot be achieved in a single step. To address this, we propose a negotiation step where the node with the token interacts with its unvisited children. During this step, each child announces its identifier piece by piece, and the child with the largest identifier wins the negotiation. We also account for unvisited children who did not win the negotiation by proposing a mechanism to ensure that they will be visited later. Self-stabilizing algorithms must be able to detect and correct inconsistencies locally caused by transient faults. One fundamental idea in the design of our algorithm is to minimize the frequency of node repairs. Firstly, the algorithm is already highly complex due to the numerous tasks that nodes must execute to achieve fair token circulation. Introducing additional rules to repair inconsistent situations would complicate both the algorithm and its proof. Secondly, in most cases, it is unnecessary to repair nodes as the token will eventually circulate in the network if the algorithm is resilient enough. The presence of the token and the rules for fair executions will allow the network to stabilize. However, there are some exceptions, i.e., repairing rules that are necessary for convergence.

To summarize our approach, let us introduce an informal algorithm (see Figure 2) that describes the steps of the algorithm for a node v.

Algorithm 1: *Local_DFS*(v)

```
if The token is present and some inconsistency is detected then
    Release the token and consider that the token is in one of your descendants
else
    if Received token from the parent then
        - Switch as a visited node;
        - Wait for parent to release the token;
    if Received token from a visited child then
        Wait for child to release the token;
    if v has the token and nobody in its neighborhood has a token then
        if ∃ unvisited children then
            while ∃ more than one unvisited child which are candidates to receive the token do
                - Ask all the candidates unvisited children for a piece of their IDs;
                - Publish the strongest piece;
                - Wait for all the candidates which do not have the strongest piece to declare
                  themselves losers;
            - Send the token to the winner;
            - Wait for the winner to receive the token;
            - Ask the losing unvisited children to be ready for the next negotiation;
            - Wait for the update of the unvisited children;
            - Release the token;
        else
            - Send the token to the parent;
            - Wait for the parent to receive the token;
            - Release the token;
```

Figure 2: Local algorithm for a node v of DFS Token circulation

3.1 Variables

Before presenting the rules of our algorithm, let us first introduce the four variables it uses.

Visited and Unvisited Children. Nodes have to remember which of their children have already been visited in order to either send the token to unvisited children or to send it back to a parent. Classically, we implement this with a boolean variable *color*:

$$\mathtt{c}_v \in \{\mathtt{red}, \mathtt{green}\}$$

Visited nodes are nodes that have the same color as the root. Nodes change from the state "unvisited" to "visited" (during a traversal initiated by the root) by changing their color upon receipt of the token, detailed bellow. The choice of the next child to be visited is based on the identifier of the unvisited children of the token holder: the chosen child is the one with highest identifier. As a consequence, the circulation of the token always follows the same traversal path, depending on the relative order of the identifiers in the DODAG. Let us consider a node v with the token, and choose one of its unvisited neighbors u and sends the token to it, and so on to a node w surrounded by visited nodes. The node w sends the token back to its parent w_p which sends it to one of its unvisited children. The process is repeated until all nodes are visited, and the token returns to the node r. By construction the DFS token circulation is fair, since the unvisited nodes become visited exactly once. When the root r has no unvisited children, r switches its color to the other value (from red to green for example). Immediately, all the visited nodes become unvisited, for they now have the opposite color of r. This guarantees that this fair token circulation is perpetual. Only one operation, denoted by $\mathtt{c}_v := \neg\mathtt{c}_v$, is used in this variable, which is switching its value from `red` to `green` and vice versa.

State of the Token. To achieve atomicity of the operations required to maintain the network in a consistent state, we need to define a variable which takes eight different values.

$$\mathtt{tok}_v \in \{\bot, \bigstar, \bullet, \downdownarrows, \circlearrowleft, \downarrow, \circ, \uparrow\}$$

Let us give some explanations on those different values. The state $\underline{\mathtt{tok}_v = \bot}$ corresponds to nodes that are **away** from the current token circulation. For example, when the root has the token, all the other nodes are such that $\mathtt{tok}_v = \bot$. Reciprocally $\bot$ is a value that cannot be taken by the root, which would mean that no token is circulating in the network.

When a node that is not involved in the current token circulation receives the token from its parent, it simultaneously switches it color, and switches its token value to $\underline{\mathtt{tok}_v = \bigstar}$. This value corresponds to the moment when a node **receives the token from its parent**, which is supposed to happen exactly once in each circulation round.

When the parent of v has finally left the token (recall that such operations are not atomic), v can **start negotiating** with its children to decide to which one it will send the token. To do that, it sets $\underline{\mathtt{tok}_v = \bullet}$, and keep that state until one winner is elected.

Once a winner u is elected *i.e.* when all nodes of a different color than v have declared themselves losers, u excepted, v updates $\underline{\mathtt{tok}_v = \downdownarrows}$ to **offer the token to one of its children**. After that, the child u that won the negotiation can update its own variable $\mathtt{tok}_u$ to $\bigstar$.

After its child u took the token, v updates its variable to $\underline{\mathtt{tok}_v = \circlearrowleft}$, to let its children which lost the negotiation reset their variable in order to be **ready for the next negotiation**, when v will receive back the token from u.

When all the children of v have finished resetting their variable, v sets $\underline{\mathtt{tok}_v = \downarrow}$, which indicates that it is a node involved in a token circulation, but that **the token it had is below it**, to one of its descendants. After u has finished the token circulation below it, it sets $\mathtt{tok}_u = \uparrow$, to send the token back to v. If nothing wrong happened, v has only one child with a value $\mathtt{tok} \neq \bot$ at that moment, and therefore can take the token. To do so, it updates $\underline{\mathtt{tok}_v = \circ}$, and then **waits for u to effectively drop the token**, by setting $\mathtt{tok}_u = \bot$.

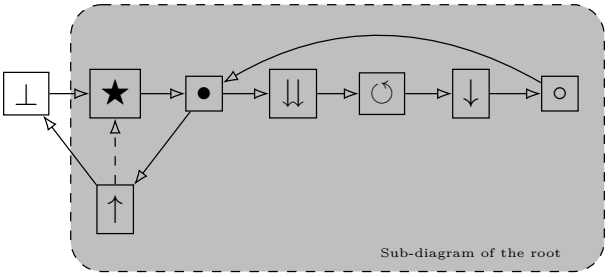

Figure 3: Transition diagram for variable $\texttt{tok}_v$.

After that, v restarts a negotiation, setting $\texttt{tok}_v = \bullet$. After some time, v has no more children of a different color. When this happens, v **offers the token to its parent** w by setting $\underline{\texttt{tok}_v = \uparrow}$. Before effectively dropping the token, v waits that w acknowledges the token by updating its variable to $\texttt{tok}_w = \circ$, but it also waits for all of its children to update their variable linked to the negotiation.

Indeed, in the current situation, all the children of v have successively won the token. As explained above, those nodes are prevented from taking one token of another color, to avoid livelocks. But now, the circulation of the token of v is terminated, so this becomes irrelevant. Worse, if nodes do not reset their variable to a value that allows them to negotiate, then they will all be ignored when a token from another color will come the next round. When all the children of v have reset their variable, v finally drops the token, and sets $\texttt{tok}_v = \perp$.

Remark that for a node v with $\texttt{tok}_v \notin \{\perp, \downarrow\}$, there is a strong pressure on the possible values of $\texttt{tok}_u$, where u us a child of v, and most combinations are actually not supposed to occur in an execution. To summarize the Figure 3 represents the order between the different states taken by the variable $\texttt{tok}$.

Negotiation Variables: Bit-by-Bit Communication. To determine which of its children will receive the token, the parent progressively eliminates them, until only one remains. When it has exactly one child running for the token, it can finally declare that it gives the token. All the children that were eliminated ignore that message, and the one winner can take the token from its parent, and therefore no duplication of the token can occur.

The elimination process relies on the identifiers of the children, and on one additional variable which allows unvisited nodes to declare themselves as negotiating, or as losers of the negotiation. Step by step, the parent asks for the value of the i-th bit of the identifier of its children, and when all have answered, it reveals the biggest value it sees. All the nodes that have not announced this value consider themselves as eliminated until their parent allows them to negotiate again. This moment, when the parent allows its children who lost the election to negotiate again happens immediately after the winner of the election takes the token.

To negotiate for the token, nodes send bit-by-bit the value of their identifier to their parent. Two variables are required for that. The first one, $\texttt{ph}$, is used to communicate the position of the bit that is currently asked. It takes values between 1 and N, where N is the largest length of an identifier in the network, and thus $N \in O(\log n)$. The second variable is $\texttt{b}$, and is used to communicate the value of the bit that each child has, and for the parent to communicate the highest value it has seen. The variable $\texttt{b}_v$ is used to communicate the values given by function $\texttt{Bit}_v$. It takes values in $\{0, 1, \perp\}$, where $\perp$ is the value that corresponds, for children, to the absence of a bit at that position (the identifier is too short), and, for the parent, to the request of the value of the bit. For simplicity, we denote the tuple $(\texttt{ph}_v, \texttt{b}_v)$ by the notation $\texttt{id}_v$.

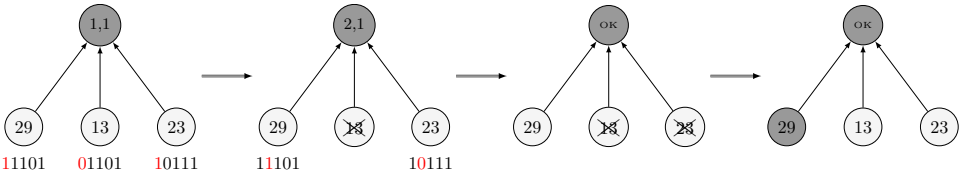

Figure 4: Process of the designation of one child to give the token to. The parent reveals the biggest bit announced, then the children with small identifiers quit the negotiation, and in the end the winner takes the token and the losers are ready to negotiate again.

The general scheme is the following: first the parent sets $\texttt{id}_v = (1, \perp)$. Then all the children answer with the value of their first bit, by setting $\texttt{id}_u = (1, \texttt{Bit}_u(1))$. After all the children have answered, v sets $\texttt{id}_v = (1, \texttt{b})$ where $\texttt{b}$ is 1 if v saw a 1 in its children, and 0 otherwise. Then, all the children that do not have answered $(1, \texttt{b})$ lose the negotiation. After all losers have left the negotiation, v sets $\texttt{id}_v = (2, \perp)$, and so on until only one node remains.

Negotiation Variables: State in the Negotiation Process. In addition to the identifier variables `ph` and `b`, we need a variable to remember where each node is in the negotiation process. This variable is denoted by `play` and takes four values:

$$\texttt{play}_v \in \{\texttt{P}, \texttt{L}, \texttt{W}, \texttt{F}\}$$

Nodes v such that $\texttt{play}_v = \texttt{P}$ are the nodes that are available for a negotiation process with any of their parent that has a different color. They are the only nodes which can **participate** to a negotiation, the others have either already won, or lost. Note that `P` also stands for players.

When a node v **loses** the negotiation, it sets $\texttt{play}_v = \texttt{L}$, and thus becomes a loser, and is not involved in the current negotiation process anymore.

When a node v **wins** the negotiation and receives the token, it updates its color, its variable $\texttt{tok}_v$, and it also updates $\texttt{play}_v$ to `W`, to signify that it won a negotiation. After that, and until its parent sends the token back to its own parent, v won't negotiate with any of its parents. This guarantees that whatever the initial configuration of the system, v will not create oscillations between two tokens, and therefore no livelock.

However, the color is not sufficient to prevent livelocks or deadlocks caused by multiple circulating tokens. Indeed, even assuming only two tokens, one can design pathological configurations, and especially if the two tokens are of different colors. What we must absolutely prevent is situations in which one node alternatively takes a red token from one parent, sends it back later, then switches color and takes the green token from its other parent, sends it back, and then takes the red one again, and so on. With two such children, we can design scenarios where the tokens are in a livelock, and the circulation is blocked.

Note that nodes can neither simply ignore the token, for it would create a deadlock: both token would be blocked. To avoid such situations, we force a node that has received the token from one of its parents to not being available for a new token until this parent has sent the token back to its own parent. This requires being more subtle in how we define the election-related variables on the children.

As a consequence, we need the fourth value `F` to overcome a tricky situation. Recall that when a node u sends back the token to its parent w, it first waits for its children who won the negotiation (*i.e.* such that $\texttt{play}_v = \texttt{W}$) to reset their variable $\texttt{play}_v$ to `P`, so that they will be able to negotiate during the next circulation round. But in triangle configurations, it might be that one such node v with $\texttt{play}_v = \texttt{W}$ has a common ancestor, with its parent u, as depicted in

Figure 5. In such a situation, v should not update $\texttt{play}_v = \texttt{P}$. Indeed if it does, then from w's perspective, it is exactly as if we never introduced the value W: it has one child who won the token and is at P. In particular, we can now create a livelock at the level of w.

But v can neither keep its value at W. That would create a deadlock with its parent u. Therefore, v **fakes** resetting its variable $\texttt{play}_v$, by setting it to F, which is understood by its parent with $\texttt{tok}_u = \uparrow$ as a P value, but both v and its grandparent w do recall that v has already had the token. This requires that as soon as possible, node v update its variable $\texttt{play}_v = \texttt{W}$, otherwise it could miss the opportunity to eventually reset its variable at P, notably if w sets $\texttt{tok}_w = \uparrow$.

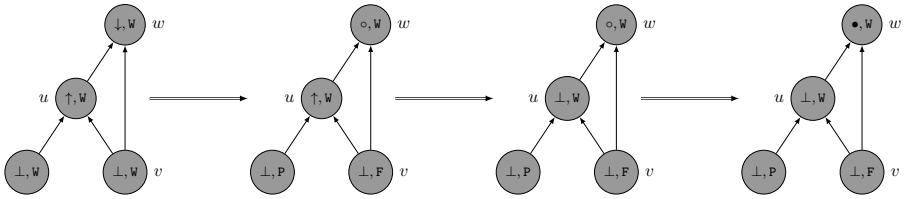

Figure 5: Cleaning children before sending up the token in a triangle: $\texttt{play}_v = \texttt{F}$.

Although the alternating between two colors and the state $\texttt{play}_v = \texttt{W}$ share the common goal, which is to prevent a node to have several times the token, they are relevant in totally different situations. The alternation between two colors is built to prevent a node from receiving twice the same token, from parents that might be totally unrelated in the DODAG. On the other hand, the value W prevents a node from oscillating between two colors, from two different parents, which would block the circulation of the token.

3.2 Rules of our algorithm

Due to the lack of space, the formal descriptions and subtleties of the rules of our algorithm are reported in Appendix B. Figure 6 presents how the different rules follow each other during the execution of the algorithm, one rule, specific to the root is not presented ni this section. For the sake of readability, we have grouped the rules according to the task to which they relate: error handling, negotiation to choose the unvisited child, returning the token back to the parent, sending the token to a child.

Error. As hinted above, some combinations of the variable tok should not occur on a node and its children. To repair such errors, the principle is that the parent trusts its child, in the sense that if both v and u believe they have a token, then v forgets its token, and repairs itself by setting $\texttt{tok}_v = \downarrow$, acknowledging the fact that there is a token below it. The illegal pairs are detected by the predicate $\texttt{Er}(v)$ and the correctness of the state of the token of node v is handled by Rule $\mathbb{E}_{\texttt{TrustChild}}$. All the other rules of the algorithm suppose that node v is not in such an error. For homogeneity and readability, we constrained the other rules with $\neg\texttt{Er}(v)$.

Negotiation. In a negotiation, two types of nodes are involved, the parent v which owns the token, and the unvisited children involved in the negotiation (we call these children "players" formally defined by the set $\texttt{Players}(v)$).

Before describing the negotiation itself, let's explain how the nodes involved in the negotiation can leave it. The parent can leave the negotiation phase in two ways. The first is captured by Rule $\mathbb{R}_{\texttt{OfferUp}}$, when the parent observes that the set of $players(v)$ is empty, which means that it has finished circulating below itself. As a consequence, v returns the token to its own

parent ($\texttt{tok}_v = \uparrow$). The second possibility happens when node v has exactly one child u in $\texttt{Players}(v)$, after verifying some tricky problems (see Appendix B), v can send the token to u, to achieve that, v puts $\texttt{tok}_v := \downdownarrows$, thanks to Rule $\mathbb{R}_{\texttt{Give}}$. The first action that v takes, after u has actually received the token, is switching its variable $\texttt{tok}$ to $\circlearrowleft$ to inform its other unvisited children who lost the negotiation (aka the losers) that the negotiation phase is terminated, this is made by Rule $\mathbb{R}_{\texttt{NewPlay}}$. So the losers can change theirs variables $\texttt{play}_w = \texttt{P}$ and become players again when v will receive the token back (see Rule $\mathbb{R}_{\texttt{ReplayD}}$). Let us now consider the children $u \in \texttt{Players}(v)$, u quits the negotiation if it quits $\texttt{Players}(v)$. The node u can do this in three ways, u detects an inconsistent behavior in his neighborhood (see Rule $\mathbb{R}_{\texttt{FakeWin}}$), u wins negotiation (see Rule $\mathbb{R}_{\texttt{Win}}$), u loses in the negotiation (see Rule $\mathbb{R}_{\texttt{Lose}}$).

Now let us focus on the negotiation itself. When a node v has finished receiving the token (i.e., the parent of v has released the token $\texttt{tok}_{p(v)} = \downarrow$, Rule $\mathbb{R}_{\texttt{Drop}}$), it can begin the negotiation (execution of Rule $\mathbb{R}_{\texttt{Nego}}$). When all players have announced their binary value (see Rule $\mathbb{R}_{\texttt{NewBit}}$), v announces the highest value (see Rule $\mathbb{R}_{\texttt{maxPos}}$). After that, some players lose the negotiation and others remain players. If there is only one player left, the negotiation is over, otherwise v continues the negotiation (see Rule $\mathbb{R}_{\texttt{NewPh}}$).

Reception of the Token from a Child. Now consider the actions when a child u offers the token to its parent v by setting $\texttt{tok}_u = \uparrow$. The node v needs to be sure that it does not have any other children involved in a token circulation. If it has, then taking the token would lead to an inconsistent configuration, where one token is above an other one. Actually, u may have several parents related to each other, and only the deepest should take the token. If v can take the token, it executes the Rule $\mathbb{R}_{\texttt{Receive}}$ signifying to u that it has taken the token. Thus, when u is sure that v has taken the token, u can release the token and v can resume negotiating with its remaining players (Rule $\mathbb{R}_{\texttt{ReNego}}$).

End of the Circulation. After it has offered the token to its parent, a node v which is not the root should eventually drop the token, and set $\texttt{tok}_v = \perp$ (Rule $\mathbb{R}_{\texttt{Return}}$). Before that, it must check that all its children have correctly updated their variable from $\texttt{W}$ to $\texttt{P}$ to assure that when a token of the other color will come, all those nodes will be available for a negotiation phase.

In a similar situation, where all of its children have repaired themselves, the root does not drop the token, for it would simply destroy it. When the root is in such a situation, it means that the current circulation round is terminated, and one other can start. To achieve that the root changes its color, and thus if the circulation has already stabilized, the root has one color and all the node have the other color (*i.e.* all the other nodes are unvisited).

Let us consider a node v having parents that are sending the token higher and no other parent involved in a token circulation. If a node v is not ready to play again, then v cleans its variable $\texttt{play}_v$ to be ready for the next circulation round (Rule $\mathbb{R}_{\texttt{ReplayUp}}$).

If v has some parents sending the token higher, and other parents which do not have terminated their token circulation, it must update its variable $\texttt{play}_v = \texttt{F}$ (Rule $\mathbb{R}_{\texttt{Fake}}$).

Finally, we must design a rule to cancel the effect of updating one's variable at $\texttt{F}$, to reset it at $\texttt{W}$, without Rule $\mathbb{R}_{\texttt{ReWin}}$, the token could not be sent higher.

To summarize. Figure 6 presents the scheme of an execution of our algorithm. It presents the update of the variables of one node v by the different rules of our algorithm.

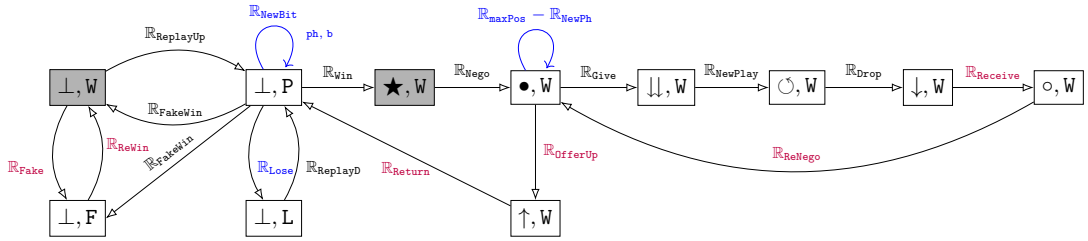

Figure 6: Chain of the states taken by a node during the execution of the algorithm. The update of $\mathtt{c}_v$ is represented by color gray. Rules in blue corresponds to the negotiation. Rules in red correspond to the sending of a token from a child to a parent. Rules in black correspond to the sending of a token from a parent to a child.

4 Correctness

Due to the lack of space, the details of the correctness of our algorithm are postponed to Appendix C, but let us draw here a roadmap of it. We prove that our algorithm is a self-stabilizing algorithm for the token circulation problem, and we also prove that it is optimal in terms of memory, *i.e.* that any algorithm solving $\mathcal{TC}$ in $\mathcal{S}$ requires $\Omega(\log\log n)$ bits per nodes, even under less general hypothesis than ours.

This correctness is subdivided into four parts. First, we establish that maximal executions of our algorithm are infinite (Appendix C.1). Formally, we prove that in any configuration, there is at least one enabled node. This part of the proof is static, in the sense that we do not need to consider an execution of the algorithm but only isolated configurations. It boils down to a syntactical analysis of the guards of the rules of our algorithm (Algorithm 2), and to an induction for one specific case.

Although the previous does not need to consider executions, the two following parts of the proof are trickier. We aim at proving that eventually, one token fairly circulates in the network, in a self-stabilizing way. Therefore, we have to consider configurations in which several token exist in the network. To do so, we introduce a *ad hoc* structure which corresponds to the paths from the root to the tokens in the network, which is a DODAG. More precisely, it is a sub-DODAG of the DODAG representing the network. But there may exist some tokens that are not connected to the root by an admissible path. For these tokens, we also define this sub-DODAG structure, but we distinguish these tokens by saying that they are not pointed by the root. All the reasoning of the two following parts rely on these sub-DODAGs structures.

Secondly, we establish at once two crucial properties for the validity of our algorithm (Appendix C.2). The first property is that if a token is not pointed by the root of the network, then it can only circulate a finite number of times before being blocked, or deleted. This guarantees that no scheduler, even unfair, can create an execution in which the token pointed by the root is never activated. The second property is that even the token pointed by the root is restricted in its circulation, in the sense that it can only circulate a finite number of times before it reaches the root again, and the root "resets" it by the execution of a rule specific to the root, $\mathbb{R}^r_{\texttt{NewDFS}}$. The combination of these two properties, and of the liveness established in Appendix C.1 leads to the guarantee that there is an infinity of finite circulations in any maximal execution of our algorithm. To prove both properties, we define a potential function, which associates a positive value (a weight) to each token in any configuration. Then, we establish that any computing step decreases the weight of the token, unless the token is reset by an execution of $\mathbb{R}^r_{\texttt{NewDFS}}$. This establishes that a token cannot circulate indefinitely, since any step it takes makes a positive

quantity decrease, unless this token is anchored at the root and there are an infinity of new circulations. Since we also prove that no token can be created, this guarantees that at some point, although several tokens may exist, only one is activated, the one which is pointed by the root.

Thirdly, we establish that our algorithm behaves correctly (Appendix C.3). By "behaves correctly" we mean that the token eventually fairly visit all the nodes of the network. To establish that, we largely use the previous results. In particular, we start the reasoning at a moment where the algorithm has already partly converged, in the sense that all the tokens except the one pointed by the root are not activated anymore. Then, we first establish some additional stability properties on the underlying structure of the DODAG which corresponds to the path from the root to the token. Then, after having properly defined what "to have the token" is, we establish that if one node receives the token from one of its parent, then all of its children that are able to receive the token will actually receive it at some point. On the other hand, we prove that if some children of one such node cannot receive the token, then after one or two additional circulations of the token, then these children will be able to receive it, and by the previous, will receive it. Therefore, circulation after circulation, the token visits nodes in deeper and deeper parts of the network. By induction, we finally prove that the token eventually fairly visits all the nodes of the network, indefinitely. In particular, this guarantees that all the other token were reached by the token pointed by the root, and merged with it.

Joined, these three first parts of the proof establish that our algorithm is a self-stabilizing algorithm for the fair token circulation in arbitrary DODAGs.

Finally, we extend the results of [3] to prove that the space complexity of our algorithm is optimal (see Appendix C.4). More precisely, we prove that no algorithm can solve the fair token circulation problem using $o(\log\log n)$ bits of memory per node, even under least challenging hypothesis than ours. We prove that even under a central strongly fair scheduler, or a synchronous scheduler, on a tree topology, and without the requirement of being self-stabilizing, no algorithm can solve the fair token circulation problem in anonymous network. The result established in [3] concludes to the space optimality of our algorithm.

5 Conclusion

In this paper, we have presented a memory-optimal self-stabilizing algorithm for the fair token circulation in arbitrary DODAGs under the unfair scheduler. Our algorithm works in the state model where no information is given on the neighbors, and requires only $\Theta(\log\log n)$ bits per node. To our knowledge, this is the first self-stabilizing algorithm achieving this space complexity in this model for graphs of arbitrary degree.

Of course, the construction of a DODAGs with the same space complexity remains an open question. Beyond the issue of DODAGs, our algorithm is providing hope that it might be possible to design self-stabilizing token circulation algorithms for arbitrary networks with space-complexity $O(\log\log n)$ bits per node. The design of such algorithms requires to overcome at least two problems: the presence of more than one root, and the symmetry caused by the presence of cycles. The presence of multiple roots is an issue which may not be too dramatic, as it may be possible to let several tokens circulate, one per root, and to remove the tokens one by one until a single token remains. The symmetries caused by the presence of cycles appear to cause severe difficulties, and our current knowledge is insufficient to guarantee that a space-complexity of $O(\log\log n)$ bits per node can be achieved under such symmetries.

Such an algorithm with space-complexity $\Theta(\log\log n)$ bits per node, solving token circulation on arbitrary graphs would be a valuable toolbox which could be used to solve other problems, such as leader election, spanning tree construction, *etc.*

References

[1] Lélia Blin, Fadwa Boubekeur, and Swan Dubois. A self-stabilizing memory efficient algorithm for the minimum diameter spanning tree under an omnipotent daemon. *J. Parallel Distributed Comput.*, 117:50–62, 2018.

[2] Lélia Blin, Swan Dubois, and Laurent Feuilloley. Silent mst approximation for tiny memory. In *Stabilization, Safety, and Security of Distributed Systems - 22nd International Symposium, SSS 2020*, pages 118–132, 2020.

[3] Lélia Blin, Laurent Feuilloley, and Gabriel Le Bouder. Optimal space lower bound for deterministic self-stabilizing leader election algorithms. *Discret. Math. Theor. Comput. Sci.*, 25, 2023. doi:10.46298/dmtcs.9335.

[4] Lélia Blin, Maria Potop-Butucaru, Stéphane Rovedakis, and Sébastien Tixeuil. A new self-stabilizing minimum spanning tree construction with loop-free property. In *23rd International Symposium on Distributed Computing (DISC 2009)*, pages 407–422, 2009.

[5] Lélia Blin and Sébastien Tixeuil. Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative. *Distributed Computing*, 31(2):139–166, 2018. doi:10.1007/s00446-017-0294-2.

[6] Lélia Blin and Sébastien Tixeuil. Compact self-stabilizing leader election for general networks. *J. Parallel Distributed Comput.*, 144:278–294, 2020. doi:10.1016/j.jpdc.2020.05.019.

[7] GM Brown, MG Gouda, and CL Wu. Token systems that self-stabilize. *IEEE Transactions on Computers*, 38:845–852, 1989.

[8] JE Burns and J Pachl. Uniform self-stabilizing rings. *ACM Transactions on Programming Languages and Systems*, 11:330–344, 1989.

[9] Alain Cournier, Stéphane Devismes, Franck Petit, and Vincent Villain. Snap-stabilizing depth-first search on arbitrary networks. *The Computer Journal*, 49(3):268–280, 2006.

[10] Alain Cournier, Stéphane Devismes, and Vincent Villain. Light enabling snap-stabilization of fundamental protocols. *ACM Transactions on Autonomous and Adaptive Systems*, 4(1):6:1–6:27, 2009.

[11] Ajoy Kumar Datta, Colette Johnen, Franck Petit, and Vincent Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distributed Computing*, 13(4):207–218, 2000. doi:10.1007/PL00008919.

[12] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974. doi:10.1145/361179.361202.

[13] S Dolev, A Israeli, and S Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7:3–16, 1993.

[14] Swan Dubois and Sébastien Tixeuil. A taxonomy of daemons in self-stabilization. arXiv: 1110.0334, 2011.

[15] S Ghosh. An alternative solution to a problem on self-stabilization. *ACM Transactions on Programming Languages and Systems*, 15:735–742, 1993.

[16] Mohamed G. Gouda and F. Furman Haddix. The stabilizing token ring in three bits. *J. Parallel Distrib. Comput.*, 35(1):43–48, 1996. doi:10.1006/jpdc.1996.0066.

[17] ST Huang and NS Chen. Self-stabilizing depth-first token circulation on networks. *Distributed Computing*, 7:61–66, 1993.

[18] C Johnen and J Beauquier. Space-efficient distributed self-stabilizing depth-first token circulation. In *Proceedings of the Second Workshop on Self-Stabilizing Systems*, pages 4.1–4.15, 1995.

[19] J. J. Martin. Distribution of the time through a directed, acyclic network. *Operations Research.*, 13(1):46–66, 1965. doi:10.1287/opre.13.1.46.

[20] Franck Petit and Vincent Villain. Optimal snap-stabilizing depth-first token circulation in tree networks. *Journal of Parallel Distributed Computing*, 67(1):1–12, 2007.

[21] P. Thubert and M. Richardson. Routing for rpl (routing protocol for low-power and lossy networks) leaves. RFC 9010, RFC Editor, April 2021.

[22] Tim Winter, Pascal Thubert, Anders Brandt, Jonathan W. Hui, Richard Kelsey, Philip Alexander Levis, Kris Pister, René Struik, Jean-Philippe Vasseur, and Roger K. Alexander. RPL: ipv6 routing protocol for low-power and lossy networks. *RFC*, 6550:1–157, 2012.

A Appendix on Model and formalisations

A.1 DODAG formalisation

Let us be more formal:

Definition 2 (*Destination-Oriented Directed Acyclic Graph, DODAG*)

A DODAG is a tuple $D = (G, \rightarrow)$ where $G = (V, E)$ is a graph and $\rightarrow$ is a relation between the nodes of G which:

- *covers the edges of G: $(u \rightarrow v \vee v \rightarrow u) \iff \{u, v\} \in E$,*
- *is acyclic: there does not exist any path from one node v to itself w.r.t. $\rightarrow$, and*
- *is rooted: there exists one node $r \in V$ such that $\forall v \in V, v \rightarrow^* r$.*

The nodes of D are V, and the edges of D are $\rightarrow$.

By the acyclic property, such a node r is unique, and is called the root *of D. Also by the acyclic property, for any edge $\{u, v\} \in E$, we have either $u \rightarrow v$ or $v \rightarrow u$ but not both.*

In practice, the relation $\rightarrow$ is implemented at the layer of the local port numbers of the nodes. A DODAG is a graph in which the port numbers can be split into two disjoint sets $\texttt{port}^+$ and $\texttt{port}^-$, and the relation $\rightarrow$ is defined by $u \rightarrow v \iff \texttt{port}_u(v) \in \texttt{port}^+$.

In the latter, we often need to consider, on one node v, the set of its neighbors that precedes it, and the set of neighbors that it precedes. We call those sets the parents and the children of the node, by analogy with the lexical field relative to trees.

Note that contrary to what is possible in trees, nodes may have several parents in DODAGs.

Definition 3 (*Parental Relationships in a DODAG*)

Let $D = (G, \rightarrow)$ be a DODAG.

We call parents *of v in D the set $\mathcal{P}(v) = \{u \in N_v : v \rightarrow u\}$.*

We call children *of v in D the set $\mathcal{C}(v) = \{u \in N_v : u \rightarrow v\}$.*

Remark 1

The information of the set of the parents of each node is equivalent to the information of the relation $\rightarrow$. In practice, we define a DODAG by the parental relationship it induces more than by expliciting the relation $\rightarrow$.

It will be useful in the latter to reason on sub-structures of the DODAG in which our algorithm is executed. In most cases, the sub-structures considered also have the desirable property of being DODAGs. A sub-DODAG of D is any DODAG whose nodes and edges are a subset of those of D.

Definition 4 (*Sub-DODAG of G, Anchor*)

A tuple $D' = (G', \rightarrow')$ is a sub-DODAG of $D = (G, \rightarrow)$ if

- *D' is a substructure of D: $(V' \subseteq V) \wedge (\rightarrow' \subseteq \rightarrow) \wedge (\rightarrow' \subseteq V' \times V')$.*
- *D' is a DODAG*

Remark that the root of D' may not be the same than the root of D. To avoid being misleading, we call the root of a sub-DODAG an anchor.

Remark 2

Following what was stated in Remark 1, we will not use relation $\rightarrow'$ to define sub-DODAGs in practice, and rather use the underlying set of parents and children, $\mathcal{P}_{D'}$ and $\mathcal{C}_{D'}$.

One typical example of sub-DODAG is the restriction of the DODAG to the nodes which can reach one particular node (other than the root) in $(G, \rightarrow)$.

Definition 5 (G_a *DODAG under* a)

Let $(G, \rightarrow)$ be a DODAG, and let $a \in V$.

We denote by V_a the nodes that can reach a by $\rightarrow$: $V_a = \{v \in V : v \rightarrow^* a\}$. We denote by E_a and $\rightarrow_a$ the restrictions of E and $\rightarrow$ to nodes of V_a: $E_a = E \cap \mathcal{P}_2(V_a)$ and $\rightarrow_a = \rightarrow \cap (V_a \times V_a)$.

We denote by $(G_a, \rightarrow_a)$, or simply G_a, and call the DODAG under a, the sub-DODAG $((V_a, E_a), \rightarrow_a)$.

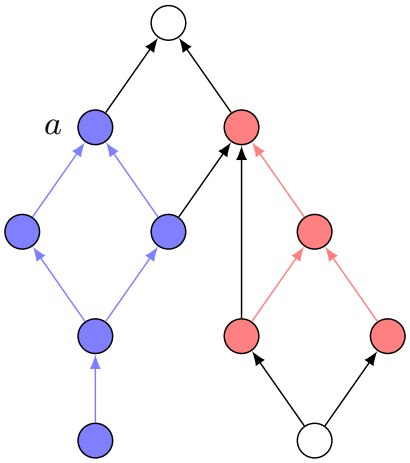

Figure 7: Example of a DODAG under the anchor a in blue, and of a sub-DODAG of G in red

Although DODAGs are the structure in which our algorithm and our proofs are established, it is often simpler to reason on linear structures than on DODAGs. In the latter, we will consider branches of DODAGs, which are descending paths from one node to one of its descendants.

Definition 6 (*Branch of a DODAG*)

We call branch of a DODAG $D = (G, \rightarrow)$ a path of nodes in D (which is a path of nodes in G w.r.t. $\rightarrow$). For the sake of readability, we denote branches by starting at the highest node. For example, if $w \rightarrow v \rightarrow u$ is a path in D, then we say that uvw is a branch of D.

We denote branches by the letter $\mathcal{B}$.

A branch $\mathcal{B} = v_0 v_1 \dots v_k$ of a DODAG D is maximal if v_0 is the root (or the anchor) of D, and if v_k does not have any children in D.

Given a sub-DODAG of D, it will be interesting to consider the longest branches it shares with D. These longest branches are called projections of the sub-DODAG on branches.

Definition 7 (*Projection of a sub-DODAG on a Branch*)

Let us consider a DODAG D, and a sub-DODAG of D with anchor a, D_a. Let $\mathcal{B} = v_1 v_2 \dots v_k$ be a maximal branch of G_a, the DODAG under a. In particular, $v_1 = a$.

We denote $D_a(\mathcal{B})$, and call the projection of D_a on $\mathcal{B}$ the longest branch of D_a that coincides with $\mathcal{B}$:

$D_a(\mathcal{B}) = v_1 v_2 \cdots v_j$ where $\forall i < j, v_{i+1} \in \mathcal{C}_{D_a}(v_i)$ and $v_{j+1} \notin \mathcal{C}_{D_a}(v_j)$ (recall that $v \in \mathcal{C}_{D_a}(u) \iff v \to_{Da'} u$).

Remark 3 (*DODAG* $\equiv$ $\mathtt{D}^o$)

In the latter, we consider different types of DODAGs, depending on their purpose. For the sake of readability, we will use $\mathtt{D}^o$ as an alias for the classical notion of DODAG, and $\mathtt{CD}^o$, $\mathtt{WD}^o$, $\mathtt{FD}^o$ as aliases for particular types of DODAG.

A.2 Bit-by-Bit Communication of Identifier

We use a similar technique as [5], but for the sake of simplicity, we slightly modified this function for this work, and send the identifier bit-by-bit, which does not change the asymptotic complexity. Essentially, nodes do not communicate their entire identifier at once, but when asked for, they send to their neighbors the value of their i-th bit (which is 0 or 1). Although the value of the bit is 1-bit long, we must take into account the fact that the position of the bit which is asked, i, is now part of the message.

This position variable i takes value between 1 and the maximal length of an identifier, which is in $\Theta(\log n)$. Therefore, it is encoded on $\Theta(\log \log n)$ bits. To simplify, we suppose that all nodes are given a local function `Bit` that associates to each position, the value of the bit at this position in their identifier. If the position asked is bigger than the length of their identifier, then the function simply returns $\perp$. Suppose for example that node v has identifier $\mathtt{id}_v = 1011$. Then we define the function $\mathtt{Bit}_v$ by:

$$\mathtt{Bit}_v(i) := \begin{cases} 1 & \text{if} \quad \text{i=1} \\ 0 & \text{if} \quad \text{i=2} \\ 1 & \text{if} \quad \text{i=3} \\ 1 & \text{if} \quad \text{i=4} \\ \perp & \text{if} \quad i > 4 \end{cases}$$

Nodes keep storing their own identifier in their immutable memory, but what is communicated to the neighbors through the immutable memory is the tuple (i, bit), which requires $\Theta(\log \log n)$ bits. Remark that since the identifiers are globally unique, the different functions `Bit` are also globally unique, although they can coincide for some values of i.

A.3 Well-Founded Sets

Recall that an ordered set $(M, \preceq)$ is well-founded if there does not exist infinite sequence of elements of M, $v_0 v_1 \ldots$ such that $\forall i \in \mathbb{N}, v_{i+1} \prec v_i$. Such relations are very desirable to prove termination of algorithms, or their convergence. In the latter, we consider an arbitrary well-founded set $(M, \preceq)$.

Given $(M, \preceq)$, we can define an order on tuples of elements of M, which is itself well-founded when restricted to sequences of length at most k, for any k.

Definition 8 (*Lexicographic Order*)

For any $k \geq 1$ we define the lexicographic order on M^k induced by $\preceq$, and denote $\preceq^k_{lex}$, by:

$$(u_1, \ldots, u_k) \prec^k_{lex} (v_1, \ldots, v_k) \iff \exists i \in [1, k] : \begin{cases} (u_1, \ldots, u_{i-1}) = (v_1, \ldots, v_{i-1}) \\ u_i \prec v_i \end{cases}$$

In our proof, we are sometimes led to compare sequence of elements of M which have variable length. We extend the definition of lexicographic order to sequences of arbitrary length, which corresponds to the alphabetical order. The resulting order is itself well-founded.

Definition 9 (*Alphabetical Order*)

The lexicographic order on (possibly empty) sequences of elements of M induced by $\preceq$, is denoted $\preceq_\alpha$ and defined by:

$$(u_1, \ldots, u_k) \prec_\alpha (v_1, \ldots, v_l) \iff \begin{cases} \exists i \leq \min k, l : (u_1, \ldots u_i) \prec^i_{lex} (v_1, \ldots v_i) \\ \vee \\ k < l \wedge (u_1, \ldots, u_k) = (v_1, \ldots, v_k) \end{cases}$$

A.4 Token Circulation

In this article, we consider the problem of fair token circulation, which requires that one unique token perpetually circulates through the network.

The part of the specification of the token circulation problem which corresponds to the existence and unicity of the token is not complicated to state, and is basically what is made in [3] when the authors give a specification for the leader election problem. It boils down to defining a local predicate on nodes, which represents the fact that the token is held by that node, and guaranteeing that the predicate is evaluated to true on exactly one node in each configuration. This first predicate will be denoted by T, for token.

It is a bit harder to formally specify the fairness circulation property. For a token circulation to be fair, one could expect that all nodes have the token at the same rate. But the more a node has children, the more it will receive it back from its children before sending it to its other children. We must be more specific on what fairness is. We define a more restrictive local predicate, which represents the fact that the node has the token *and* is allowed to use it to access the resource, the service... This second predicate will be denoted by R, for resource access.

Now that we have this second predicate, we only have to guarantee that, between two configurations in which one particular node has access to the resource, then all the other nodes also have it, and exactly once. In practice, the node that we consider to delimit fairness is the root. A sub-execution in which the root receives the token for access, then drops it, and then receives it back for access, is called a *round*, or a *circulation round*.

Definition 10 (*Round according to R*)

Let R be a local predicate, and let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be an execution.
A sub-execution $\epsilon' = \gamma_i \to \cdots \to \gamma_j$ of ϵ is a round according to R if:

- $\neg R^{\gamma_{i-1}}(r)$
- $R^{\gamma_{j+1}}(r)$
- $\exists i \leq k < j$ *such that*
  - $\forall t \in [i, k], R^{\gamma_t}(r)$
  - $\forall t \in [k+1, j], \neg R^{\gamma_t}(r)$

Note that by construction, circulation rounds perfectly follow each other in any execution.

We say that a circulation round is fair if for any node v, there exists exactly one continuous sequence of configurations in which v has access to the resource.

Definition 11 (*Fair Round according to R*)

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be an execution, and let $\epsilon' = \gamma_i \to \cdots \to \gamma_j$ be a round according to R.

ϵ' is fair if for any $v \in V$, there exist $i \leq t_1 < t_2 \leq j$ such that

- $\forall t \in [t_1, t_2], R^{\gamma_t}(v)$
- $\forall t \in [i, j] \setminus [t_1, t_2], \neg R^{\gamma_t}(v)$

We can now formally define the specification of Fair Token Circulation:

Specification 1 (*Token Circulation*)

$\mathcal{TC}$ (Token Circulation) is defined by the existence of two local predicates T and R such that $\neg T \Rightarrow \neg R$.

A configuration γ is correct if it contains one unique token

$$U_{\mathcal{TC}}(\gamma) \equiv \exists! v \in V : T^{\gamma}(v)$$

An execution $\epsilon = \gamma_0 \to \cdots$ is correct if

1. *All its configurations are correct: $\forall i \geq 0, U_{\mathcal{TC}}(\gamma_i)$.*
2. *ϵ is infinite and can be divided into an infinite number of rounds according to R.*
3. *All of these rounds are fair*

Definition 12 (*Circulation Rounds*)

In the context of Token Circulation, we call Circulation Round *a round according to the predicate R.*

B Appendix on our Algorithm

B.1 Common Sets

To define the rules of our algorithm, some sets are especially useful. An important distinction that nodes almost systematically do is the difference between their parents with the same color, and their parents with the other color.

$$\mathcal{P}_{\texttt{eq}}(v) = \{u \in \mathcal{P}(v) \mid \texttt{c}_u = \texttt{c}_v\} \tag{1}$$

$$\mathcal{P}_{\texttt{neq}}(v) = \{u \in \mathcal{P}(v) \mid \texttt{c}_u \neq \texttt{c}_v\} \tag{2}$$

Similarly, nodes distinguish their children with the same color from their children with the other color.

$$\mathcal{C}_{\texttt{eq}}(v) = \{u \in \mathcal{C}(v) \mid \texttt{c}_u = \texttt{c}_v\} \tag{3}$$

$$\mathcal{C}_{\texttt{neq}}(v) = \{u \in \mathcal{C}(v) \mid \texttt{c}_u \neq \texttt{c}_v\} \tag{4}$$

One other central set for the negotiation is the set of the nodes with whom the parent negotiates. The parent negotiates with its children that do not have the same color, and such

that $\texttt{play}_u = \texttt{P}$. A node that negotiates is not supposed to have any children u such that $\texttt{tok}_u \neq \bot$, and a specific rule is designed for such erroneous situations. Yet, to simplify some reasoning, we also add as a constraint the fact that v only negotiates with its children such that $\texttt{tok}_u = \bot$. In the latter, we call these nodes the players of v.

$$\texttt{Players}(v) = \{u \in \mathcal{C}_{\texttt{neq}}(v) \mid \texttt{tok}_v = \bot \wedge \texttt{play}_u = \texttt{P}\} \tag{5}$$

B.1.1 Rules of the Algorithm

In this section, we present the predicates, actions, and rules of our algorithm. For the sake of readability, we have grouped the rules according to the task they relate to. In Section B.1.1, we present the rule dedicated to deal with inconsistencies of the variable tok. In Section B.1.3, we present the negotiation process, based on the identifiers of the children of the node holding the token. In Section B.1.2 we present the rules that make nodes quit that negotiation process. In Section B.1.4 we present the rules which reset variables of nodes to a proper value after the end of the negotiation process. In Section B.1.5 we present the rules that allow one node to receive the token back from one of its children. In Section B.1.6 we present the rules that are executed when one circulation below a node is terminated, whether the node is the root or another node. Finally, in Section B.1.7, all the rules are gathered to allow an easy access during the reading of the proofs.

Error As hinted above, some combinations of the variable tok should not occur on a node and its children. In general, a node v that has the token, or which is very near a token, *i.e.* with $\texttt{tok}_v \in \{\bigstar, \bullet, \downdownarrows, \circlearrowleft, \circ, \uparrow\}$, should only have children u with $\texttt{tok}_u = \bot$, with a few exceptions.

Indeed, if $\texttt{tok}_v \in \{\downdownarrows, \circlearrowleft\}$, then v is allowed to have a child such that $\texttt{tok}_u = \bigstar$, the child to which it just gave the token.

If $\texttt{tok}_v = \circ$, then v is allowed to have a child such that $\texttt{tok}_u = \uparrow$, the child from which it is receiving the token.

One other exception is when $\texttt{tok}_v = \uparrow$. In this case, v is sending the token back to its parent, which means that it is on its way to set $\texttt{tok}_v = \bot$. We do not consider any error for such nodes, since it would not be very effective, and could create an error at the level of its own parent.

To repair such errors, the principle is that the parent trusts its child, in the sense that if both v and u believe they have a token, then v forgets its token, and repairs itself by setting $\texttt{tok}_v = \downarrow$, acknowledging the fact that there is a token below it.

The illegal pairs are detected by the predicate $\texttt{Er}(v)$ and the correctness of the state of the token of node v is handled by the rule $\mathbb{E}_{\texttt{TrustChild}}$.

$$\begin{aligned} \texttt{Er}(v) \quad \equiv \quad & \big(\texttt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \neq \bot\big) \\ \vee \quad & \big(\texttt{tok}_v \in \{\downdownarrows, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \bigstar\}\big) \\ \vee \quad & \big(\texttt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \uparrow\}\big) \\ \longrightarrow \quad & \texttt{tok}_v :=\downarrow \end{aligned} \tag{6}$$

$\mathbb{E}_{\texttt{TrustChild}}$: for all $v \in V$

$$\texttt{Er}(v) \longrightarrow \texttt{tok}_v :=\downarrow$$

All the other rules of the algorithm suppose that node v is not in such an error. For homogeneity and readability, we constrained all rules with $\neg\texttt{Er}(v)$, even rules which suppose that $\texttt{tok}_v \in \{\bot, \downarrow, \uparrow\}$.

B.1.2 End of the Negotiation Phase

Before showing how the negotiation phase is implemented, let us first explain how nodes may quit this phase. In a negotiation, two types of nodes are involved, the parent v which has the token, such that $\texttt{tok}_v = \bullet$, and the children u elements of $\texttt{Players}(v)$.

The parent can quit the negotiation phase by two means.

Rule $\mathbb{R}_{\texttt{OfferUp}}$ The first one is when the parent observes that all of its children in $\texttt{Players}(v)$ have been visited, which means that it ended the circulation below itself. When this happens the parent sends the token back to its own parent.

Before doing so, the parent must first be assured that it is not in a situation as depicted in the last configuration of Figure 5. If v has a child u such that $\texttt{play}_u = \texttt{F}$, then it is a node which actually won the token, but faked being reset to P to another parent p such that $\texttt{tok}_p = \uparrow$. If v is the last parent of u, and switches $\texttt{tok}_v = \uparrow$, then u will never be reset to P, which breaks the fairness of the token circulation. Therefore, before executing this action, v waits that all of its children have quit the value F.

The predicate WaitSib corresponds to this requirement.

$$\texttt{WaitSib}(v) \equiv \exists c \in \mathcal{C}_{\texttt{eq}}(v) : \texttt{play}_c = \texttt{F} \tag{7}$$

$\mathbb{R}_{\texttt{OfferUp}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \neg\texttt{WaitSib}(v) \wedge (\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \in \{\texttt{W}, \texttt{F}\}) \longrightarrow \texttt{tok}_v := \uparrow$$

Note that even if the root has no parent, it may execute rule $\mathbb{R}_{\texttt{OfferUp}}$ when it does not have any more children to visit. Since some operations subsequent to having finished circulation below oneself are identical for the root and the other nodes, we factorize the rules as far as possible.

Rule $\mathbb{R}_{\texttt{Give}}$ The second possibility for a parent to quit the negotiation is to decide to offer the token to one of its children. This happens when node v has exactly one child in $\texttt{Players}(v)$.

But there is one other situation to which we must give attention. Suppose that due to a wrong initialization of the network, no node belongs to $\texttt{Players}(v)$, but the circulation is not over yet, for one or several children of v are losers, *i.e.* $\texttt{play}_u = \texttt{L}$. In this situation, node v needs to reset the variable play of these children to P, and then will perform a negotiation between them. One simple way to do that is to follow the transition diagram of tok depicted in Figure 3. No child of v will take the token offered by $\texttt{tok}_v = \downdownarrows$, but immediately after, when $\texttt{tok}_v$ is set to $\circlearrowleft$, the losers will update $\texttt{play}_u = \texttt{P}$, which will allow a further negotiation process.

Actually, v basically executes a full round of its variable tok just to refresh its wrongly initialized children. Note that this may only be caused by a corrupted initial configuration, and that after convergence, this part of the rule becomes useless.

The predicate Give describes the situations in which node v ends the negotiation by sending the token to one of its children.

$$\texttt{Give}(v) \equiv |\texttt{Players}(v)| = 1 \vee (|\texttt{Players}(v)| = 0 \wedge \exists u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u = \texttt{L}) \tag{8}$$

$\mathbb{R}_{\texttt{Give}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \texttt{Give}(v) \longrightarrow \texttt{tok}_v := \downdownarrows$$

Let us now consider the children $u \in \texttt{Players}(v)$. A child quits the negotiation if it quits $\texttt{Players}(v)$. It may do that by three means.

Rule $\mathbb{R}_{\texttt{FakeWin}}$ The first situation in which a child quits the negotiation is when it has an inconsistent parenthood. More precisely, if v has several parents that are dealing with a token. Such parents are those whose value $\texttt{tok}_p \notin \{\bot, \downarrow, \uparrow\}$.

We do not consider the nodes who are not involved in a token circulation ($\texttt{tok}_p = \bot$) neither do we consider the nodes who announce a token in their descendants ($\texttt{tok}_p = \downarrow$). We also do not consider $\uparrow$ for the same reason that it was ignored in predicate `Er`: such parent is about to drop the token to its own parent, it is therefore unnecessary to consider it.

The parents which may force v to quit the negotiation are `ParNeg`:

$$\texttt{ParNeg}(v) = \{p \in \mathcal{P}(v) \mid \texttt{tok}_p \notin \{\bot, \uparrow, \downarrow\}\} \tag{9}$$

Let us describe the situations where v does not have to quit the negotiation due to an inconsistent parenthood. First, if v has exactly one parent in $\texttt{ParNeg}(v)$, then there is no inconsistency. If v has no parent in $\texttt{ParNeg}(v)$, there is no inconsistency either.

There is one other situation that may occur in legal execution. In a triangle configuration, v has two parents and one of them is the children of the other. Then the middle one receives the token and set its variable `tok` to ★, there is a moment where v has its variable $\texttt{play}_v = \texttt{P}$ and also has two parents in $\texttt{ParNeg}(v)$, since the other parent is at ↻. This is described in Figure 8

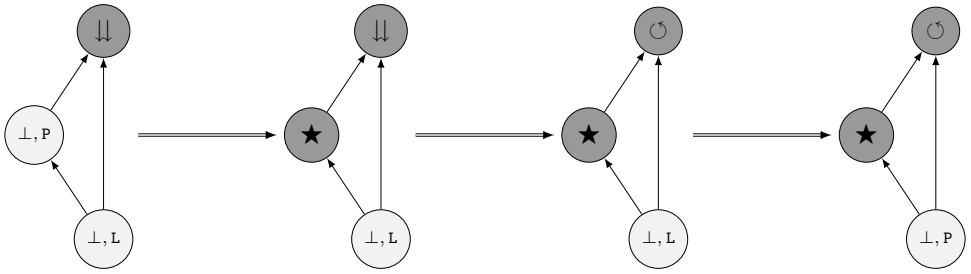

Figure 8: Node with several parents indicating a token

Formally, the situations that do not force v to interrupt the negotiation are:

$$\texttt{OkNeg}(v) \equiv \begin{cases} |\texttt{ParNeg}(v)| \leq 1 \\ \vee \\ (|\texttt{ParNeg}(v)| = 2 \wedge \{\texttt{tok}_p \mid p \in \texttt{ParNeg}(v)\} = \{↻, ★\}) \end{cases} \tag{10}$$

When a node decides to stop negotiating with its parent for inconsistency reasons, it fakes winning the token, in the sense that it becomes visited by switching its color, and updates its variable `play`, but does not actually take the token. In most cases, switching to `W` is sufficient. Hence, if before switching its color, v has a parent of the other color, such that $\texttt{tok}_p = \uparrow$, then at the next computing step, v will update $\texttt{play}_v$ to `F`, as shown in Figure 5. In order to spare one computing step, and to simplify some proofs, we set $\texttt{play}_v$ at the right value at once.

The action corresponding to the rule $\mathbb{R}_{\texttt{FakeWin}}$ is given by `FakeWin`:

$$\texttt{FakeWin}(v) \equiv \begin{cases} \texttt{c}_v & := \neg \texttt{c}_v; \\ \texttt{play}_v & := \begin{cases} \texttt{W} & \text{if } \forall p \in \mathcal{P}_{\texttt{neq}}(v), \texttt{tok}_p \neq \uparrow; \\ \texttt{F} & \text{if } \exists p \in \mathcal{P}_{\texttt{neq}}(v), \texttt{tok}_p = \uparrow; \end{cases} \end{cases} \tag{11}$$

$\mathbb{R}_{\texttt{FakeWin}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{P} \wedge \neg\texttt{OkNeg}(v) :\longrightarrow \texttt{FakeWin}(v)$$

Rule $\mathbb{R}_{\texttt{Win}}$ The second situation in which a child quits the negotiation is when it wins the negotiation. A node v wins the negotiation if its parents are not in an inconsistent configuration (which is $\texttt{OkNeg}(v)$), and if it is still at $\texttt{play}_v = \texttt{P}$ when its parent offers it the token (*i.e.* when $\texttt{tok}_p = \downdownarrows$).

The predicate `WinNeg` is dedicated to detect such situations.

$$\texttt{WinNeg}(v) \equiv \texttt{OkNeg}(v) \wedge \exists p \in \mathcal{P}_{\texttt{neq}}(v) : \texttt{tok}_p = \downdownarrows \tag{12}$$

When a node v wins the negotiation, it takes the token, and switches its color. Furthermore, it updates its variable $\texttt{play}_v$ to `W`. Contrary to how we did for `FakeWin`, we do not have to worry about the precise value of $\texttt{play}_v$ right now. Indeed, since v has a value different that $\bot$ for $\texttt{tok}_v$, v does not have to worry about the moment where it will be reactivated by its parent, which happens at the end of the circulation of its parent.

$$\texttt{Win}(v) \equiv \begin{cases} \texttt{tok}_v & := \bigstar; \\ \texttt{c}_v & := \neg\texttt{c}_v; \\ \texttt{play}_v & := \texttt{W}; \end{cases} \tag{13}$$

$\mathbb{R}_{\texttt{Win}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{WinNeg}(v) \longrightarrow \texttt{Win}(v)$$

Rule $\mathbb{R}_{\texttt{Lose}}$ The third and last situation in which a child quits the negotiation is when it loses it. The exact condition of how a child loses the negotiation will be properly defined in the following section. For now, let us just consider that there exists a predicate $\texttt{LosePar}(v, p)$, which depends on the values of $\texttt{id}_v$ and $\texttt{id}_p$, where $p \in \mathcal{P}_{\texttt{neq}}(v)$, which describes that v should lose the negotiation according to p.

A node loses the negotiation if its parents are not in an inconsistent configuration (which is $\texttt{OkNeg}(v)$, and if it does not win the negotiation (which is $\neg\texttt{WinNeg}(v)$).

All these conditions are grouped in the predicate $\texttt{LoseNeg}(v)$:

$$\texttt{LoseNeg}(v) \equiv \texttt{OkNeg}(v) \wedge \neg\texttt{WinNeg}(v) \wedge \exists p \in \mathcal{P}_{\texttt{neq}}(v) : (\texttt{tok}_p = \bullet \wedge \texttt{LosePar}(v, p)) \tag{14}$$

When v loses the negotiation, it simply sets $\texttt{play}_v = \texttt{L}$.

$\mathbb{R}_{\texttt{Lose}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{LoseNeg}(v) \longrightarrow \texttt{play}_v := \texttt{L}$$

B.1.3 Negotiation Identifier-Based

In the negotiation phase, the only children which are relevant to consider, from the parent's perspective, are the children in $\texttt{Players}(v)$. In the description of the rules, we write players instead of children in $\texttt{Players}(v)$, to facilitate the reading.

Let us first describe the general mechanism of the election, then explain some subtleties, before presenting the rules of the bit-by-bit negotiation.

One first important idea is that the parent does not execute any action unless all of its players have executed their rule. In particular, the parent waits for all of its players to be at the same value of $\texttt{ph}$ as it is.

$$\texttt{Synch}(v) \equiv \forall u \in \texttt{Players}(v), \texttt{ph}_u = \texttt{ph}_v \tag{15}$$

One other general principle, which is crucial due to the asynchrony of the system, and due to the self-stabilizing requirement, is that nodes do not correct their answer. More precisely, if one node v (parent or child in the negotiation) is in a state that may have an influence on the execution of a rule by one other node u, then v does not update its state. Even if this state is inconsistent with its identifier, for a child, or with the values of $\texttt{id}_u$ on its players, for a parent, v waits that its neighbors have finished interpreting this state, and accept the consequences of this error. Such inconsistencies are typically due to an incorrect initialization of the system. This general principle only applies to the negotiation rules, and does not prevent nodes to execute rules such as $\mathbb{E}_{\texttt{TrustChild}}$, $\mathbb{R}_{\texttt{FakeWin}}$, or $\mathbb{R}_{\texttt{Give}}$ for example.

Let us recall the general scheme of how the negotiation process happens

0. Parent v asks for the value of the first bit of its players $u \in \texttt{Players}(v)$, by setting $\texttt{id}_v = (1, \bot)$.
1. Players u of v answers with their own value $(1, \texttt{Bit}_u(1))$ (rule $\mathbb{R}_{\texttt{NewBit}}$).
2. When all its players have answered, *i.e.* when all have the right value $\texttt{ph}_u = 1$, v announces the biggest value it sees, by setting $\texttt{id}_v = (1, \texttt{b})$ (rule $\mathbb{R}_{\texttt{maxPos}}$).
3. Players u of v which have a lower value than $\texttt{b}_v$ lose the negotiation (rule $\mathbb{R}_{\texttt{Lose}}$).
4. When all of its players have $\texttt{ph}_u = 1$ and $\texttt{b}_u = \texttt{b}_v$, v asks for the value of the second bit of its remaining players, by setting $\texttt{id}_v = (2, \bot)$ (rule $\mathbb{R}_{\texttt{NewPh}}$), and so on.

A particular situation is when all the players u of v answer $(\texttt{ph}, \bot)$, *i.e.* when all the players declare that they do not have a long-enough identifier to provide a $\texttt{ph}$-th bit. This situation is not possible in correct executions, since all identifiers are distinct, we cannot reach a point where several identifiers are similar and terminated. Hence, this may happen in the early steps of an execution which starts in an inconsistent configuration. Either at least one child has an identifier actually greater than $\texttt{ph}$, and could go further, but due to an incorrect initialization, it has a wrong value for $\texttt{b}$, either due to an incorrect initialization of the parent, the negotiation process did not start at $\texttt{ph} = 1$ but further, and we could have missed the opportunity to distinct the identifiers of the nodes in $\texttt{Players}(v)$.

When this happens, the parent cannot simply increase $\texttt{ph}$, since its players formulate that they can't even reach the current value of $\texttt{ph}$. Therefore, if all of its players answer $(\texttt{ph}, \bot)$, then the correct move for the parent is to restart at $\texttt{ph}_v = 1$. Only one exception to that rule, is when it happens when $\texttt{ph}_v = 1$ already. In this situation, v would not change a single bit of information by setting $\texttt{id}_v = (1, \bot)$. We suppose that all identifiers have at least one bit, even if it is a single bit 0. Therefore, $(1, \bot)$ can never be a valid answer for a child. In consequences, without any contradiction with the previous principle stating that no possibly valid answer can be revised, players such that $\texttt{id}_v = (1, \bot)$ can actually update their state to send their actual value for $\texttt{Bit}_u(1)$. This is not true for other values of $\texttt{ph}$: if $\texttt{id}_v = (\texttt{ph}, \bot)$ with $\texttt{ph} > 1$ and $\texttt{Bit}_v(\texttt{ph}) \neq \bot$, then to avoid confusing its parent, it does not update its state.

Let us now treat the different rules corresponding to the scheme presented above.

Rule $\mathbb{R}_{\texttt{NewBit}}$ Some first conditions for a node v to answer the negotiation is that its parenthood is not inconsistent, and it is not winning the negotiation, nor losing it. This corresponds to the partial predicate $\texttt{OkNeg}(v) \wedge \neg\texttt{WinNeg}(v) \wedge \neg\texttt{LoseNeg}(v)$.

The other condition is that v is in a situation where it is its turn to answer. One first situation in which it should answer is when it does not have the same value of $\texttt{ph}$ as its negotiating parent p. The second situation is the particular case where $\texttt{ph}_v = \texttt{ph}_p = 1$, and v has answered $\bot$, which is never a valid answer, and the parent p is still waiting for its players to answer. Indeed, in an poorly initialized configuration, the parent p could have already decided which value was the highest, and thus v must not answer.

This is synthesized in predicate $\texttt{AnswerPar}$:

$$\texttt{AnswerPar}(v,p) \equiv \texttt{ph}_v \neq \texttt{ph}_p \vee \begin{cases} \texttt{ph}_p = 1 \\ \texttt{b}_p = \bot \\ \texttt{b}_v = \bot \end{cases} \tag{16}$$

We also define the predicate which excludes concurrency with $\mathbb{R}_{\texttt{FakeWin}}$, $\mathbb{R}_{\texttt{Win}}$, and $\mathbb{R}_{\texttt{Lose}}$:

$$\texttt{AnswerNeg}(v) \equiv \begin{cases} \texttt{OkNeg}(v) \wedge \neg\texttt{WinNeg}(v) \wedge \neg\texttt{LoseNeg}(v) \wedge \\ \exists p \in \mathcal{P}_{\texttt{neq}}(v) : (\texttt{tok}_p = \bullet \wedge \texttt{AnswerPar}(v,p)) \end{cases} \tag{17}$$

The action which corresponds to answering the negotiation is updating variable $\texttt{ph}_v$ to the value of our parent, and $\texttt{b}_v$ according to function $\texttt{Bit}_v$.

Remark that such a parent p is necessarily unique, due to $\texttt{OkNeg}(v)$. To simplify the notations, we consider that such parent p is given as an external information to the action which corresponds to $\texttt{AnswerNeg}$, $\texttt{Announce}$:

$$\texttt{Announce}(v) \equiv \texttt{id}_v := (\texttt{ph}_p, \texttt{Bit}_v(\texttt{ph}_p) \tag{18}$$

$\mathbb{R}_{\texttt{NewBit}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{AnswerNeg}(v) \longrightarrow \texttt{Announce}(v)$$

Rule $\mathbb{R}_{\texttt{maxPos}}$ One first condition for the parent to announce the maximal value it sees on its players is that the negotiation is not over yet, *i.e.* there are at least two players.

One second condition is that all of those players have answered, *i.e.* that they have the same value as itself for $\texttt{ph}$, which is captured by $\texttt{Synch}(v)$.

One third condition is that it does not have already announced a value, *i.e.* $\texttt{b}_v = \bot$.

There are two more subtleties. The first one is that if we have $\texttt{ph}_v = 1$ then v actually waits for all of its players to produce a non-empty answer, a value for $\texttt{b}_u$ which is not $\bot$, an invalid answer for $\texttt{ph} = 1$.

The second subtlety is that if, for a value of $\texttt{ph}$ greater than 1, all the players of v announce $\bot$, then it is irrelevant to keep increasing phases, and actually irrelevant to answer $\bot$ as well. In this situation, the correct move is to restart the negotiation from $\texttt{ph}_v = 1$, which will be dealt by rule $\mathbb{R}_{\texttt{NewPh}}$.

The last condition and the subtleties are described by predicate $\texttt{NextPlay}$.

$$\texttt{NextPlay}(v) \equiv \texttt{b}_v = \bot \wedge \begin{cases} \texttt{ph}_v = 1 \wedge \forall u \in \texttt{Players}(v), \texttt{b}_u \neq \bot \\ \vee \\ \texttt{ph}_v \neq 1 \wedge \exists u \in \texttt{Players}(v) : \texttt{b}_u \neq \bot \end{cases} \tag{19}$$

Once all these conditions are fulfilled, v can announce the biggest value of $\mathtt{b}$ it sees in its playing childhood (classically we have $\bot < 0 < 1$). This action is denoted $\mathtt{MaxPos}$.

$$\mathtt{MaxPos}(v) \equiv \mathtt{b}_v := \max_{u \in \mathtt{Players}(v)} \mathtt{b}_u \tag{20}$$

$\mathbb{R}_{\mathtt{maxPos}}$: for all $v \in V$

$$\neg\mathtt{Er}(v) \wedge \mathtt{tok}_v = \bullet \wedge |\mathtt{Players}(v)| \geq 2 \wedge \mathtt{Synch}(v) \wedge \mathtt{NextPlay}(v) \longrightarrow \mathtt{MaxPos}(v)$$

Predicate for Rule $\mathbb{R}_{\mathtt{Lose}}$ In the previous section we did not explicitly give the predicate $\mathtt{LosePar}$. One node v loses the negotiation when they have the same value of $\mathtt{ph}$ as their parent, and that the answer provided by their parent is different from the one they have.

$$\mathtt{LosePar}(v, p) \equiv \mathtt{ph}_p = \mathtt{ph}_v \wedge \mathtt{b}_p \neq \bot \wedge \mathtt{b}_p \neq \mathtt{b}_v \tag{21}$$

Rule $\mathbb{R}_{\mathtt{NewPh}}$ The parent increases its phase only when the negotiation round is terminated, which means that nonetheless all its players have the same value as it has, for $\mathtt{ph}$, but also that they all have the same value as is has for $\mathtt{b}$.

Either this value is $\bot$, and therefore all players are out of bits, and the new phase should be 1, either it is not $\bot$, and therefore all the nodes asked to lose have actually lost and are now out of $\mathtt{Players}(v)$, and the new phase should be one more than the current one.

There is one particular case where this increase should not happen: when we have $\mathtt{ph} = 1$ and $\mathtt{b} = \bot$ (on both the parent and its players), since it does not correspond to a valid answer. Predicate $\mathtt{PhComplete}$ summarizes that.

$$\mathtt{PhComplete}(v) \equiv (\mathtt{ph}_v \neq 1 \vee \mathtt{b}_v \neq \bot) \wedge \forall u \in \mathtt{Players}(v), \mathtt{b}_u = \mathtt{b}_v \tag{22}$$

$$\mathtt{PhasePlus}(v) \equiv \begin{cases} \mathtt{b}_v := \bot \\ \mathtt{ph}_v := \begin{cases} 1 & \text{if } \forall u \in \mathtt{Players}(v), \mathtt{b}_u = \bot \\ \mathtt{ph}_v + 1 & \text{if } \exists u \in \mathtt{Players}(v), \mathtt{b}_u \neq \bot \end{cases} \end{cases} \tag{23}$$

$\mathbb{R}_{\mathtt{NewPh}}$: for all $v \in V$

$$\neg\mathtt{Er}(v) \wedge \mathtt{tok}_v = \bullet \wedge |\mathtt{Players}(v)| \geq 2 \wedge \mathtt{Synch}(v) \wedge \mathtt{PhComplete}(v) \longrightarrow \mathtt{PhasePlus}(v)$$

B.1.4 Operations Post-Negotiation

In this section we consider actions subsequent to the executions of $\mathbb{R}_{\mathtt{Give}}$ by the parent, and $\mathbb{R}_{\mathtt{Win}}$ by one child, which were presented in Section B.1.2. The parent has its variable $\mathtt{tok} = \downdownarrows$, the winning player has its variable $\mathtt{tok} = \bigstar$, and the other children that have not received the token yet have their variable $\mathtt{play} = \mathtt{L}$.

Rule $\mathbb{R}_{\mathtt{NewPlay}}$ The first action that the parent takes, after its child has actually received the token, is switching its variable $\mathtt{tok}$ to $\circlearrowleft$ to inform its other children that the negotiation phase is terminated, and that they are free to negotiate again. This is necessary to ensure a proper depth-first traversal, since some of them may also be children of the winning player.

This can be done as soon as v does not have any players. Since we classically suppose that v is not in error, it means that its children u of the opposite color are either at $\mathtt{tok}_u = \bigstar$, or at $\mathtt{play}_u \neq \mathtt{P}$.

$\mathbb{R}_{\texttt{NewPlay}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_u = \downdownarrows \wedge (\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{P} \vee \texttt{tok}_u = \bigstar) \longrightarrow \texttt{tok}_v := \circlearrowleft$$

Rule $\mathbb{R}_{\texttt{ReplayD}}$ When v has one parent which notifies that it can reset its value of $\texttt{play}_v$ to P, v first gets sure that it is not near one other negotiation phase. Indeed, v should not join a running negotiation phase by updating $\texttt{play}_v$ from L to P. In correct executions, v cannot be close to several negotiation phases, by unicity of the token. Yet, due to an inconsistent initialization of the network, this must be considered.

Parents that correspond to ongoing negotiation phases are those with $\texttt{tok}_p = \bullet$ and $\texttt{tok}_p = \downdownarrows$.

$$\texttt{ReplayL}(v) \equiv \forall u \in \mathcal{P}_{\texttt{neq}}(v), \texttt{tok}_u \notin \{\bullet, \downdownarrows\} \wedge \exists u \in \mathcal{P}_{\texttt{neq}}(v), \texttt{tok}_u \in \{\circlearrowleft, \circ\} \tag{24}$$

Note that we also consider situations where v has a parent p with $\texttt{tok}_p = \circ$. This corresponds to a similar situation, which we will develop in Section B.1.5

When this predicate is satisfied, v updates $\texttt{play}_v$ to P, and simultaneously resets its variable $\texttt{id}_v$ to a neutral value.

$$\texttt{Replay}(v) \equiv \begin{cases} \texttt{play}_v := \texttt{P}; \\ \texttt{id}_v := (1, \perp) \end{cases} \tag{25}$$

$\mathbb{R}_{\texttt{ReplayD}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{L} \wedge \texttt{ReplayL}(v) \longrightarrow \texttt{Replay}(v)$$

Rule $\mathbb{R}_{\texttt{Drop}}$ When all of its children have reset their variable play from L to P, node v can finally update its variable tok to $\downarrow$, which terminates the passing of the token to the child who won the negotiation.

To avoid deadlocks due to an inconsistent initial configuration, we must consider situations in which some child of v has the token ($\texttt{tok}_u = \bigstar$) but also has its variable $\texttt{play}_u$ at L. Rather than creating a new rule to make u updates its variable play, we chose to let v drop the token in such situations.

$\mathbb{R}_{\texttt{Drop}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \circlearrowleft \wedge (\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{L} \vee \texttt{tok}_u = \bigstar) \longrightarrow \texttt{tok}_v := \downarrow$$

Rule $\mathbb{R}_{\texttt{Nego}}$ Once the parent of the node v who won the negotiation has terminated the previous actions, and has set its variable tok to $\downarrow$, v can finally start negotiating with its own children, by setting $\texttt{tok}_v = \bullet$ and its variables ph and b to the initial value of the negotiating process.

$$\texttt{StartNego}(v) \equiv \begin{cases} \texttt{tok}_v & := \bullet; \\ \texttt{id}_v & := (1, \perp); \end{cases} \tag{26}$$

$\mathbb{R}_{\texttt{Nego}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bigstar \wedge (\forall p \in \mathcal{P}(v) : \texttt{tok}_u \notin \{\downdownarrows, \circlearrowleft\}) \longrightarrow \texttt{StartNego}(v)$$

Figure 10 presents the process of how a token is given to a child.

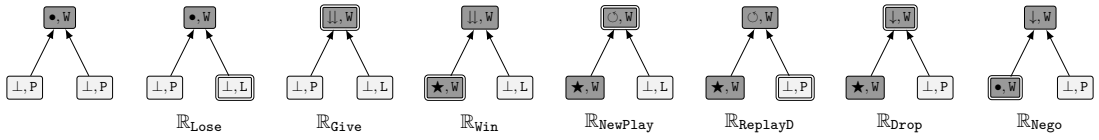

Figure 9: Execution of rules for returning the token back to one child. The activated node is the one with its line doubled

B.1.5 Reception of the Token from a Child

In this section we consider actions subsequent to the execution of $\mathbb{R}_{\texttt{OfferUp}}$ (presented in Section B.1.2). We consider one node v such that $\texttt{tok}_v = \downarrow$, which has one child u that offered the token to its parent by setting $\texttt{tok}_u = \uparrow$.

Rule $\mathbb{R}_{\texttt{Receive}}$ When node v has its child u announcing that it returns the token back, v needs to be sure that it does not have any other children which is involved in a token circulation, in which case it cannot take the token before its other children have terminated their circulation.

If v takes the token while having another child involved in a token circulation, it becomes in error, and thus executes $\mathbb{E}_{\texttt{TrustChild}}$, and sets $\texttt{tok}_v = \downarrow$, which may create a livelock.

$\mathbb{R}_{\texttt{Receive}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \downarrow \wedge (\forall u \in \mathcal{C}(v), \texttt{tok}_u \in \{\bot, \uparrow\}) \longrightarrow \texttt{tok}_v := \circ$$

Rule $\mathbb{R}_{\texttt{ReNego}}$ If node v did not have any children and took the token by setting $\texttt{tok}_v = \circ$ with rule $\mathbb{R}_{\texttt{Receive}}$, then before starting negotiating, it waits that its child, from which it received the token, actually drops it.

More precisely, v start negotiating only if it all of its children are such that $\texttt{tok}_u = \bot$, otherwise it would create an error by setting $\texttt{tok}_v = \bullet$. The following predicate guarantees that one node has no children involved in a token circulation:

$$\texttt{Leaf}(v) \equiv \forall u \in \mathcal{C}(v), \texttt{tok}_u = \bot \tag{27}$$

Furthermore, before starting a negotiation round, v also waits that all its children which have not won the token yet have properly reset their variable `play` to `L`. This is guaranteed since Rule B.1.4 applies to situations where the parent has its variable $\texttt{tok} = \circ$.

$\mathbb{R}_{\texttt{ReNego}}$: for all $v \in V$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \circ \wedge \texttt{Leaf}(v) \wedge \forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{L} \longrightarrow \texttt{StartNego}(v)$$

B.1.6 End of the Circulation

In this section we also present actions subsequent to the execution of $\mathbb{R}_{\texttt{OfferUp}}$ (presented in Section B.1.2), but from the child perspective. We consider the node which executed $\mathbb{R}_{\texttt{OfferUp}}$, and its potential children.

Rule $\mathbb{R}_{\texttt{Return}}$ After it has offered the token to its parent, a node v which is not the root should eventually drop the token, and set $\texttt{tok}_v = \bot$. Before that, it must check that all its children have correctly updated their variable `W` to `P` to assure that when a token of the other color will come, all those nodes will be available for a negotiation phase. This only applies to children of v which have the same color as v, since other children may be involved in a concurrent token circulation, and we do not want to create any livelock. Also, it only applies to nodes which are not involved in any circulation of the token, *i.e.* such that $\texttt{tok}_u = \bot$ (recall that since $\texttt{tok}_v = \uparrow$, v cannot be in error).

If such children u of v, with $\texttt{play}_u = \texttt{W}$ still have parents involved in a negotiation, they will shift that value to `F`, faking a reset to v. Thus, v should tolerate children u with $\texttt{play}_u = \texttt{F}$.

On the other hand, if v has children u which have their variable $\texttt{play}_u = \texttt{L}$, those children might be involved in another negotiation phase, which they have lost. Those children should not reset their variable `play` to `P`, for it would harm the negotiation process.

Thus, v checks that all of its children with the same color have a value of `play` different from `W` before it sends the token back.

We add one more consistency check, which is that v does not have any parent in error, *i.e.* parents p such that $\texttt{tok}_p \in \{\bigstar, \bullet, \downdownarrows, \circlearrowleft\}$. If it has, it waits for those parents to execute $\mathbb{E}_{\texttt{TrustChild}}$ before returning the token to its parents. This predicate is not necessary for our algorithm to work, but it may make convergence faster.

These predicates are combined in predicate `MayDropUp`.

$$\texttt{MayDropUp}(v) \equiv \left\{ \begin{array}{ll} & \forall p \in \mathcal{P}(v), \texttt{tok}_p \in \{\bot, \downarrow, \uparrow, \circ\} \\ \wedge & \forall u \in \mathcal{C}_{\texttt{eq}}(v), (\texttt{tok}_u = \bot \Rightarrow \texttt{play}_u \in \{\texttt{P}, \texttt{L}, \texttt{F}\}) \end{array} \right. \tag{28}$$

When this condition is fulfilled, v can drop the token. This includes setting $\texttt{tok}_v = \bot$ and updating $\texttt{play}_v$ to a correct value depending on the parents of v. As depicted in Figure 5, there are situations where nodes without a token must set their variable at `F` instead of `W`, to avoid blocking one parent which is close to execute $\mathbb{R}_{\texttt{Return}}$ itself.

$$\texttt{Drop}(v) \equiv \left\{ \begin{array}{lll} \texttt{tok}_v & := \bot; & \\ \texttt{play}_v & := \left\{ \begin{array}{ll} \texttt{W} & \text{if } \forall p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p \neq \uparrow \\ \texttt{F} & \text{if } \exists p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p = \uparrow \end{array} \right. & \end{array} \right. \tag{29}$$

$\mathbb{R}_{\texttt{Return}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \uparrow \wedge \texttt{MayDropUp}(v) \longrightarrow \texttt{Drop}(v)$$

Rule $\mathbb{R}^r_{\texttt{NewDFS}}$ In a similar situation, where all of its children have repaired themselves, the root does not drop the token, for it would simply destroy it. When the root is in such a situation, it means that the current circulation round is terminated, and one other can start.

To do that, the root reset its token value to $\bigstar$, which corresponds to the first time the token is held by any node in the new circulation round, and switches its color. After that, the root will be able to execute rule $\mathbb{R}_{\texttt{Nego}}$ and to start a negotiation process with its children.

$$\texttt{NewToken}(r) \equiv \begin{cases} \texttt{c}_r & := \neg \texttt{c}_r \\ \texttt{tok}_r & := \bigstar \end{cases} \tag{30}$$

This action can be seen as one atomic execution of the two actions `Drop` and `Win`, which corresponds to what r would have done, non-atomically, if it were not the root.

$\mathbb{R}^r_{\texttt{NewDFS}}$: for $v = r$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \uparrow \wedge \texttt{MayDropUp}(v) \longrightarrow \texttt{NewToken}(v)$$

Let us now consider the actions taken by children of such a node, *i.e.* a node v which has one parent of the same color and its variable $\texttt{tok} = \uparrow$.

Rule $\mathbb{R}_{\texttt{ReplayUp}}$ If node v only has parents that are sending the token higher, *i.e.* no other parent which is still involved in a token circulation, then v can reset its variable $\texttt{play}_v = \texttt{P}$ to be ready for the next circulation round.

$$\texttt{ReplayW}(v) \equiv \exists p \in \mathcal{P}_{\texttt{eq}}(v) : \texttt{tok}_p = \uparrow \wedge \forall p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p \in \{\bot, \uparrow\} \tag{31}$$

Note that v only consider its parent with the same color, since the situation which we want to prevent is livelock caused by the oscillation between two tokens. Therefore, v can join a negotiation of the other color by this action.

$\mathbb{R}_{\texttt{ReplayUp}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{W} \wedge \texttt{ReplayW}(v) \longrightarrow \texttt{Replay}(v)$$

Rule $\mathbb{R}_{\texttt{Fake}}$ If v has, in addition to some parents that are sending the token higher, other parents which do not have terminated their token circulation, it must update its variable $\texttt{play}_v = \texttt{F}$.

$$\texttt{FakeReplay}(v) \equiv \exists p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p = \uparrow \wedge \neg\texttt{ReplayW}(v) \tag{32}$$

$\mathbb{R}_{\texttt{Fake}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{W} \wedge \texttt{FakeReplay}(v) \longrightarrow \texttt{play}_v := \texttt{F}$$

Rule $\mathbb{R}_{\texttt{ReWin}}$ Finally, we must design a rule to cancel the effect of updating one's variable at `F`, to reset it at `W`. Without this rule, the predicate `WaitSib` could be infinitely false one one parent of v, and thus the token could not be sent higher. This rule can be activated only when the node does not have any parent with the same color such that $\texttt{tok} = \uparrow$.

Up to this rule, all the guards of the rules require that the node is near a token. For consistency, we also require proximity of a token for this rule, but it is not actually necessary.

$$\texttt{StopFaking}(v) \equiv \exists p \in \mathcal{P}_{\texttt{eq}}(v) : \texttt{tok}_v \neq \bot \wedge \forall p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p \neq \uparrow \tag{33}$$

$\mathbb{R}_{\texttt{ReWin}}$: for all $v \in V \setminus \{r\}$

$$\neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{F} \wedge \texttt{StopFaking}(v) \longrightarrow \texttt{play}_v := \texttt{W}$$

Figure 9 present the process of how a token is returned to a parent.

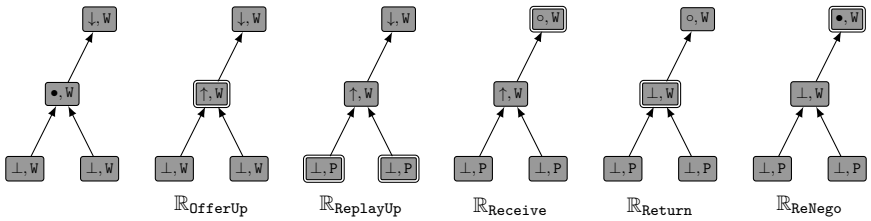

Figure 10: Execution of rules for returning the token back to one parent. The activated node is the one with its line doubled

B.1.7 Complete Algorithm

Algorithm 2 presents all the rules of our algorithm.

Algorithm 2: Token Circulation algorithm

$\forall v$	$\mathbb{E}_{\texttt{TrustChild}}$	$: \texttt{Er}(v)$	$\longrightarrow \texttt{tok}_v :=\downarrow$
$\forall v$	$\mathbb{R}_{\texttt{OfferUp}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \neg\texttt{WaitSib}(v) \wedge (\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \in \{\texttt{W}, \texttt{F}\})$	$\longrightarrow \texttt{tok}_v :=\uparrow$
$\forall v$	$\mathbb{R}_{\texttt{Give}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \texttt{Give}(v)$	$\longrightarrow \texttt{tok}_v :=\downdownarrows$
$v \neq r$	$\mathbb{R}_{\texttt{FakeWin}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \neg\texttt{OkNeg}(v)$	$\longrightarrow \texttt{FakeWin}(v)$
$v \neq r$	$\mathbb{R}_{\texttt{Win}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{WinNeg}(v)$	$\longrightarrow \texttt{Win}(v)$
$v \neq r$	$\mathbb{R}_{\texttt{Lose}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{LoseNeg}(v)$	$\longrightarrow \texttt{play}_v := \texttt{L}$
$v \neq r$	$\mathbb{R}_{\texttt{NewBit}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{AnswerNeg}(v)$	$\longrightarrow \texttt{Announce}(v)$
$\forall v$	$\mathbb{R}_{\texttt{maxPos}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \lvert\texttt{Players}(v)\rvert \geq 2 \wedge \texttt{Synch}(v) \wedge \texttt{NextPlay}(v)$	$\longrightarrow \texttt{MaxPos}(v)$
$\forall v$	$\mathbb{R}_{\texttt{NewPh}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \lvert\texttt{Players}(v)\rvert \geq 2 \wedge \texttt{Synch}(v) \wedge \texttt{PhComplete}(v)$	$\longrightarrow \texttt{PhasePlus}(v)$
$\forall v$	$\mathbb{R}_{\texttt{NewPlay}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_u =\downdownarrows \wedge(\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{P} \vee \texttt{tok}_u = \bigstar)$	$\longrightarrow \texttt{tok}_v :=\circlearrowleft$
$v \neq r$	$\mathbb{R}_{\texttt{ReplayD}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{L} \wedge \texttt{ReplayL}(v)$	$\longrightarrow \texttt{Replay}(v)$
$\forall v$	$\mathbb{R}_{\texttt{Drop}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v =\circlearrowleft \wedge(\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{L} \vee \texttt{tok}_u = \bigstar)$	$\longrightarrow \texttt{tok}_v :=\downarrow$
$\forall v$	$\mathbb{R}_{\texttt{Nego}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bigstar \wedge (\forall p \in \mathcal{P}(v) : \texttt{tok}_u \notin \{\downdownarrows, \circlearrowleft\})$	$\longrightarrow \texttt{StartNego}(v)$
$\forall v$	$\mathbb{R}_{\texttt{Receive}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v =\downarrow \wedge(\forall u \in \mathcal{C}(v), \texttt{tok}_u \in \{\perp, \uparrow\})$	$\longrightarrow \texttt{tok}_v := \circ$
$\forall v$	$\mathbb{R}_{\texttt{ReNego}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \circ \wedge \texttt{Leaf}(v) \wedge \forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{L}$	$\longrightarrow \texttt{StartNego}(v)$
$v \neq r$	$\mathbb{R}_{\texttt{Return}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v =\uparrow \wedge\texttt{MayDropUp}(v)$	$\longrightarrow \texttt{Drop}(v)$
r	$\mathbb{R}^r_{\texttt{NewDFS}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v =\uparrow \wedge\texttt{MayDropUp}(v)$	$\longrightarrow \texttt{NewToken}(v)$
$v \neq r$	$\mathbb{R}_{\texttt{ReplayUp}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{W} \wedge \texttt{ReplayW}(v)$	$\longrightarrow \texttt{Replay}(v)$
$v \neq r$	$\mathbb{R}_{\texttt{Fake}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{W} \wedge \texttt{FakeReplay}(v)$	$\longrightarrow \texttt{play}_v := \texttt{F}$
$v \neq r$	$\mathbb{R}_{\texttt{ReWin}}$	$: \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{F} \wedge \texttt{StopFaking}(v)$	$\longrightarrow \texttt{play}_v := \texttt{W}$

Figures 11 and 12 present the scheme of an execution of our algorithm. These are two dual visions of the same phenomenon. The first one focuses on the values of the different variables of one node v, and the second on the executions of the rules on one node v.

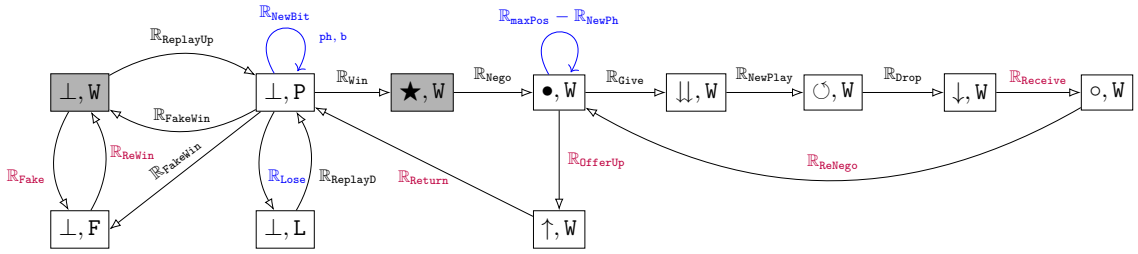

Figure 11: Chain of the states taken by a node during the execution of the algorithm. The update of c_v is represents by color gray. Rules in blue corresponds to the negotiation. Rules in red correspond to the sending of a token from a child to a parent. Rules in black correspond to the sending of a token from a parent to a child.

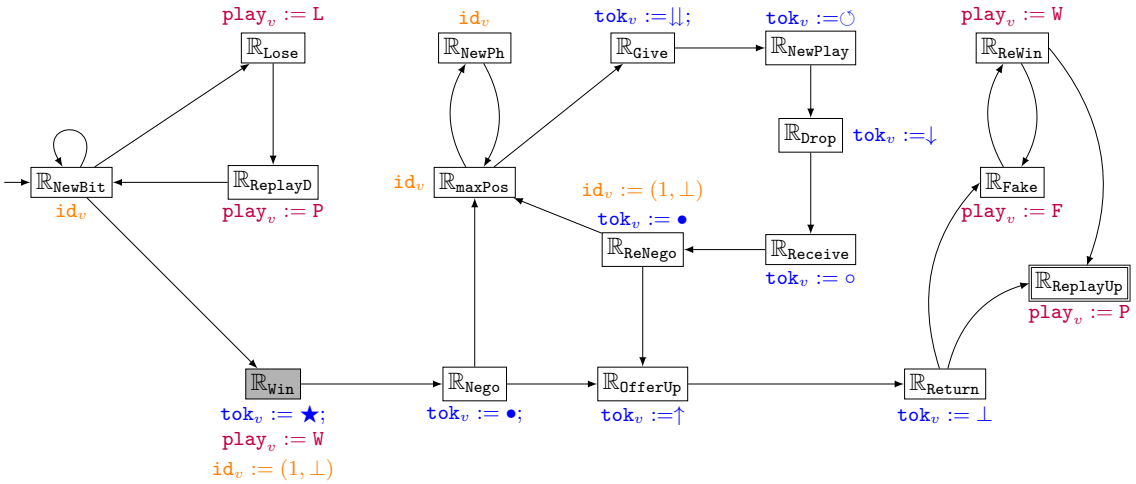

Figure 12: Chain of execution of the different rules on one node, and effects of these rules. The update of c_v is represents by color gray.

C Appendix on the Correctness of our Algorithm

In this section, we prove that Algorithm 2 is a self-stabilizing algorithm for the token circulation problem, as it is defined in Specification 1. We also prove that this algorithm is optimal in terms of memory, *i.e.* that any algorithm which solves $\mathcal{TC}$ in $\mathcal{S}$ requires $\Omega(\log\log n)$ bits per nodes, even under less general hypothesis than ours.

This section is subdivided in four subsections. In Section C.1 we establish that maximal executions of our algorithm are infinite. Formally, we prove that in any configuration, there is at least one enabled node. This work is static in the sense that we do not need to consider an execution of the algorithm. It boils down to a syntactical analysis of the guards of the rules of our algorithm (Algorithm 2).

In Section C.2 we establish that no token can circulate indefinitely in the $\mathtt{D}^o$, unless it regularly reaches the root and is reset by the execution of $\mathbb{R}^r_{\mathtt{NewDFS}}$. This guarantees that a token which is not anchored at the root of the $\mathtt{D}^o$ either vanishes, or is eventually not activated. Using the previous, we also establish that there are an infinite number of circulation rounds in maximal executions of our algorithm. This guarantees that our algorithm satisfies Condition 2 of Specification 1.

In Section C.3 we use the previous results to establish that the token anchored at the root circulates fairly in deeper and deeper parts of the $\mathtt{D}^o$. Thus, even if there are other tokens in the network, they will eventually be reached and destroyed by the legitimate token anchored at the root. Therefore we prove in this section that, at some point, the execution of our algorithm satisfies Conditions 1 and 3 of Specification 1, which means that it is a self-stabilizing algorithm for $\mathcal{TD}$.

Finally, in Section C.4 we use the results of [3] to prove that the space complexity of our algorithm is optimal.

In order to prove that our algorithm satisfies conditions of Specification 1, we must first define the predicates T and R involved in this specification. One node v is considered to hold the token if it has a non-empty value for its variable $\mathtt{tok}_v$, and if none of its children holds the token. One node v is considered to have access to the resource if it is in the state where it just received the token from its parent, which corresponds to $\mathtt{tok}_v = \bigstar$.

Definition 13 (*Predicates for our Token Circulation Algorithm*)

The resolution of $\mathcal{TD}$ by our algorithm will be proven under the following specification predicates:

$$T(v,\gamma) \equiv \mathtt{tok}_v^\gamma \neq \bot \wedge \forall u \in \mathcal{C}(v), \mathtt{tok}_u^\gamma = \bot$$
$$R(v,\gamma) \equiv T(v,\gamma) \wedge \mathtt{tok}_v^\gamma = \bigstar$$

Definition 14 (*Reset Points*)

Let $\epsilon = \gamma_0 \to \cdots$ be an execution. We say that γ_i is a reset point of ϵ if r executes $\mathbb{R}^r_{\texttt{NewDFS}}$ during $\gamma_{i-1} \to \gamma_{i+1}$. We say that two reset points γ_i and γ_j are consecutive if $i < j$ and $\forall k \in [i+1, j+1]$, γ_k is not a reset point.

Theorem 1 (*Circulation Rounds*)

Let $\epsilon = \gamma_0 \to \cdots$ be an execution. Let us denote by $t_0 \leq t_1 \leq \cdots$ all the reset points of ϵ, such that $\forall i \geq 0, \gamma_{t_i}$ and $\gamma_{t_{i+1}}$ are consecutive.

$\forall i \geq 0, \gamma_{t_1} \to \cdots \to \gamma_{t_{i+1}-1}$ is a circulation round.

Proof: For any computing step $cs = \gamma \to \gamma'$, we have $\neg R^{\gamma}(r) \wedge R^{\gamma'}(r)$ if and only if r executes $\mathbb{R}^r_{\texttt{NewDFS}}$, which is equivalent to γ' being a reset point. ■

C.1 Liveness

Recall that we denote by $\mathcal{A}^e(\gamma)$ the set of the enabled nodes for $\mathcal{A}$ in configuration γ. In this section, we prove that $\forall \gamma \ \in \Gamma, \mathcal{A}^e(\gamma) \neq \emptyset$.

The proof is subdivided in several lemmas. In each lemma we work under the hypothesis that some value for $\texttt{tok}_v$, on one node v, is present in γ, and in each case we prove that there exists at least one enabled node, in γ. Not all the lemmas are independent: it happens that to prove a lemma, we simply prove that there exists one node u with another value of $\texttt{tok}_u$, value which has already been dealt in a previous lemma.

One lemma is even proven by induction, the lemma which considers as a hypothesis the presence of a node v such that $\texttt{tok}_v = \downarrow$. Such nodes can actually pretty far from the actual token, and thus to find an enabled node, we may have to descend a branch of the $\texttt{D}^o$, until we fall on a leaf, or on a node with a different value for $\texttt{tok}_v$.

More precisely, we prove that either we can find an enabled node, either there exists one other node u such that $\texttt{tok}_u = \downarrow$, but u is deeper than v in the DODAG. Since DODAGs are finite, and acyclic, there can only be a finite number of times where the second hypothesis is the right one, and therefore this proves that there exists at least one enabled node.

The proofs are basically reasoning by case disjunction, which forms decision trees of variable depth. We must guarantee that at the end of each branch *i.e.* at each leaf, we find one enabled node.

Since it is basically syntactical analysis, we only present the different decision trees, which contain all the elements of the proofs, and are easier to parse than the written version of the proofs.

In each proof, we also provide a reasoning toolbox, to remind the reader the detail of the predicates considered in the reasoning.

Lemma 1

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\texttt{tok}_v = \uparrow$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figure 13 ■

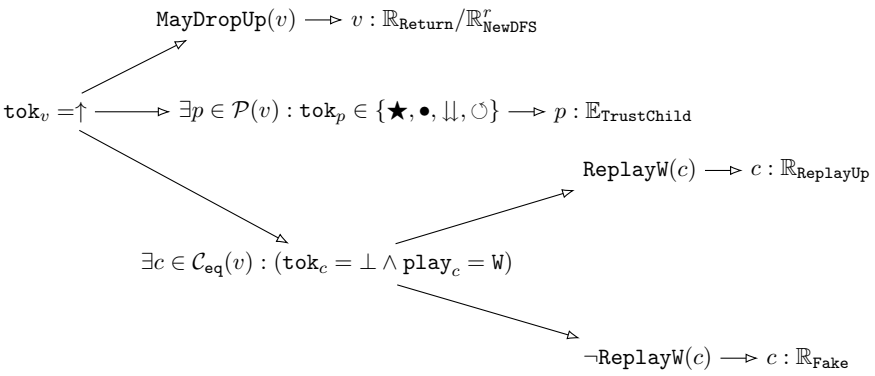

Figure 13: Proof of Lemma 1

Tools Lemma 1

$$
\begin{array}{llll}
\forall v & \mathbb{E}_{\texttt{TrustChild}} : \texttt{Er}(v) & & \longrightarrow \texttt{tok}_v := \downarrow \\
v \neq r & \mathbb{R}_{\texttt{Return}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \uparrow \wedge \texttt{MayDropUp}(v) & & \longrightarrow \texttt{Drop}(v) \\
r & \mathbb{R}^r_{\texttt{NewDFS}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \uparrow \wedge \texttt{MayDropUp}(v) & & \longrightarrow \texttt{NewToken}(v) \\
v \neq r & \mathbb{R}_{\texttt{ReplayUp}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{W} \wedge \texttt{ReplayW}(v) & & \longrightarrow \texttt{Replay}(v) \\
v \neq r & \mathbb{R}_{\texttt{Fake}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bot \wedge \texttt{play}_v = \texttt{W} \wedge \texttt{FakeReplay}(v) & & \longrightarrow \texttt{play}_v := \texttt{F}
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{Er}(v) & \equiv & \big(\texttt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \neq \bot\big) \\
& \vee & \big(\texttt{tok}_v \in \{\Downarrow, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \bigstar\}\big) \\
& \vee & \big(\texttt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \uparrow\}\big) \\
& \longrightarrow & \texttt{tok}_v := \downarrow
\end{array}
$$

$$
\texttt{MayDropUp}(v) \equiv \left\{ \begin{array}{ll} & \forall p \in \mathcal{P}(v), \texttt{tok}_p \in \{\bot, \downarrow, \uparrow, \circ\} \\ \wedge & \forall u \in \mathcal{C}_{\texttt{eq}}(v), (\texttt{tok}_u = \bot \Rightarrow \texttt{play}_u \in \{\texttt{P}, \texttt{L}, \texttt{F}\}) \end{array} \right.
$$

$$
\texttt{ReplayW}(v) \equiv \exists p \in \mathcal{P}_{\texttt{eq}}(v) : \texttt{tok}_p = \uparrow \wedge \forall p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p \in \{\bot, \uparrow\}
$$

$$
\texttt{FakeReplay}(v) \equiv \exists p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p = \uparrow \wedge \neg\texttt{ReplayW}(v)
$$

Lemma 2

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\mathtt{tok}_v = \downdownarrows$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figure 14 ■

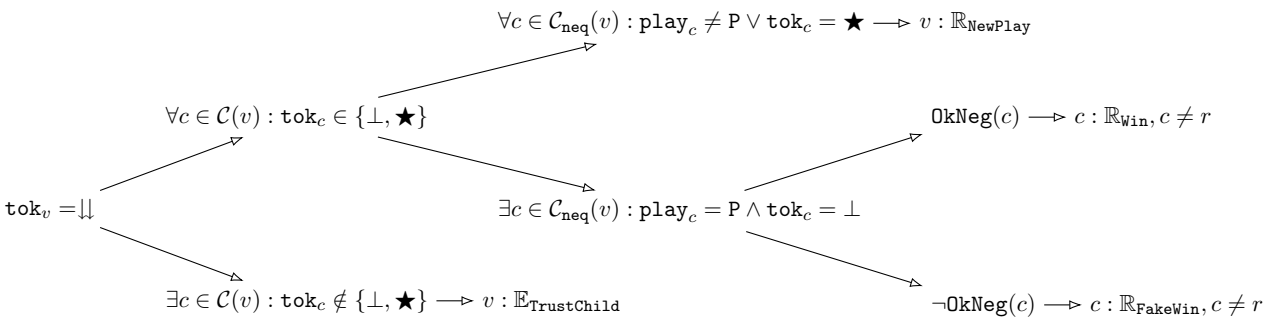

Figure 14: Proof of Lemma 2

Tools Lemma 2

$$
\begin{array}{lllll}
\forall v & \mathbb{E}_{\mathtt{TrustChild}} & : \mathtt{Er}(v) & & \longrightarrow \mathtt{tok}_v := \downarrow \\
\forall v & \mathbb{R}_{\mathtt{NewPlay}} & : \neg\mathtt{Er}(v) \wedge \mathtt{tok}_u = \downdownarrows \wedge (\forall u \in \mathcal{C}_{\mathtt{neq}}(v) : \mathtt{play}_u \neq \mathtt{P} \vee \mathtt{tok}_u = \bigstar) & & \longrightarrow \mathtt{tok}_v := \circlearrowleft \\
\forall v \neq r & \mathbb{R}_{\mathtt{Win}} & : \neg\mathtt{Er}(v) \wedge \mathtt{tok}_v = \bot \wedge \mathtt{play}_v = \mathtt{P} \wedge \mathtt{WinNeg}(v) & & \longrightarrow \mathtt{Win}(v) \\
\forall v \neq r & \mathbb{R}_{\mathtt{FakeWin}} & : \neg\mathtt{Er}(v) \wedge \mathtt{tok}_v = \bot \wedge \mathtt{play}_v = \mathtt{P} \wedge \neg\mathtt{OkNeg}(v) & & \longrightarrow \mathtt{FakeWin}(v)
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{Er}(v) & \equiv & \big(\mathtt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \mathtt{tok}_u \neq \bot\big) \\
& \vee & \big(\mathtt{tok}_v \in \{\downdownarrows, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \mathtt{tok}_u \notin \{\bot, \bigstar\}\big) \\
& \vee & \big(\mathtt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \mathtt{tok}_u \notin \{\bot, \uparrow\}\big) \\
& \longrightarrow & \mathtt{tok}_v := \downarrow
\end{array}
$$

$$\mathtt{WinNeg}(v) \equiv \mathtt{OkNeg}(v) \wedge \exists p \in \mathcal{P}_{\mathtt{neq}}(v) : \mathtt{tok}_p = \downdownarrows$$

$$\mathtt{OkNeg}(v) \equiv |\mathtt{ParNeg}(v)| \leq 1 \vee (|\mathtt{ParNeg}(v)| = 2 \wedge \{\mathtt{tok}_p \mid p \in \mathtt{ParNeg}(v)\} \in \{\{\downdownarrows, \bigstar\}, \{\circlearrowleft, \bigstar\}\})$$

$$\mathtt{ParNeg}(v) = \{p \in \mathcal{P}(v) \mid \mathtt{tok}_p \notin \{\bot, \uparrow, \downarrow\}\}$$

Lemma 3

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\texttt{tok}_v = \bullet$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figures 15 and 16 ■

Tools Lemma 3

$$
\begin{array}{llll}
\forall v & \mathbb{E}_{\texttt{TrustChild}} : \texttt{Er}(v) & & \longrightarrow \texttt{tok}_v := \downarrow \\
\forall v & \mathbb{R}_{\texttt{maxPos}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge |\texttt{Players}(v)| \geq 2 \wedge \texttt{Synch}(v) \wedge \texttt{NextPlay}(v) & & \longrightarrow \texttt{MaxPos}(v) \\
\forall v & \mathbb{R}_{\texttt{NewPh}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge |\texttt{Players}(v)| \geq 2 \wedge \texttt{Synch}(v) \wedge \texttt{PhComplete}(v) & & \longrightarrow \texttt{PhasePlus}(v) \\
\forall v & \mathbb{R}_{\texttt{Give}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \texttt{Give}(v) & & \longrightarrow \texttt{tok}_v := \downdownarrows \\
\forall v & \mathbb{R}_{\texttt{OfferUp}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bullet \wedge \neg\texttt{WaitSib}(v) \wedge (\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \in \{\texttt{W}, \texttt{F}\}) & & \longrightarrow \texttt{tok}_v := \uparrow \\
v \neq r & \mathbb{R}_{\texttt{NewBit}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{AnswerNeg}(v) & & \longrightarrow \texttt{Announce}(v) \\
v \neq r & \mathbb{R}_{\texttt{Lose}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{LoseNeg}(v) & & \longrightarrow \texttt{play}_v := \texttt{L} \\
v \neq r & \mathbb{R}_{\texttt{Win}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \texttt{WinNeg}(v) & & \longrightarrow \texttt{Win}(v) \\
v \neq r & \mathbb{R}_{\texttt{FakeWin}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{P} \wedge \neg\texttt{OkNeg}(v) & & \longrightarrow \texttt{FakeWin}(v) \\
v \neq r & \mathbb{R}_{\texttt{ReWin}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{F} \wedge \texttt{StopFaking}(v) & & \longrightarrow \texttt{play}_v := \texttt{W}
\end{array}
$$

$$
\begin{array}{rll}
\texttt{Er}(v) \quad \equiv & & \big(\texttt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \neq \perp\big) \\
& \vee & \big(\texttt{tok}_v \in \{\downdownarrows, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\perp, \bigstar\}\big) \\
& \vee & \big(\texttt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\perp, \uparrow\}\big) \\
& \longrightarrow & \texttt{tok}_v := \downarrow
\end{array}
$$

$$\texttt{AnswerPar}(v, p) \equiv (\texttt{ph}_v \neq \texttt{ph}_p) \vee (\texttt{ph}_p = 1 \wedge \texttt{b}_p = \perp \wedge \texttt{b}_v = \perp)$$

$$\texttt{LosePar}(v, p) \equiv \texttt{ph}_p = \texttt{ph}_v \wedge \texttt{b}_p \neq \perp \wedge \texttt{b}_p \neq \texttt{b}_v$$

$$\texttt{ParNeg}(v) = \{p \in \mathcal{P}(v) \mid \texttt{tok}_p \notin \{\perp, \uparrow, \downarrow\}\}$$

$$\texttt{OkNeg}(v) \equiv |\texttt{ParNeg}(v)| \leq 1 \vee (|\texttt{ParNeg}(v)| = 2 \wedge \{\texttt{tok}_p \mid p \in \texttt{ParNeg}(v)\} \in \{\{\downdownarrows, \bigstar\}, \{\circlearrowleft, \bigstar\}\})$$

$$\texttt{WinNeg}(v) \equiv \texttt{OkNeg}(v) \wedge \exists p \in \mathcal{P}_{\texttt{neq}}(v) : \texttt{tok}_p = \downdownarrows$$

$$\texttt{LoseNeg}(v) \equiv \texttt{OkNeg}(v) \wedge \neg\texttt{WinNeg}(v) \wedge \exists p \in \mathcal{P}_{\texttt{neq}}(v) : \texttt{tok}_p = \bullet \wedge \texttt{LosePar}(v, p)$$

$$\texttt{AnswerNeg}(v) \equiv \texttt{OkNeg}(v) \wedge \neg\texttt{WinNeg}(v) \wedge \neg\texttt{LoseNeg}(v) \wedge \exists p \in \mathcal{P}_{\texttt{neq}}(v) : \texttt{tok}_p = \bullet \wedge \texttt{AnswerPar}(v, p)$$

$$\texttt{Players}(v) = \{u \in \mathcal{C}_{\texttt{neq}}(v) \mid \texttt{tok}_v = \perp \wedge \texttt{play}_u = \texttt{P}\}$$

$$\texttt{Synch}(v) \equiv \forall u \in \texttt{Players}(v), \texttt{ph}_u = \texttt{ph}_v$$

$$\texttt{NextPlay}(v) \equiv \texttt{b}_v = \perp \wedge \begin{cases} \texttt{ph}_v = 1 \wedge \forall u \in \texttt{Players}(v), \texttt{b}_u \neq \perp \\ \vee \\ \texttt{ph}_v \neq 1 \wedge \exists u \in \texttt{Players}(v) : \texttt{b}_u \neq \perp \end{cases}$$

$$\texttt{PhComplete}(v) \equiv (\texttt{ph}_v \neq 1 \vee \texttt{b}_v \neq \perp) \wedge \forall u \in \texttt{Players}(v), \texttt{b}_u = \texttt{b}_v$$

$$\texttt{Give}(v) \equiv |\texttt{Players}(v)| = 1 \vee (|\texttt{Players}(v)| = 0 \wedge \exists u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u = \texttt{L})$$

$$\texttt{WaitSib}(v) \equiv \exists c \in \mathcal{C}_{\texttt{eq}}(v) : \texttt{play}_c = \texttt{F}$$

$$\texttt{StopFaking}(v) \equiv \exists p \in \mathcal{P}_{\texttt{eq}}(v) : \texttt{tok}_v \neq \perp \wedge \forall p \in \mathcal{P}_{\texttt{eq}}(v), \texttt{tok}_p \neq \uparrow$$

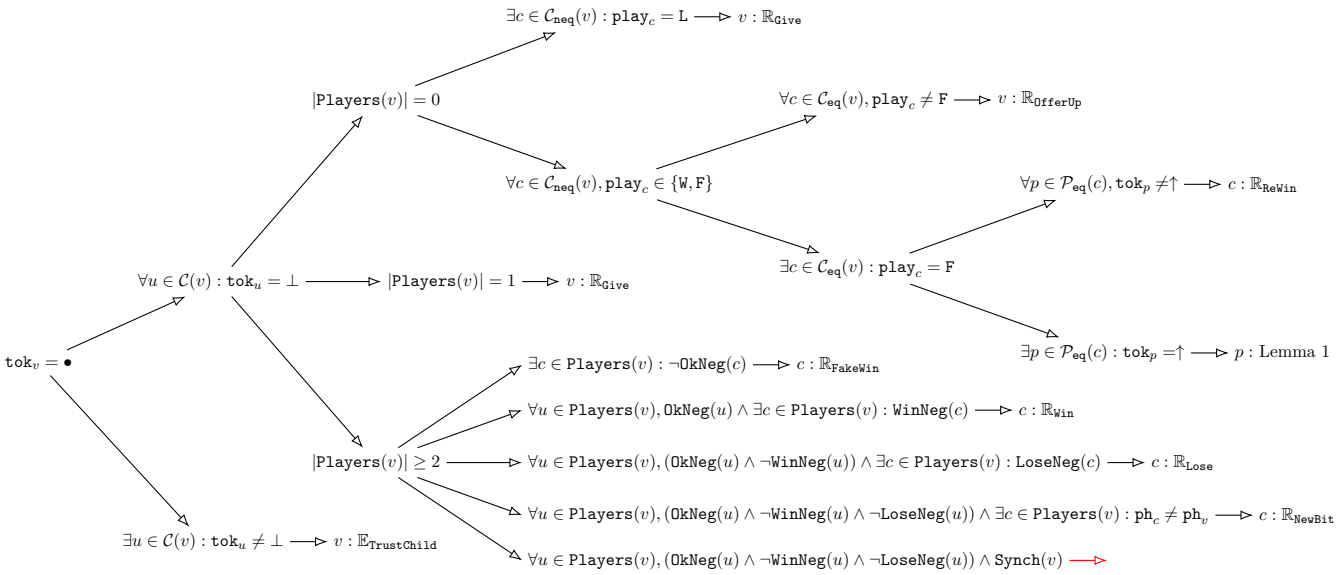

Figure 15: Proof of Lemma 3, part. 1

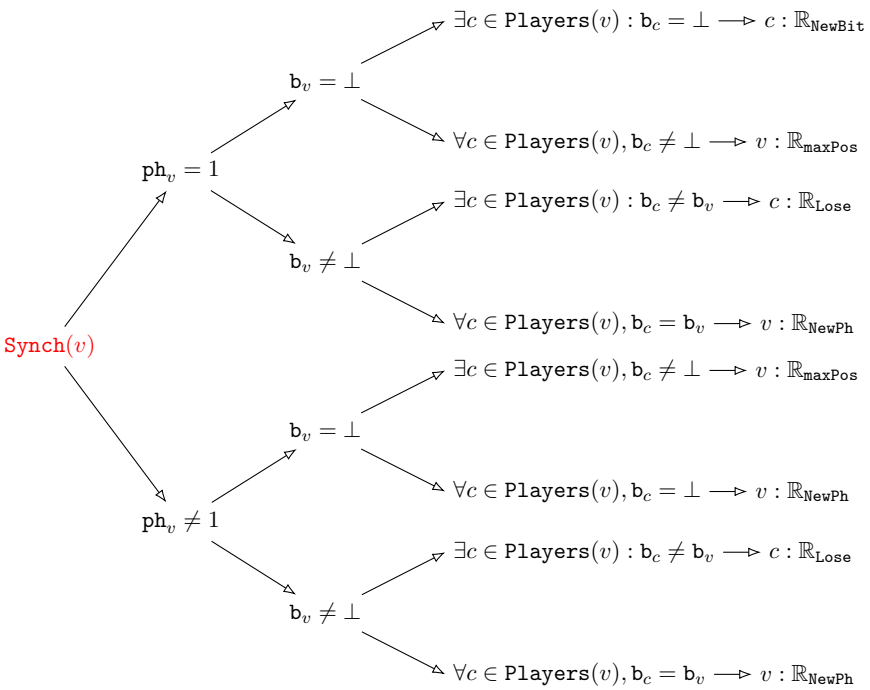

Figure 16: Proof of Lemma 3, part. 2

Lemma 4

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\texttt{tok}_v = \circlearrowleft$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figure 17 ■

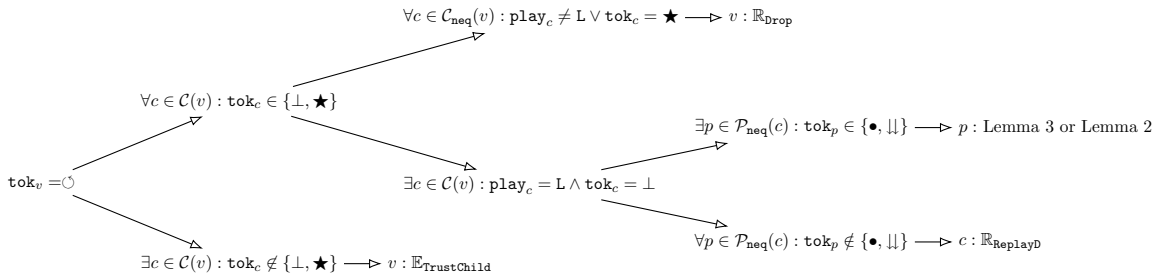

Figure 17: Proof of Lemma 4

Tools Lemma 4

$$
\begin{array}{lll}
\forall v & \mathbb{E}_{\texttt{TrustChild}} : \texttt{Er}(v) & \longrightarrow \texttt{tok}_v :=\downarrow \\
\forall v & \mathbb{R}_{\texttt{Drop}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \circlearrowleft \wedge (\forall u \in \mathcal{C}_{\texttt{neq}}(v) : \texttt{play}_u \neq \texttt{L} \vee \texttt{tok}_u = \bigstar) & \longrightarrow \texttt{tok}_v :=\downarrow \\
\forall v \neq r & \mathbb{R}_{\texttt{ReplayD}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \perp \wedge \texttt{play}_v = \texttt{L} \wedge \texttt{ReplayL}(v) & \longrightarrow \texttt{Replay}(v)
\end{array}
$$

$$
\begin{array}{rcll}
\texttt{Er}(v) & \equiv & \big(\texttt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \neq \perp\big) \\
& \vee & \big(\texttt{tok}_v \in \{\downdownarrows, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\perp, \bigstar\}\big) \\
& \vee & \big(\texttt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\perp, \uparrow\}\big) \\
& \longrightarrow & \texttt{tok}_v :=\downarrow
\end{array}
$$

$$\texttt{ReplayL}(v) \equiv \forall u \in \mathcal{P}_{\texttt{neq}}(v), \texttt{tok}_u \notin \{\bullet, \downdownarrows\} \wedge \exists u \in \mathcal{P}_{\texttt{neq}}(v), \texttt{tok}_u \in \{\circlearrowleft, \circ\}$$

Lemma 5

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\mathtt{tok}_v = \circ$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figure 18 ■

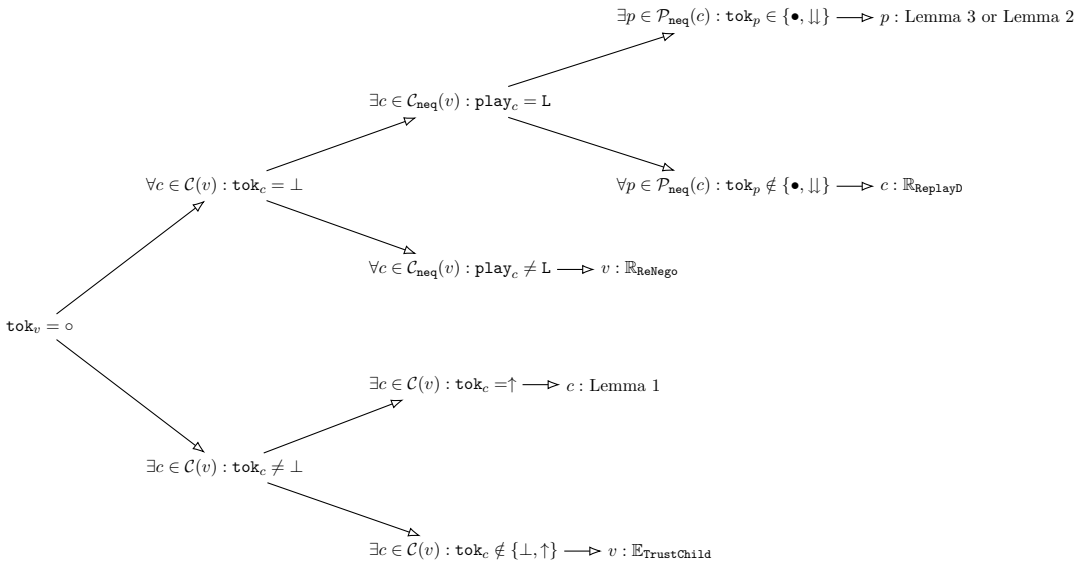

Figure 18: Proof of Lemma 5

Tools Lemma 5

$$
\begin{array}{llll}
\forall v & \mathbb{E}_{\mathtt{TrustChild}} : \mathtt{Er}(v) & & \longrightarrow \mathtt{tok}_v := \downarrow \\
\forall v & \mathbb{R}_{\mathtt{ReNego}} & : \neg\mathtt{Er}(v) \wedge \mathtt{tok}_v = \circ \wedge \mathtt{Leaf}(v) \wedge \forall u \in \mathcal{C}_{\mathtt{neq}}(v) : \mathtt{play}_u \neq \mathtt{L} & \longrightarrow \mathtt{StartNego}(v) \\
v \neq r & \mathbb{R}_{\mathtt{ReplayD}} & : \neg\mathtt{Er}(v) \wedge \mathtt{tok}_v = \bot \wedge \mathtt{play}_v = \mathtt{L} \wedge \mathtt{ReplayL}(v) & \longrightarrow \mathtt{play}_v := \mathtt{P}
\end{array}
$$

$$
\begin{array}{rcll}
\mathtt{Er}(v) & \equiv & \big(\mathtt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \mathtt{tok}_u \neq \bot\big) \\
& \vee & \big(\mathtt{tok}_v \in \{\downdownarrows, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \mathtt{tok}_u \notin \{\bot, \bigstar\}\big) \\
& \vee & \big(\mathtt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \mathtt{tok}_u \notin \{\bot, \uparrow\}\big) \\
& \longrightarrow & \mathtt{tok}_v := \downarrow
\end{array}
$$

$$\mathtt{ReplayL}(v) \equiv \forall u \in \mathcal{P}_{\mathtt{neq}}(v), \mathtt{tok}_u \notin \{\bullet, \downdownarrows\} \wedge \exists u \in \mathcal{P}_{\mathtt{neq}}(v), \mathtt{tok}_u \in \{\circlearrowleft, \circ\}$$

Lemma 6

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\texttt{tok}_v = \bigstar$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figure 19 ■

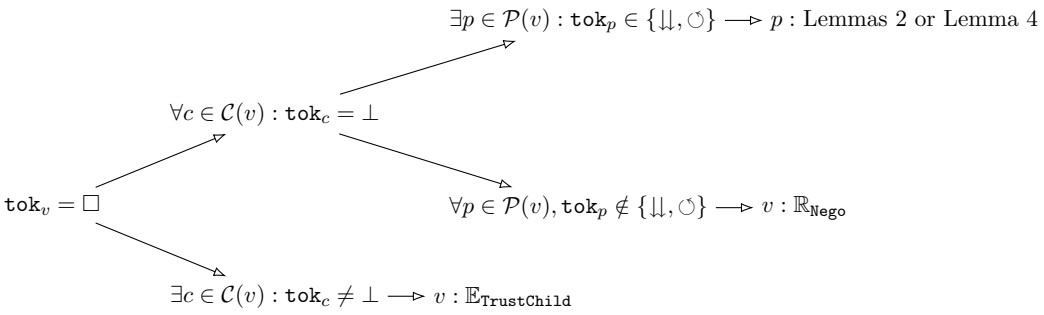

Figure 19: Proof of Lemma 6

Tools Lemma 6

$$\begin{array}{lll} \forall v & \mathbb{E}_{\texttt{TrustChild}} : \texttt{Er}(v) & \longrightarrow \texttt{tok}_v :=\downarrow \\ v \neq r & \mathbb{R}_{\texttt{Nego}} \quad : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \bigstar \wedge (\forall p \in \mathcal{P}(v) : \texttt{tok}_u \notin \{\Downarrow, \circlearrowleft\}) & \longrightarrow \texttt{StartNego}(v) \end{array}$$

$$\begin{array}{rcll} \texttt{Er}(v) & \equiv & \big(\texttt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \neq \bot\big) \\ & \vee & \big(\texttt{tok}_v \in \{\Downarrow, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \bigstar\}\big) \\ & \vee & \big(\texttt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \uparrow\}\big) \\ & \longrightarrow & \texttt{tok}_v :=\downarrow \end{array}$$

Lemma 7

Let $\gamma \in \Gamma$. If there exists $v \in V$ such that $\texttt{tok}_v = \downarrow$ in γ then $\mathcal{A}^e(\gamma) \neq \emptyset$.

Proof: The proof is given in Figure 20 ■

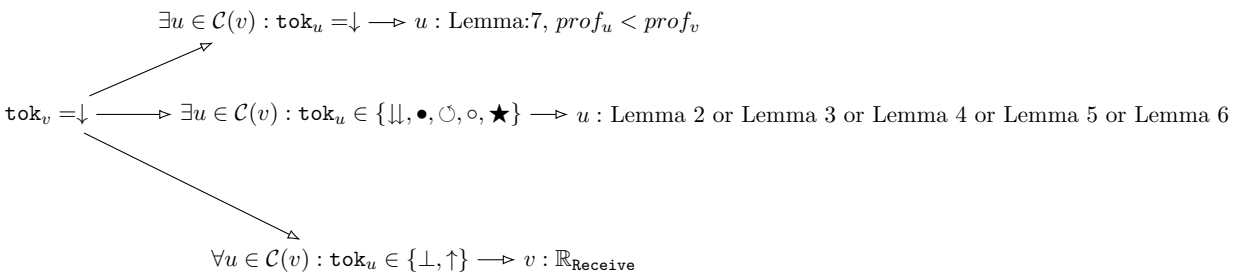

Figure 20: Proof of Lemma 7

Tools Lemma 7

$$\frac{\begin{array}{lll} \forall v & \mathbb{E}_{\texttt{TrustChild}} : \texttt{Er}(v) & \longrightarrow \texttt{tok}_v := \downarrow \\ \forall v & \mathbb{R}_{\texttt{Receive}} : \neg\texttt{Er}(v) \wedge \texttt{tok}_v = \downarrow \wedge (\forall u \in \mathcal{C}(v), \texttt{tok}_u \in \{\bot, \uparrow\}) & \longrightarrow \texttt{tok}_v := \circ \end{array}}{\begin{array}{rcll} \texttt{Er}(v) & \equiv & \big(\texttt{tok}_v \in \{\bullet, \bigstar\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \neq \bot\big) \\ & \vee & \big(\texttt{tok}_v \in \{\downdownarrows, \circlearrowleft\} \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \bigstar\}\big) \\ & \vee & \big(\texttt{tok}_v = \circ \wedge \exists u \in \mathcal{C}(v) : \texttt{tok}_u \notin \{\bot, \uparrow\}\big) \\ & \longrightarrow & \texttt{tok}_v := \downarrow \end{array}}$$

Theorem 2

$\forall \gamma \in \Gamma, \mathcal{A}^e(\gamma) \neq \emptyset$

Proof: Let $\gamma \in \Gamma$ be a configuration. Since $\texttt{tok}_r \in \{\uparrow, \downdownarrows, \bullet, \circlearrowleft, \circ, \bigstar, \downarrow\}$, one of the following lemmas applies: 1, 2, 3, 4, 5, 6, 7. ■

C.2 Progress

In this section we formally establish, at once, two crucial properties for the validity of our algorithm. The first property is that if a token is not anchored at the root of the $\texttt{D}^o$ (we will give later a proper definition of the anchor of a token), then it can only circulate a finite number of times before being blocked, or deleted. This means that no scheduler, even unfair, can create an execution in which the token held by the root is never activated.

The second property is that circulation rounds (see Section A.4), are always finite. We actually prove that the token held by the root can only circulate a finite number of times before it reaches the root, and the root executes $\mathbb{R}^r_{\texttt{NewDFS}}$.

The combination of these two properties, and of the liveness established in Theorem 2 leads to the guarantee that there is an infinity of finite circulation round in any maximal execution of our algorithm. We will prove in the next section that, eventually, the circulation of the token is fair.

To prove both properties, we define a potential function, which associates a positive value (a *weight*) to each token in any configuration. Then, we establish that any computing step decreases the weight of the token, unless the token is reset by an execution of $\mathbb{R}^r_{\texttt{NewDFS}}$. This establishes that a token cannot circulate indefinitely, since any step it takes makes a positive quantity decrease, unless this token is anchored at the root and there are an infinity of new circulation rounds.

In this section we provide a formal proof of these properties. Let us have first an overview of this proof. In a corrupted initial configuration, the network may contain several tokens, which may have common ancestors, and also have several ancestors. To deal with the diversity of possible configurations, we define a new object to reason on, circulation DODAGs, denoted by $\texttt{CD}^o$, for each token. Then, we define a weight function, which associates a weighted DODAG, denoted by $\texttt{WD}^o$, to each $\texttt{CD}^o$. We finally prove that each time a node of a $\texttt{CD}^o$ is activated, the corresponding $\texttt{WD}^o$ decreases. According to Theorem 2, there are at least one enabled node in each configuration, which guarantees that $\texttt{WD}^o$'s actually decrease in maximal executions.

In Section C.2.1 we present the inherent difficulties to overcome for proving our results, and show how $\texttt{CD}^o$'s and $\texttt{WD}^o$'s are suitable solutions. In Section C.2.2 we formally define $\texttt{CD}^o$'s and establish basic properties on them. In Section C.2.3 we show how to define the weight function we use, and introduce the concept of $\texttt{WD}^o$'s. Then, in Sections C.2.4, C.2.5, C.2.6, C.2.7, and C.2.8 we define the actual weight function, step by step, and prove properties on it at each step. Finally, we establish our main theorems in Section C.2.9.

C.2.1 Difficulties to Overcome, Circulation DODAG

To associate a decreasing weight to a token, the main difficulty to overcome is that during one circulation, the token shifts a lot, upward and downward alternately. Thus, the information of whether the token is very close to the end of the circulation round, or still far, is not local. This information strongly depends, for example, on which branches have already been visited by the token, on the number of unvisited children the different nodes have, on whether one error will be raised, *etc.* As a result, we cannot focus only on the token if we want to succeed in estimating how many steps it still has to execute before it ends its circulation.

One first condition for our potential function to be consistent is that it takes into account the variables of the node holding the token, but also information relative to some ancestors of that node. In practice, only the ancestors which are somehow pointing to this token will be considered.

Yet, it might happen that one node in the ancestry of the token holder has several children involved in a token circulation. If this is due to a triangle topology, *i.e.* if one node has two parents, one being the parent of the other one, then considering the ascendants of the token is sufficient. But this could also be a situation in which one node has two independent token circulations below it. Our algorithm does not create such situations, but they might exist due to an incorrect initial configuration. In such a situation, our previous definition is not fully satisfactory since it does not highlight the fact that those two token circulation are, though independent, intertwined.

To embrace that complexity, we define an *ad hoc* object, called a Circulation DODAG, denoted $\texttt{CD}^o$, which has a $\texttt{D}^o$ structure. That new object follows the same idea as the upwards branch starting at the token, but we reverse that idea. We first look for nodes with no ancestry (called *anchor*), that is to say nodes v such that $\texttt{tok}_v \neq \bot \wedge \forall p \in \mathcal{P}(v), \texttt{tok}_p \in \{\bot, \uparrow\}$. Then, we go down from each anchor to all its descendants, following nodes u such that $\texttt{tok}_u \neq \bot$. Since nodes with $\texttt{tok}_u = \uparrow$ can only be updated by $\mathbb{R}_{\texttt{Return}}$, with effect $\texttt{tok}_u := \bot$, we include them in our structures, but not their potential descendants: such nodes are necessarily leaves

of the $\texttt{CD}^o$. Note that if the initial configuration is inconsistent, it may contain several $\texttt{CD}^o$'s. In Section C.2.2 we formally define $\texttt{CD}^o$'s and we establish some basic properties on $\texttt{CD}^o$'s.

Our goal is now to associate a finite positive quantity, a weight, to any $\texttt{CD}^o$ in any configuration. Such an association will be called a weight function. A weight function is said admissible if any activation of the $\texttt{CD}^o$ results in a decrease of the weight of the $\texttt{CD}^o$. Since there does not exist infinite decreasing sequence of positive integers, it comes that any $\texttt{CD}^o$ has a finite lifetime. Actually, since our algorithm of token circulation does not terminate, there necessarily exists one rule that does not respect that specification. The rule $\mathbb{R}^r_{\texttt{NewDFS}}$ is the only rule that increases the weight of a $\texttt{CD}^o$, as it refreshes the token by switching its color. Since only the root of the $\texttt{D}^o$ can execute $\mathbb{R}^r_{\texttt{NewDFS}}$, we guarantee that any other $\texttt{CD}^o$, anchored at another node than the root, has a finite lifetime.

Unfortunately, associating a simple integer to a $\texttt{CD}^o$ does not allow to take into account the diversity of situations that we must consider. The appropriate quantity to reason on has itself a $\texttt{D}^o$ structure, which has the same topology as the $\texttt{CD}^o$ to which it is associate by the weight function. Such $\texttt{D}^o$'s are called weighted DODAGs and are denoted $\texttt{WD}^o$.

More precisely, a $\texttt{WD}^o$ is a 5-tuple of $\texttt{D}^o$'s. Each component of the tuple deals with one aspect of the algorithm. The order in which are set the different components is crucial. Indeed, it happens that some computing step makes one of the component increase. We prove that this can be only if one component with higher priority decreases at the same time.

- The first component of the 5-tuple evaluates the number of unvisited children of each node of the $\texttt{CD}^o$.
- The second component the numbers of errors in the network.
- The third component is dedicated to the updates of variable `tok` on each node, as depicted in Figure 3.
- The fourth component to the variable `play` on the children of each node of the $\texttt{CD}^o$.
- And the last component is dedicated to the negotiation process near nodes with $\texttt{tok} = \bullet$.

C.2.2 Circulation DODAGs

In this subsection, we define tools to capture the behavior of the token circulation, and especially Circulation DODAGs, $\texttt{CD}^o$'s. We prove in particular that the number of $\texttt{CD}^o$'s can never increase, which is a very desirable property to have to establish that the token is eventually unique.

Remember that we denote by $\texttt{var}_v^\gamma$ the state of variable `var` for the node v in configuration γ, and by $P^\gamma(v)$ the values of predicate P on node v in configuration γ.

We first define the anchors of our $\texttt{CD}^o$'s. The anchors are nodes v that are involved in a token circulation, *i.e.* $\texttt{tok}_v \neq \bot$, and which do not have any parent in this token circulation. Recall that nodes u such that $\texttt{tok}_u = \uparrow$ cannot be considered as parents, since they can only execute rule $\mathbb{R}_{\texttt{Return}}$, which makes them quit the circulation.

Definition 15 (*Anchor*)

Let $\gamma \in \Gamma$ be a configuration. Node $a \in V$ is an anchor in γ if

- $\texttt{tok}_a^\gamma \neq \bot$, *and*
- $\forall u \in \mathcal{P}(a), \texttt{tok}_u^\gamma \in \{\bot, \uparrow\}$.

We denote by $\mathbb{A}(\gamma)$ the set of all anchors at γ, and by $\mathbb{A}^*(\gamma)$ the set of all anchors except the root.

Remark 4

The root of G is an anchor in any configuration γ.

Now that we have defined anchors, we can define the nodes of the $\texttt{CD}^o$ anchored at one particular anchor. The nodes of the $\texttt{CD}^o$ anchored at a are all the nodes which can reach a by paths corresponding to a actual token circulation. In other words, we only consider paths in which all nodes have a value $\texttt{tok} \neq \bot$, and such that all the nodes apart from the source have a value $\texttt{tok} \neq \uparrow$. Indeed, nodes u such that $\texttt{tok}_u = \uparrow$ are necessarily leaves of $\texttt{CD}^o$'s.

Definition 16 (*Nodes of a Circulation DODAG*)

Let us consider a configuration $\gamma \in \Gamma$ and an anchor $a \in \mathbb{A}(\gamma)$. We define by induction the nodes of the Circulation DODAG ($\texttt{CD}^o$) anchored at a in γ, and denote by V_a^γ as the smallest set which contains the node a and such that:

$$v \in V_a^\gamma \iff \begin{cases} \texttt{tok}_v^\gamma \neq \bot \\ \exists u \in \mathcal{P}(v) : \big(u \in V_a^\gamma \wedge \texttt{tok}_u^\gamma \neq \uparrow\big). \end{cases}$$

We now give a definition of $\texttt{CD}^o$ and we present an example with several $\texttt{CD}^o$'s in one $\texttt{D}^o$ in Figure 21.

Definition 17 (*Circulation DODAG*)

Let us consider a configuration $\gamma \in \Gamma$ and an anchor $a \in \mathbb{A}(\gamma)$. We define the Circulation DODAG ($\texttt{CD}^o$) anchored at a in γ, and denote D_a^γ, as the $\texttt{D}^o$ with nodes V_a^γ, and whose parental relationship is defined by:

$$\forall u, v \in V_a^\gamma, (u \in \mathcal{C}_{D_a^\gamma}(v) \iff u \in \mathcal{C}_G(v) \wedge \texttt{tok}_v^\gamma \neq \uparrow)$$

Remark 5

We use the notation $v \in D_a^\gamma$ as an alias for $v \in V_a^\gamma$.

Remark 6

A $\texttt{CD}^o$ with anchor a is a sub-$\texttt{D}^o$ of G_a, the $\texttt{D}^o$ under a (see Definition 5).

Lemma 8 establishes that any node v such that $\texttt{tok}_v \neq \bot$ belongs to at least one $\texttt{CD}^o$. It will be useful in the latter to prove that any action taken by some node has an effect on at least one $\texttt{CD}^o$, and therefore decreases the weight of at least one $\texttt{CD}^o$.

Lemma 8

$\forall v \in V$, if $\texttt{tok}_v^\gamma \neq \bot$, then there exists an anchor $a \in \mathbb{A}(\gamma)$ such that $v \in D_a^\gamma$

Proof: Let us consider a maximal sequence $v_0 v_1 \cdots v_k$ such that $v = v_0$ and $\forall i \in [1, k], \texttt{tok}_{v_i}^\gamma \notin \{\bot \uparrow\} \wedge v_i \in \mathcal{P}(v_{i-1})$. Since the sequence is maximal, $v_k \in \mathbb{A}(\gamma)$, and by definition we also have $\forall i, v_i \in D_{v_k}^\gamma$. Consequently, $v \in D_{v_k}^\gamma$. ■

Lemma 9 states that the set of anchors $\mathbb{A}$ of G is non-increasing: if one node v is not an anchor in a configuration γ, then it will never be an anchor in a further configuration. This lemma justifies that focusing on the decrease of the weight of $\texttt{CD}^o$ is sufficient, since no $\texttt{CD}^o$ appears.

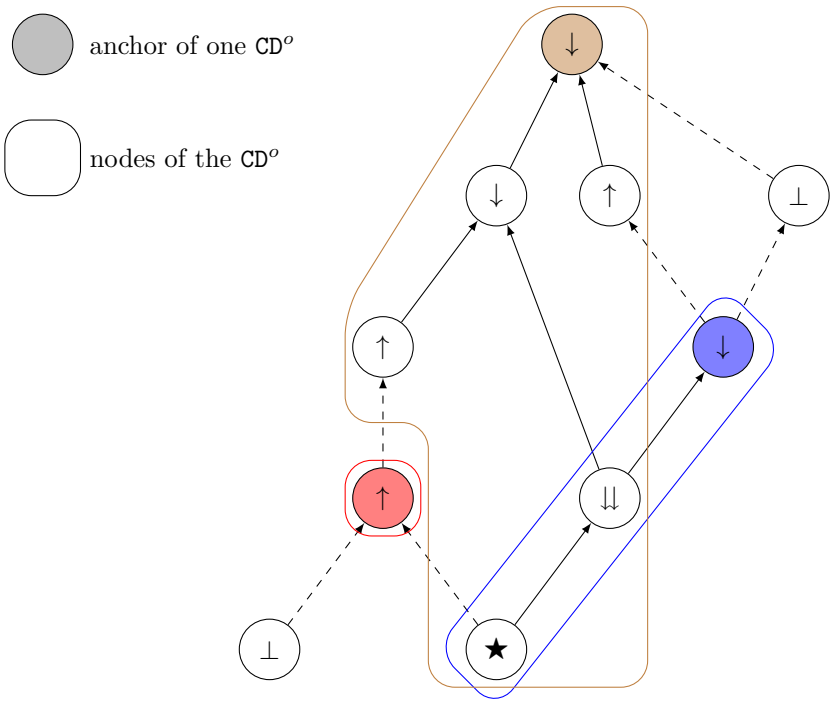

Figure 21: Example of three $\texttt{CD}^o$'s in the same graph G

Lemma 9

Let $cs = \gamma \to \gamma'$ be a computing step. We have $\mathbb{A}(\gamma') \subseteq \mathbb{A}(\gamma)$

Proof: Let us prove the equivalent statement: $V \setminus \mathbb{A}(\gamma) \subseteq V \setminus \mathbb{A}(\gamma')$. Let the node v be $v \notin \mathbb{A}(\gamma)$, and prove that $v \notin \mathbb{A}(\gamma')$.

- Suppose first that $\texttt{tok}_v^\gamma = \bot$.
  If $\texttt{tok}_v^{\gamma'} = \bot$, then $v \notin \mathbb{A}(\gamma')$. Otherwise, we have $\texttt{tok}_v^\gamma = \bot$ and $\texttt{tok}_v^{\gamma'} \neq \bot$, which is possible only if v executes $\mathbb{R}_{\texttt{Win}}$ during cs. But this is possible only if v has one parent p such that $\texttt{tok}_p^\gamma = \Downarrow$. But in such a case, we necessarily have $\texttt{tok}_p^{\gamma'} \in \{\Downarrow, \circlearrowleft, \downarrow\}$, and thus $v \notin \mathbb{A}(\gamma')$.
- Suppose now that $\texttt{tok}_v^\gamma \neq \bot$.
  Since $v \notin \mathbb{A}(\gamma)$, v has one parent p such that $\texttt{tok}_p^\gamma \notin \{\bot, \uparrow\}$. Let us prove that $\texttt{tok}_p^{\gamma'} \notin \{\bot, \uparrow\}$. Rule $\mathbb{R}_{\texttt{OfferUp}}$ is the only rule whose effect might update $\texttt{tok}$ from a state that is neither $\bot$ nor $\uparrow$, to a state that is $\bot$ or $\uparrow$. But p can execute $\mathbb{R}_{\texttt{OfferUp}}$ during cs only if $\texttt{tok}_p^\gamma = \bullet$. Under such circumstances, since $\texttt{tok}_v^\gamma \neq \bot$, $\texttt{Er}^\gamma(p) = \texttt{true}$, so p cannot execute $\mathbb{R}_{\texttt{OfferUp}}$ during cs. Thus, $\texttt{tok}_p^{\gamma'} \notin \{\bot, \uparrow\}$ and by definition, $v \notin \mathbb{A}(\gamma')$. ■

Remark 7

In the latter, when considering the effect of a computing step $\gamma \to \gamma'$ on a $\texttt{CD}^o$, we always work under the hypothesis that $a \in \mathbb{A}(\gamma')$, which also implies that $a \in \mathbb{A}(\gamma)$ according to Lemma 9.

Corollary 1

The number of $\texttt{CD}^o$ does not increase.

A $\texttt{CD}^o$ evolves through an execution of the algorithm, and in particular it can federate new nodes during some computing steps, if some node close to the $\texttt{CD}^o$ executes $\mathbb{R}_{\texttt{Win}}$. To be able to consider such situations, we introduce the notion of *border* of a $\texttt{CD}^o$, which are close nodes that might join the $\texttt{CD}^o$.

Definition 18 (*Border of a* $\texttt{CD}^o$)

Let D_a^γ be a $\texttt{CD}^o$. We define the border of D_a^γ and denote $\mathbb{B}(D_a^\gamma)$ the set of nodes $v \in V$ such that $\texttt{tok}_v^\gamma = \perp$, and $\exists p \in \mathcal{P}(v) : p \in D_a^\gamma \wedge \texttt{tok}_p^\gamma \neq \uparrow$.

The border of a $\texttt{CD}^o$ describes the nodes that can join a $\texttt{CD}^o$. On the other hand, some nodes can leave a $\texttt{CD}^o$. Lemma 10 establishes that the only nodes susceptible to leave a $\texttt{CD}^o$ during one computing step are the leaves of the $\texttt{CD}^o$: the nodes v such that $\texttt{tok}_v = \uparrow$.

Lemma 10

Let $cs = \gamma \to \gamma'$ be a computing step, and let $a \in \mathbb{A}(\gamma')$. Let $v_1 v_2 \cdots v_k$ be a branch of D_a^γ such that $v_1 = a$. Then $v_1 v_2 \cdots v_{k-1}$ is a branch of $D_a^{\gamma'}$, and if v_k does not execute $\mathbb{R}_{\texttt{Return}}$ during cs, then $v_1 v_2 \cdots v_k$ is a branch of $D_a^{\gamma'}$.

Proof: If $\mathcal{B} = v_1 v_2 \cdots v_k$ is a branch of $D_a^{\gamma'}$ the result is immediate. Let us rather suppose that $\mathcal{B}$ is not a branch of $D_a^{\gamma'}$, and let us consider the highest value i such that $\mathcal{B}_i = v_1 v_2 \cdots v_i$ is a branch of $D_a^{\gamma'}$. Remark that $v_1 = a$ is necessarily a branch of $D_a^{\gamma'}$, so $i \in [1, k-1]$.

By definition, $v_{i+1} \in \mathcal{C}_{D_a^\gamma}(v_i)$, so $\texttt{tok}_{v_i}^\gamma \notin \{\perp, \uparrow\}$ and $\texttt{tok}_{v_{i+1}}^\gamma \neq \perp$. Let us first prove that $\texttt{tok}_{v_i}^{\gamma'} \notin \{\perp, \uparrow\}$. The only possibility for the opposite would be that v_i executes $\mathbb{R}_{\texttt{OfferUp}}$ during cs. But this implies that $\texttt{tok}_{v_i}^\gamma = \bullet$ and thus $\texttt{Er}^\gamma(v_i)$ due to v_{i+1}. Thus, if activated, v_i does not execute $\mathbb{R}_{\texttt{OfferUp}}$ during cs but $\mathbb{E}_{\texttt{TrustChild}}$.

Consequently, since we have $v_{i+1} \notin \mathcal{C}_{D_a^{\gamma'}}(v_i)$, we deduce that $\texttt{tok}_{v_{i+1}}^{\gamma'} = \perp$, and thus v_{i+1} executes $\mathbb{R}_{\texttt{Return}}$ during cs. But then, $\texttt{tok}_{v_{i+1}}^\gamma = \uparrow$ and thus v_{i+1} has no children in D_a^γ.

We deduce that $v_{i+1} = v_k$, which means that $v_i = v_{k-1}$ and thus $v_1 v_2 \cdots v_{k-1}$ is a branch of $D_a^{\gamma'}$, and if v_k does not execute $\mathbb{R}_{\texttt{Return}}$ then $v_1 v_2 \cdots v_k$ is a branch of $D_a^{\gamma'}$. ■

Lemma 11 establishes constraints on which actions might be taken by a node.

Lemma 11

Let $cs = \gamma \to \gamma'$ be a computing step, and let v be a node. If $v = r$ then v can update $\texttt{c}_v$ only if $\texttt{tok}_v^\gamma = \uparrow$, and if $v \neq r$ then v can update $\texttt{c}_v$ only if $\texttt{tok}_v^\gamma = \perp$.

Proof: If $v = r$ then only $\mathbb{R}^r_{\texttt{NewDFS}}$ can update $\texttt{c}_v$, and it is enabled only if $\texttt{tok}_v^\gamma = \uparrow$. If $v \neq r$ then only $\mathbb{R}_{\texttt{Win}}$ and $\mathbb{R}_{\texttt{FakeWin}}$ can update $\texttt{c}_v$, and both are enabled only if $\texttt{tok}_v^\gamma = \perp$. ■

C.2.3 Introduction to the potential function W

To prove that one $\texttt{CD}^o$ can only be activated a finite number of time, we associate a weight to each $\texttt{CD}^o$ in each configuration γ. We design this weight function (or potential function) such that if one node is activated in the $\texttt{CD}^o$ or near it during the computing step $\gamma \to \gamma'$, then the weight of the $\texttt{CD}^o$ is less in γ' than in γ. The goal is to provide the domain of weights with a well-founded order, so that such decreasing sequences are necessarily finite. Recall that decreasing occurs only during periods during which no execution of $\mathbb{R}^r_{\texttt{NewDFS}}$ resets the weight of a $\texttt{CD}^o$ to a high value.

Unfortunately, the structure of $\texttt{CD}^o$ is too complex for it can be valuated by a object as simple as an integer. The most natural way to associate a weight to a $\texttt{CD}^o$ is to build a $\texttt{D}^o$, similar to

the $\texttt{CD}^o$ considered, but whose nodes are labeled with weights. We call such structures Weighted DODAGs, denoted $\texttt{WD}^o$'s for short.

In this section, we consider $(M, \leq)$ an arbitrary well-founded set. Although our weight function is defined on $\texttt{CD}^o$'s, it is simpler to work through this section by considering arbitrary $\texttt{D}^o$'s. Since $\texttt{CD}^o$'s have a $\texttt{D}^o$ structure, all this work naturally applies to $\texttt{CD}^o$'s.

Definition 19 formally introduces $\texttt{WD}^o$'s: the object on which we will define a well-founded order and that will later be considered as the weight of $\texttt{D}^o$'s.

Definition 19 (*Weighted DODAG*)

A weighted $\texttt{D}^o$ ($\texttt{WD}^o$) is a $\texttt{D}^o$ with labels in M: $(D_a, \texttt{W})$, where D_a is a $\texttt{D}^o$ and $\texttt{W} \in M^{V_a}$ is a map that associates an element of M to each node of D_a.

The next step is to define an order of $\texttt{WD}^o$'s. This is feasible only if the two $\texttt{WD}^o$'s we compare are based on $\texttt{D}^o$'s having the same anchor a. This won't be restrictive since in practice, we compare the weights of the same $\texttt{CD}^o$ at different, consecutive, configuration of an execution.

To compare $\texttt{WD}^o$'s, the first step is to compare branches of $\texttt{WD}^o$'s. Since two $\texttt{WD}^o$'s that share the same anchor do not necessarily share the same branches in their topology, the appropriate notion to this comparison is the projection of $\texttt{WD}^o$'s on the branches of the $\texttt{D}^o$ under their common anchor. This is similar to what we introduced in Definition 7.

Definition 20 (*Projection of a $\texttt{WD}^o$ on a Branch*)

Let $(D_a, \texttt{W})$ be a $\texttt{WD}^o$, and let $\mathcal{B} = v_1 v_2 \cdots v_k$ be a maximal branch of G_a. We call projection of $(D_a, \texttt{W})$ on $\mathcal{B}$, and denote $(D_a, \texttt{W})(\mathcal{B})$ the sequence $\texttt{W}(v_1)\texttt{W}(v_2)\cdots\texttt{W}(v_j)$, where $D_a(\mathcal{B}) = v_1 v_2 \cdots v_j$ is the projection of D_a on $\mathcal{B}$.

Since the branches of two $\texttt{WD}^o$'s may have variable length, we compare them using the alphabetical order (see Definition 9). This order is well-founded since the branches of the $\texttt{WD}^o$'s are all bounded by n, the number of nodes in the graph.

To compare $\texttt{WD}^o$'s, we compare the weights of all the branches which start at a. One $\texttt{WD}^o$ D_a^1 is less than or equal to one other $\texttt{WD}^o$ D_a^2 if, for any branch $\mathcal{B}$ of G_a the $\texttt{D}^o$ under a, the projection of D_a^1 on $\mathcal{B}$ is less than or equal to the projection of D_a^2 on $\mathcal{B}$.

Definition 21 presents a well-founded order on $\texttt{WD}^o$'s.

Definition 21 (*Order on $\texttt{WD}^o$'s*)

We define a partial order on $\texttt{WD}^o$'s which share the same anchor by:

$$(D'_a, \texttt{W}') \preceq (D_a, \texttt{W}) \iff \begin{array}{l} \text{for all maximal branch } \mathcal{B} \text{ of } G_a, \\ (D'_a, \texttt{W}')(\mathcal{B}) \preceq_\alpha (D_a, \texttt{W})(\mathcal{B}) \end{array}$$

By construction, the resulting order is not a total order. Indeed, there exists $\texttt{D}^o$'s and branches such that $(D'_a, \texttt{W}')(\mathcal{B}_1) \prec_\alpha (D_a, \texttt{W})(\mathcal{B}_1)$, on the one hand, and $(D_a, \texttt{W})(\mathcal{B}_2) \prec_\alpha (D'_a, \texttt{W}')(\mathcal{B}_2)$ on the other hand. Yet, we guarantee in the following sections that any computing step makes the weight of the $\texttt{D}^o$ decrease (which implies that the corresponding $\texttt{WD}^o$'s are comparable).

Lemma 12

Definition 21 defines a well-founded order.

Proof: By construction, $\preceq$ is reflexive, anti-symmetric, and transitive. Let us prove that it is also well-founded. Consider an infinite sequence of $\texttt{WD}^o$'s $(D_a^n, \texttt{W}^n)_{n\in\mathbb{N}}$ such that $\forall n, (D_a^{n+1}, \texttt{W}^{n+1}) \preceq (D_a^n, \texttt{W}^n)$, and let us prove that there exists $N \in \mathbb{N}$ such that $\forall n \geq N, (D_a^n, \texttt{W}^n) = (D_a^N, \texttt{W}^N)$.

Let $\mathcal{B}$ be a maximal branch of G_a, and let k be the length of $\mathcal{B}$. By definition, $\forall n, (D_a^{n+1}, \texttt{W}^{n+1})(\mathcal{B}) \preceq_\alpha (D_a^n, \texttt{W}^n)(\mathcal{B})$. But $\preceq_\alpha$ is a well-founded order on sequences of length at most k, and $\forall n, (D_a^n, \texttt{W}^n)(\mathcal{B})$

is a sequence of length at most k. Consequently, there exists $N_{\mathcal{B}} \in \mathbb{N}$ such that $\forall n \geq N_{\mathcal{B}}, (D_a^{n+1}, \mathtt{W}^{n+1})(\mathcal{B}) = (D_a^n, \mathtt{W}^n)(\mathcal{B})$.

Since there exists a finite number of maximal branches $\mathcal{B}$ of G_a, let us consider $N = \max_{\mathcal{B}}(N_{\mathcal{B}})$. By definition, $\forall n \geq N$, for all maximal branch $\mathcal{B}$ of G_a, $(D_a^n, \mathtt{W}^n)(\mathcal{B}) = (D_a^N, \mathtt{W}^N)(\mathcal{B})$, and thus, $\forall n \geq N, (D_a^N, \mathtt{W}^N) \preceq (D_a^n, \mathtt{W}^n)$. **Since** $\preceq$ is an order, we have $\forall n \geq N, (D_a^n, \mathtt{W}^n) = (D_a^N, \mathtt{W}^N)$. ■

We now show how to associate a $\mathtt{WD}^o$ to any $\mathtt{CD}^o$ as soon as we are given a weight function $\mathtt{W}$ on nodes of G. This association is itself a weight function on $\mathtt{CD}^o$, induced by the weight function on nodes. For simplicity, we also call $\mathtt{W}$ the induced weight function.

Contrary to what was presented in Definition 19, the weight function on nodes may depend on the global configuration γ of the system, and not only of the state of nodes v. More formally, we now consider a weight function $\mathtt{W}(v, \gamma)$ which associates an element of M to each node in a given configuration. This does not pose any fundamental problem since, given γ, we can still associate a $\mathtt{WD}^o$ to any $\mathtt{CD}^o$ D_a^γ. We can as well associate, given γ, a weight to any projection of D_a^γ on any branch of G_a. This is formally stated in Definition 22.

Definition 22 (*Weight of a circulation DAG*)

Let $\mathtt{W} : V \times \Gamma \to M$ be a function that associates a weight to any node of a configuration.

For any fixed configuration $\gamma \in \Gamma$, we define $\mathtt{W}(\,\cdot\,, \gamma) \in M^{V_{D_a}}$ the function that associates to any node $v \in V_{D_a}$ the weight of v in γ: $\mathtt{W}(\,\cdot\,, \gamma)(v) = \mathtt{W}(v, \gamma)$.

Then, for all $\gamma \in \Gamma$, for all $\mathtt{CD}^o$ D_a^γ at γ, we can consider $\mathtt{W}(D_a^\gamma, \gamma)$ the weight of D_a^γ in γ, which is the $\mathtt{WD}^o$ $(D_a^\gamma, \mathtt{W}(\,\cdot\,, \gamma))$.

We also define the weight of a branch $\mathcal{B} = v_1 v_2 \cdots v_k$ of D_a^γ in γ, and denote $\mathtt{W}(D_a^\gamma(\mathcal{B}), \gamma) = \mathtt{W}(D_a^\gamma, \gamma)(\mathcal{B}) = (D_a^\gamma, \mathtt{W}(\,\cdot\,, \gamma))(\mathcal{B}) = \mathtt{W}(v_1, \gamma)\mathtt{W}(v_2, \gamma) \cdots \mathtt{W}(v_k, \gamma)$.

To deal with all the different aspects of the algorithm, we actually define several weight functions in Sections C.2.4 to C.2.8, dealing with the different aspects of our algorithm. Each of these functions defines one different $\mathtt{WD}^o$ to the same $\mathtt{CD}^o$. In order to deduce global properties on the evolution of the entire algorithm, we must combine all those function and to compare the different induces $\mathtt{WD}^o$'s all at once. To do so, we simply use the lexicographic order on $\mathtt{WD}^o$'s that is induced by the order $\preceq$ introduced in Definition 21.

Definition 23 (*Lexicographic order on $\mathtt{WD}^o$'s*)

We denote by $\preceq^k$ the lexicographic order on k-tuples of $\mathtt{WD}^o$'s which share the same anchor, induced by the order $\preceq$.

$\forall k \in \mathbb{N}, \preceq^k$ is a well-founded order, as a lexicographic order induced by a well-founded order.

Our goal is now to prove two properties. The first one is that the $\mathtt{WD}^o$'s associated to one $\mathtt{CD}^o$ never increases unless the anchor of the $\mathtt{CD}^o$ is the root, and the root executes $\mathbb{R}^r_{\mathtt{NewDFS}}$. The second property is that each time one node of the $\mathtt{CD}^o$, or near it, is activated (by a rule which is not $\mathbb{R}^r_{\mathtt{NewDFS}}$), the weight of the $\mathtt{CD}^o$ decreases. A weight function which respects the first property is said *pre-admissible*, and it is said *admissible* if it respects both.

Since the weight of a $\mathtt{CD}^o$ is expressed as a tuple of $\mathtt{WD}^o$'s, the formal definitions of admissibility and pre-admissibility lie on the lexicographic order introduced in Definition 23. Each potential function has an even stronger weight as it appears on the first component of the tuple.

The exact definition of admissibility actually depends on how we define what a node *near a* $\mathtt{CD}^o$ is. This definition depends on constraints which will emerge from the definitions of the weight functions, and therefore will be provided later, in Definition 26 of Section C.2.9.

We give here the definition of pre-admissibility.

Definition 24 (*Pre-admissible weight-functions*)

Let $\mathtt{W}_1, \ldots, \mathtt{W}_k : V \times \Gamma \to M$ be k weight functions on nodes. We say that $(\mathtt{W}_1, \ldots, \mathtt{W}_k)$ is pre-admissible if for any computing step $cs = \gamma \to \gamma'$ such that r does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs, for all $\mathtt{CD}^o$ $D_a^{\gamma'}$, we have

$$\big(\mathtt{W}_1(D_a^{\gamma'}, \gamma'), \cdots, \mathtt{W}_k(D_a^{\gamma'}, \gamma')\big) \preceq^k \big(\mathtt{W}_1(D_a^{\gamma}, \gamma), \cdots, \mathtt{W}_k(D_a^{\gamma}, \gamma)\big).$$

Remark 8

In order to have relatively short and readable equations in the following, the previous inequality will often be written:

$$\big(\mathtt{W}_1, \ldots, \mathtt{W}_k\big)(D_a^{\gamma'}, \gamma') \preceq^k \big(\mathtt{W}_1, \ldots, \mathtt{W}_k\big)(D_a^{\gamma}, \gamma)$$

In Sections C.2.4 to C.2.8 we define five weight functions on nodes, that belong to $V \times \Gamma \to \mathbb{N}$: which are $\mathtt{W}_{\mathtt{Ch}}, \mathtt{W}_{\mathtt{Er}}, \mathtt{W}_{\mathtt{Circ}}, \mathtt{W}_{\mathtt{Play}}$ and $\mathtt{W}_{\mathtt{Nego}}$. Each one of those 5 functions deals with one specific part of the algorithm, by decreasing order of importance, and is treated separately in one of the next five sections. Thus, we define, for each configuration, 5 $\mathtt{WD}^o$'s that, associated with the lexicographic order $\preceq^5$, will allow us to formally prove that the token circulation terminates. Figure 22 summarizes the effects of the rules, and for each of them indicates which component of $\mathtt{W}$ it decreases.

We prove that $\mathtt{W} = (\mathtt{W}_{\mathtt{Ch}}, \mathtt{W}_{\mathtt{Er}}, \mathtt{W}_{\mathtt{Circ}}, \mathtt{W}_{\mathtt{Play}}, \mathtt{W}_{\mathtt{Nego}})$ is pre-admissible, by proving step by step that all the prefixes of that 5-tuple are pre-admissible. We also establish some properties that will allow us to prove, in Section C.2.9, that $(\mathtt{W}_{\mathtt{Ch}}, \mathtt{W}_{\mathtt{Er}}, \mathtt{W}_{\mathtt{Circ}}, \mathtt{W}_{\mathtt{Play}}, \mathtt{W}_{\mathtt{Nego}})$ is admissible.

C.2.4 First component of $\mathtt{W}$: $\mathtt{W}_{\mathtt{Ch}}$ future children of v

Explanations One property which guarantees that circulation rounds are finite is that one node can receive the token from its parent only once, as long as the parent does not return the token to its own parent.

Indeed, once a node u receives the token from one of its parent p, it takes the same color as that parent, and its variable $\mathtt{play}$ becomes $\mathtt{W}$. Variable $\mathtt{play}_u$ then cannot be set to $\mathtt{P}$ before $\mathtt{tok}_p$ is set to $\uparrow$, which implies that u cannot take the token once again from p before p itself returns the token to its own parent.

For that reason, the first component of the weight function $\mathtt{W}$ should, at least, count on each node $u \in V$ the number of children of u that are susceptible to execute $\mathbb{R}_{\mathtt{Win}}$ before u execute $\mathbb{R}_{\mathtt{OfferUp}}$. The variables that we must consider to establish if one node is susceptible to execute $\mathbb{R}_{\mathtt{Win}}$ are $\mathtt{play}$ and $\mathtt{c}$. In the following we detail how we formally define this count.

Let us consider one node v and one of its children u. Remark first that if $\mathtt{tok}_v \in \{\bot, \uparrow\}$, then the weight of v can be defined as 0. Indeed, such a node cannot be involved in a pass of the token to its children before it itself receives the token from one of its parents. Thus, when v receives the token from its parent, its own weight increases from 0 to the actual count of how many children susceptible to execute $\mathbb{R}_{\mathtt{Win}}$ it has. But at the same moment, the weight of v's parent decreases by one due to v cannot execute $\mathbb{R}_{\mathtt{Win}}$ anymore. Consequently, due to how we defined $\preceq_\alpha$ on weighted chains, the weight of each branch of a $\mathtt{CD}^o$ that contains v and its parent decrease when v execute $\mathbb{R}_{\mathtt{Win}}$.

We now consider one node v such that $\mathtt{tok}_v \notin \{\bot, \uparrow\}$ and one child u of v. In this section, we only focus on the transmission of the token between nodes, and not on the transitions of the

node	Rule	id	tok	c	play	W ↘
$v \neq r$	$\mathbb{R}_{\texttt{Win}}$		$\bot \to \bigstar$	switch	$\texttt{P} \to \texttt{W}$	$\texttt{W}_{\texttt{Ch}}$
$v \neq r$	$\mathbb{R}_{\texttt{FakeWin}}$			switch	$\texttt{P} \to \{\texttt{W},\texttt{F}\}$	$\texttt{W}_{\texttt{Ch}}$
$\forall v \in V$	$\mathbb{E}_{\texttt{TrustChild}}$		$\{\bullet, \downdownarrows, \circlearrowleft, \circ, \bigstar\} \to \downarrow$			$\texttt{W}_{\texttt{Er}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{Give}}$		$\bullet \to \downdownarrows$			$\texttt{W}_{\texttt{Circ}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{NewPlay}}$		$\downdownarrows \to \circlearrowleft$			$\texttt{W}_{\texttt{Circ}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{Drop}}$		$\circlearrowleft \to \downarrow$			$\texttt{W}_{\texttt{Circ}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{Nego}}$	$(1, \bot)$	$\bigstar \to \bullet$			$\texttt{W}_{\texttt{Circ}}$
$v \neq r$	$\mathbb{R}_{\texttt{OfferUp}}$		$\bullet \to \uparrow$			$\texttt{W}_{\texttt{Circ}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{Receive}}$		$\downarrow \to \circ$			$\texttt{W}_{\texttt{Circ}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{ReNego}}$	$(1, \bot)$	$\circ \to \bullet$			$\texttt{W}_{\texttt{Circ}}$
$v \neq r$	$\mathbb{R}_{\texttt{Return}}$		$\uparrow \to \bot$		$\to \{\texttt{W},\texttt{F}\}$	$\texttt{W}_{\texttt{Circ}}$
$v \neq r$	$\mathbb{R}_{\texttt{Lose}}$				$\texttt{P} \to \texttt{L}$	$\texttt{W}_{\texttt{Play}}$
$v \neq r$	$\mathbb{R}_{\texttt{ReplayD}}$				$\texttt{L} \to \texttt{P}$	$\texttt{W}_{\texttt{Play}}$
$v \neq r$	$\mathbb{R}_{\texttt{ReplayUp}}$				$\texttt{W} \to \texttt{P}$	$\texttt{W}_{\texttt{Play}}$
$v \neq r$	$\mathbb{R}_{\texttt{Fake}}$				$\texttt{W} \to \texttt{F}$	$\texttt{W}_{\texttt{Play}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{maxPos}}$	$\max_{u \in \mathcal{C}(v)} \texttt{id}_u$				$\texttt{W}_{\texttt{Nego}}$
$\forall v \in V$	$\mathbb{R}_{\texttt{NewPh}}$	$(\texttt{ph}+1, \bot)$				$\texttt{W}_{\texttt{Nego}}$
$v \neq r$	$\mathbb{R}_{\texttt{NewBit}}$	$(\texttt{ph}, \texttt{Bit}(\texttt{ph}))$				$\texttt{W}_{\texttt{Nego}}$
$v \neq r$	$\mathbb{R}_{\texttt{ReWin}}$				$\texttt{F} \to \texttt{W}$	
r	$\mathbb{R}^{r}_{\texttt{NewDFS}}$		$\uparrow \to \bigstar$	switch		

Figure 22: Dedicated weight function for each rule. For readability, one color is associated to each weight function.

variable $\texttt{tok}$ on one node. Thus, we treat indifferently the different cases when $\texttt{tok}_v \in \{\bullet, \bigstar, \downdownarrows, \circlearrowleft, \circ, \downarrow\}$. Let us establish under which circumstances u might receive the token from v.

- Node u can receive the token, at first, if $\texttt{c}_u \neq \texttt{c}_v$ and $\texttt{play}_u \in \{\texttt{P},\texttt{L}\}$, by simply executing $\mathbb{R}_{\texttt{Win}}$ (or after one execution of $\mathbb{R}_{\texttt{ReplayD}}$).
- Secondly, if $\texttt{c}_u \neq \texttt{c}_v$ and $\texttt{play}_u \in \{\texttt{W},\texttt{F}\}$, then u can execute $\mathbb{R}_{\texttt{ReplayUp}}$, and $\mathbb{R}_{\texttt{ReWin}}$ if necessary, and then arrives in the previous situation. Therefore, we must count u as a node that might receive the token.
- Finally, if $\texttt{c}_u = \texttt{c}_v$ and $\texttt{play}_u \in \{\texttt{P},\texttt{L}\}$, then u can execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, and then arrives in the previous situation.

Note that if $\texttt{c}_u = \texttt{c}_v$ and $\texttt{play}_u \in \{\texttt{W},\texttt{F}\}$ then u cannot execute $\mathbb{R}_{\texttt{Win}}$ nor $\mathbb{R}_{\texttt{FakeWin}}$ due to $\texttt{play}_u$, and cannot execute $\mathbb{R}_{\texttt{ReplayUp}}$ since $\texttt{tok}_v \in \{\bot, \uparrow\}$. This is described in Figure 23.

All the transitions presented in the diagram of Figure 23 bring the node that executes the corresponding rule closer to the stable state where $\texttt{c}_u = \texttt{c}_v \wedge \texttt{play}_u \in \{\texttt{W},\texttt{F}\}$. We need to fully describe the process that leads to the passing of the token to one child. Therefore, we define $\texttt{W}_{\texttt{Ch}}$ such that any execution of a rule that corresponds to a transition presented in Figure 23 makes $\texttt{W}_{\texttt{Ch}}$ decrease on the parent of the node that executes it.

Yet, remark that we cannot treat $\mathbb{R}_{\texttt{ReplayUp}}$ the same way as we treat $\mathbb{R}_{\texttt{Win}}$ and $\mathbb{R}_{\texttt{FakeWin}}$. Indeed, any execution of $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ by one node u requires that at least one parent v of u has its variable $\texttt{tok}_v$ different from $\{\bot, \uparrow\}$, which means that any such execution corresponds to a transition on that diagram for that node v. On the contrary, one node u might execute $\mathbb{R}_{\texttt{ReplayUp}}$ without any parent v such that $\texttt{tok}_v \notin \{\bot, \uparrow\}$. Thus, although we treat some, we

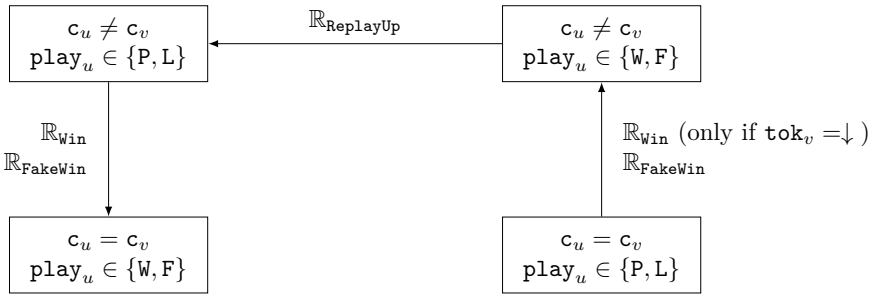

Figure 23: Evolution of the state of a child of node v with $\mathtt{tok}_v \notin \{\bot, \uparrow\}$.

do not treat all possible executions of $\mathbb{R}_{\mathtt{ReplayUp}}$ with $\mathtt{W}_{\mathtt{Ch}}$. This means that, by the end of this section, we prove that any activation of $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$ makes $\mathtt{W}$ decrease, and that some specific activation of $\mathbb{R}_{\mathtt{ReplayUp}}$ make $\mathtt{W}$ decrease. This piece of knowledge will nevertheless be useful in the following sections until we prove that any activation of $\mathbb{R}_{\mathtt{ReplayUp}}$ makes $\mathtt{W}$ decrease. This is formally stated in Theorems 3 and 4.

Definitions Since children u of v move along the different boxes of Figure 23, we must design $\mathtt{W}_{\mathtt{Ch}}$ such that any transition taken by u makes $\mathtt{W}_{\mathtt{Ch}}(v, \gamma') < \mathtt{W}_{\mathtt{Ch}}(v, \gamma)$. Our solution is to attribute different coefficients to the different boxes: the further node u is from the stable configuration $\mathtt{c}_u = \mathtt{c}_v \wedge \mathtt{play}_u \in \{\mathtt{P},\mathtt{L}\}$, the higher the coefficient. Somehow, the coefficient represents the number of transitions needed by one node to reach the stable configuration. The coefficients are the integer above and below the boxes in Figure 24.

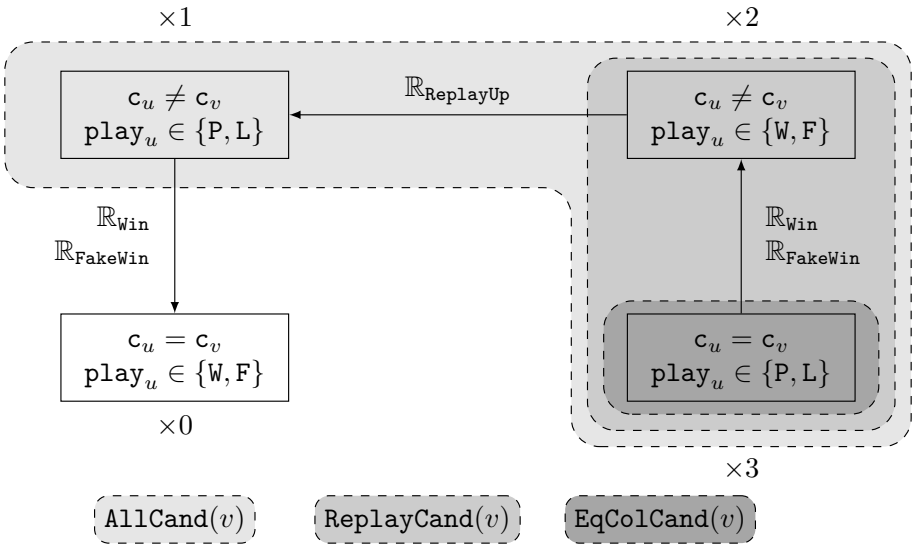

Figure 24: Weight associated to a child of node v with $\mathtt{tok}_v \notin \{\bot, \uparrow\}$.

In practice, to make the proof easier to approach, we define 3 sets on each node v, such that any transition taken by one child u of v brings u out of one of the set. Those sets have a non-empty intersection, and are such that the coefficient attributed to one node is the number of sets it belongs to.

- One set includes all the nodes that are potential candidates to receiving the token from

v, that is to say all the children u of v that are not in a stable state.

- One other set, included in the first one, includes all of those candidates that must execute $\mathbb{R}_{\texttt{ReplayUp}}$ before they might receive the token from v.
- Finally, the last set, included in the second one, includes all of those candidates that must execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ before they can execute $\mathbb{R}_{\texttt{ReplayUp}}$ and receive the token from v, that are also the candidates that are the same color as v.

Let us define those three sets, starting from the last, and smallest, one.

$$\texttt{EqColCand}^{\gamma}(v) = \{u \in \mathcal{C}(v) \mid \texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma} \wedge \texttt{play}_u^{\gamma} \in \{\texttt{P}, \texttt{L}\}\} \tag{34}$$

$$\texttt{ReplayCand}^{\gamma}(v) = \texttt{EqColCand}^{\gamma}(v) \cup \{u \in \mathcal{C}(v) \mid \texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma} \wedge \texttt{play}_u^{\gamma} \in \{\texttt{W}, \texttt{F}\}\} \tag{35}$$

$$\texttt{AllCand}^{\gamma}(v) = \texttt{ReplayCand}^{\gamma}(v) \cup \{u \in \mathcal{C}(v) \mid \texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma} \wedge \texttt{play}_u^{\gamma} \in \{\texttt{P}, \texttt{L}\}\} \tag{36}$$

Finally, we can define $\texttt{W}_{\texttt{Ch}}(v, \gamma)$ such that the coefficients on Figure 24 are respected:

$$\texttt{W}_{\texttt{Ch}}(v, \gamma) = \begin{cases} 0 & \text{if } \texttt{tok}_v^{\gamma} \in \{\bot, \uparrow\} \\ \left|\texttt{AllCand}^{\gamma}(v)\right| + \left|\texttt{ReplayCand}^{\gamma}(v)\right| + \left|\texttt{EqColCand}^{\gamma}(v)\right| & \text{otherwise} \end{cases} \tag{37}$$

Proofs Lemmas 13, 14, and 15 state that $\forall v \in V$, the three sets defined previously can only decrease during an execution, as long as $\texttt{tok}_v \notin \{\bot, \uparrow\}$. Basically, we prove that there is no other transition that the ones we represented in Figure 24. These three lemmas are one of the main arguments to establish that $\texttt{W}_{\texttt{Ch}}$ is a pre-admissible weight function.

Lemma 13

Let $cs = \gamma \to \gamma'$ be a computing step and let $v \in V$ be a node. If $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow\}$ then $\texttt{AllCand}^{\gamma'}(v) \subseteq \texttt{AllCand}^{\gamma}(v)$.

Proof: We prove the equivalent statement:

$$\mathcal{C}(v) \setminus \texttt{AllCand}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{AllCand}^{\gamma'}(v).$$

Since $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow\}$, then according to Lemma 11, v does not update $\texttt{c}_v$ during cs. Let us consider one child u of v such that $u \notin \texttt{AllCand}^{\gamma}(v)$. By definition, $\texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma} \wedge \texttt{play}_u^{\gamma} \in \{\texttt{W}, \texttt{F}\}$. During cs, u does not execute $\mathbb{R}_{\texttt{Win}}$ nor $\mathbb{R}_{\texttt{FakeWin}}$ since $\texttt{play}_u^{\gamma} \neq \texttt{P}$, and does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ since it has at least one parent. Furthermore, u does not execute $\mathbb{R}_{\texttt{ReplayUp}}$ during cs since $v \in \mathcal{P}_{\texttt{eq}}{}^{\gamma}(u) \wedge \texttt{tok}_v^{\gamma} = \uparrow$. Thus, $\texttt{c}_u^{\gamma'} = \texttt{c}_v^{\gamma'} \wedge \texttt{play}_u^{\gamma'} \in \{\texttt{W}, \texttt{F}\}$, in other words, $u \notin \texttt{AllCand}^{\gamma'}(v)$. As a result, $\mathcal{C}(v) \setminus \texttt{AllCand}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{AllCand}^{\gamma'}(v)$. ■

Lemma 14

Let $cs = \gamma \to \gamma'$ be a computing step and let $v \in V$ be a node. If $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow\}$ then $\texttt{ReplayCand}^{\gamma'}(v) \subseteq \texttt{ReplayCand}^{\gamma}(v)$.

Proof: We prove the equivalent statement:

$$\mathcal{C}(v) \setminus \texttt{ReplayCand}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{ReplayCand}^{\gamma'}(v).$$

Since $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow\}$, then according to Lemma 11, v does not update $\texttt{c}_v$ during cs. Let us consider one child u of v such that $u \notin \texttt{ReplayCand}^{\gamma}(v)$. If $u \notin \texttt{AllCand}^{\gamma}(v)$ then according

to Lemma 13, $u \notin \texttt{AllCand}^{\gamma'}(v)$ and thus $u \notin \texttt{ReplayCand}^{\gamma'}(v)$. Let us suppose now that $u \in \texttt{AllCand}^{\gamma}(v) \setminus \texttt{ReplayCand}^{\gamma}(v)$, *i.e.* $\texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma} \wedge \texttt{play}_u^{\gamma} \in \{\texttt{P}, \texttt{L}\}$. If u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ during cs, then $\texttt{c}_u^{\gamma'} = \texttt{c}_v^{\gamma'} \wedge \texttt{play}_u^{\gamma'} = \texttt{W}$ and thus $u \notin \texttt{ReplayCand}^{\gamma'}(v)$. Otherwise, since u has at least one parent it does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ and thus does not update $\texttt{c}_v$, and it can neither update $\texttt{c}_u$ to W or L, and thus $u \notin \texttt{ReplayCand}^{\gamma'}(v)$. As a result, $\mathcal{C}(v) \setminus \texttt{ReplayCand}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{ReplayCand}^{\gamma'}(v)$. ■

Lemma 15

Let $cs = \gamma \to \gamma'$ be a computing step and let $v \in V$ be a node. If $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow\}$ then $\texttt{EqColCand}^{\gamma'}(v) \subseteq \texttt{EqColCand}^{\gamma}(v)$.

Proof: We prove the equivalent statement:

$$\mathcal{C}(v) \setminus \texttt{EqColCand}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{EqColCand}^{\gamma'}(v).$$

Since $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow\}$, then according to Lemma 11, v does not update $\texttt{c}_v$ during cs. Let us consider one child u of v such that $u \notin \texttt{EqColCand}^{\gamma}(v)$. If $u \notin \texttt{ReplayCand}^{\gamma}(v)$ then according to Lemma 14, $u \notin \texttt{ReplayCand}^{\gamma'}(v)$ and thus $u \notin \texttt{EqColCand}^{\gamma'}(v)$.

Let us suppose now that $u \in \texttt{ReplayCand}^{\gamma}(v) \setminus \texttt{EqColCand}^{\gamma}(v)$, *i.e.* $\texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma} \wedge \texttt{play}_u^{\gamma} \in \{\texttt{W}, \texttt{F}\}$. Since u has at least one parent, it does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs, and since $\texttt{play}_u^{\gamma} \neq \texttt{P}$, u does not execute $\mathbb{R}_{\texttt{Win}}$ nor $\mathbb{R}_{\texttt{FakeWin}}$ during cs. Thus, we have $\texttt{c}_u^{\gamma'} \neq \texttt{c}_v^{\gamma'}$ and thus $u \notin \texttt{EqColCand}(v, \gamma')$. As a result, $\mathcal{C}(v) \setminus \texttt{EqColCand}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{EqColCand}^{\gamma'}(v)$. ■

Lemma 16 establishes that if one node executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, or $\mathbb{R}_{\texttt{ReplayUp}}$, then the weight of all its parents in any $\texttt{CD}^o$ decrease. Basically, we formally prove that any such activation corresponds to one of the transition represented in Figure 24, and thus that it makes one of the three sets decrease.

Lemma 16

Let $cs = \gamma \to \gamma'$ be a computing step, and let $v \in V$ be a node. If v executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, or $\mathbb{R}_{\texttt{ReplayUp}}$ during cs, then $\forall p \in \mathcal{P}(v)$ such that $\texttt{tok}_p^{\gamma} \notin \{\bot, \uparrow\}$, we have $\texttt{W}_{\texttt{Ch}}(p, \gamma') < \texttt{W}_{\texttt{Ch}}(p, \gamma)$.

Proof: Let us consider $p \in \mathcal{P}(v)$ such that $\texttt{tok}_p^{\gamma} \notin \{\bot, \uparrow\}$. Lemma 11 guarantees that $\texttt{c}_p^{\gamma'} = \texttt{c}_p^{\gamma}$.

Suppose first that v executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ during cs, which induces $\texttt{play}_v^{\gamma} = \texttt{P}$ and $\texttt{play}_v^{\gamma'} \in \{\texttt{W}, \texttt{F}\}$.

If $\texttt{c}_v^{\gamma} = \texttt{c}_p^{\gamma}$ then $v \in \texttt{EqColCand}^{\gamma}(p)$. But since $\texttt{play}_v^{\gamma'} \in \{\texttt{W}, \texttt{F}\}$, $v \notin \texttt{EqColCand}^{\gamma'}(p)$, which according to Lemma 15 means that $\texttt{EqColCand}^{\gamma'}(p) \subsetneq \texttt{EqColCand}^{\gamma}(p)$. According to Lemmas 13 and 14, we obtain $\texttt{W}_{\texttt{Ch}}(p, \gamma') < \texttt{W}_{\texttt{Ch}}(p, \gamma)$.

Otherwise, $\texttt{c}_v^{\gamma} \neq \texttt{c}_p^{\gamma}$. Since $\texttt{play}_v^{\gamma} = \texttt{P}$, we have $v \in \texttt{AllCand}^{\gamma}(p)$, and since $\texttt{play}_v^{\gamma} \in \{\texttt{W}, \texttt{F}\}$ and v switches its color during cs, we also have $v \notin \texttt{AllCand}^{\gamma'}(p)$. According to Lemma 13, $\texttt{AllCand}^{\gamma'}(p) \subsetneq \texttt{AllCand}^{\gamma}(p)$, and according to Lemmas 14 and 15, we obtain $\texttt{W}_{\texttt{Ch}}(p, \gamma') < \texttt{W}_{\texttt{Ch}}(p, \gamma)$.

Let us now suppose that v executes $\mathbb{R}_{\texttt{ReplayUp}}$ during cs. Since $\texttt{tok}_p^{\gamma} \notin \{\bot, \uparrow\}$, predicate $\texttt{ReplayW}^{\gamma}(v)$ implies that $\texttt{c}_p^{\gamma} \neq \texttt{c}_v^{\gamma}$. Since v execute $\mathbb{R}_{\texttt{ReplayUp}}$ during cs, we have $\texttt{play}_v^{\gamma} = \texttt{W}$, and thus $v \in \texttt{ReplayCand}^{\gamma}(p)$, and we also know that v updates $\texttt{play}_v$ to P and that it does not update its variable $\texttt{c}_v$ which implies that $\texttt{c}_v^{\gamma'} \neq \texttt{c}_p^{\gamma'} \wedge \texttt{play}_v^{\gamma'} = \texttt{P}$. Thus, $v \notin \texttt{ReplayCand}^{\gamma'}(p)$ so according to Lemma 14 means that $\texttt{ReplayCand}^{\gamma'}(p) \subsetneq \texttt{ReplayCand}^{\gamma}(p)$. According to Lemmas 13 and 15, we obtain $\texttt{W}_{\texttt{Ch}}(p, \gamma') < \texttt{W}_{\texttt{Ch}}(p, \gamma)$. ■

Lemma 17 establishes that the function $\mathtt{W_{Ch}}$ is adapted to Definitions 9 and 21. Indeed, it might happen that one node sees its weight increase, but only if all of its parents see their weight decrease.

Lemma 17

Let $cs = \gamma \to \gamma'$ be a computing step and let $v \in V$ such that v does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs. If $\mathtt{W_{Ch}}(v,\gamma') > \mathtt{W_{Ch}}(v,\gamma)$ then $\forall p \in \mathcal{P}(v)$ such that $\mathtt{tok}_p^\gamma \notin \{\bot, \uparrow\}$, we have $\mathtt{W_{Ch}}(p,\gamma') < \mathtt{W_{Ch}}(p,\gamma)$, and there exists at least one such p.

Proof: According to Lemmas 13, 14, and 15, if $\mathtt{tok}_v^\gamma \in \{\bullet, \bigstar, \downdownarrows, \circlearrowleft, \downarrow, \circ\}$, then $\mathtt{W_{Ch}}(v,\gamma') \leq \mathtt{W_{Ch}}(v,\gamma)$. Furthermore, if $\mathtt{tok}_v^\gamma = \uparrow$, then $\mathtt{tok}_v^{\gamma'} \in \{\uparrow, \bot\}$ and thus $\mathtt{W_{Ch}}(v,\gamma') = \mathtt{W_{Ch}}(v,\gamma)$.

Finally, if $\mathtt{tok}_v^\gamma = \bot$, then $\mathtt{W_{Ch}}(v,\gamma') > \mathtt{W_{Ch}}(v,\gamma)$ is possible only if v executes $\mathbb{R}_{\mathtt{Win}}$ during cs, and thus Lemma 16 states that $\forall p \in \mathcal{P}(v)$ such that $\mathtt{tok}_p^\gamma \notin \{\bot, \uparrow\}$, we have $\mathtt{W_{Ch}}(p,\gamma') < \mathtt{W_{Ch}}(p,\gamma)$. ■

Theorem 3 establishes that $\mathtt{W_{Ch}}$ is pre-admissible.

Theorem 3

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs. We have $\mathtt{W_{Ch}}(D_a^{\gamma'},\gamma') \preceq \mathtt{W_{Ch}}(D_a^\gamma,\gamma)$.

Proof: Let $\mathcal{B} = v_1v_2\cdots v_k$ be a maximal branch of G_a, and let $v_1v_2\cdots v_j = D_a^{\gamma'}(\mathcal{B})$ be the projection of $D_a^{\gamma'}$ on $\mathcal{B}$. Let us consider the following cases:

- If $v_1v_2\cdots v_j$ is a branch of $D_a^\gamma(\mathcal{B})$ and $\forall i \in [1,j], \mathtt{W_{Ch}}(v_i,\gamma') = \mathtt{W_{Ch}}(v_i,\gamma)$.
  Then we have $\mathtt{W_{Ch}}(v_1,\gamma')\mathtt{W_{Ch}}(v_2,\gamma')\cdots\mathtt{W_{Ch}}(v_j,\gamma') = \mathtt{W_{Ch}}(v_1,\gamma)\mathtt{W_{Ch}}(v_2,\gamma)\cdots\mathtt{W_{Ch}}(v_j,\gamma)$ and thus $\mathtt{W_{Ch}}(D_a^{\gamma'}(\mathcal{B}),\gamma') \preceq_\alpha \mathtt{W_{Ch}}(D_a^\gamma(\mathcal{B}),\gamma)$.
- If $v_1v_2\cdots v_j$ is a branch of $D_a^\gamma(\mathcal{B})$ and $\exists i \in [1,j] : \mathtt{W_{Ch}}(v_i,\gamma') \neq \mathtt{W_{Ch}}(v_i,\gamma)$.
  Let us consider the smallest i such that $\mathtt{W_{Ch}}(v_i,\gamma') \neq \mathtt{W_{Ch}}(v_i,\gamma)$. According to Lemma 17, $\mathtt{W_{Ch}}(v_i,\gamma') < \mathtt{W_{Ch}}(v_i,\gamma)$, and as a consequence, $\mathtt{W_{Ch}}(v_1,\gamma')\mathtt{W_{Ch}}(v_2,\gamma')\cdots\mathtt{W_{Ch}}(v_j,\gamma') \prec \mathtt{W_{Ch}}(v_1,\gamma)\mathtt{W_{Ch}}(v_2,\gamma)\cdots\mathtt{W_{Ch}}(v_j,\gamma$
  Thus, we have $\mathtt{W_{Ch}}(D_a^{\gamma'}(\mathcal{B}),\gamma') \prec_\alpha \mathtt{W_{Ch}}(D_a^\gamma(\mathcal{B}),\gamma)$.
- If $v_1v_2\cdots v_j$ is not a branch of D_a^γ.
  Let us consider $v_1v_2\cdots v_l = D_a^\gamma(\mathcal{B})$ the projection of D_a^γ on $\mathcal{B}$. Since $v_1v_2\cdots v_j$ is not a branch of D_a^γ we have $1 \leq l < j$, and thus $l+1 \leq j$. Remark first that since (v_l, v_{l+1}) is an edge in $D_a^{\gamma'}$, we have $\mathtt{tok}_{v_l}^{\gamma'} \notin \{\uparrow, \bot\}$, and thus $\mathtt{tok}_{v_l}^\gamma \neq \uparrow$. But $v_{l+1} \notin D_a^\gamma$, so $\mathtt{tok}_{v_{l+1}}^\gamma = \bot$, and since $v_{l+1} \in D_a^{\gamma'}$, $\mathtt{tok}_{v_{l+1}}^{\gamma'} \neq \bot$ so v_{l+1} executes $\mathbb{R}_{\mathtt{Win}}$ during cs. Thus, Lemma 16 applies, and as a consequence $\mathtt{W_{Ch}}(v_l,\gamma') < \mathtt{W_{Ch}}(v_l,\gamma)$.
  Let us consider the smallest $i \leq l$ such that $\mathtt{W_{Ch}}(v_i,\gamma') \neq \mathtt{W_{Ch}}(v_i,\gamma)$. According to Lemma 17, $\mathtt{W_{Ch}}(v_i,\gamma') < \mathtt{W_{Ch}}(v_i,\gamma)$, and as a consequence,

  $$\mathtt{W_{Ch}}(v_1,\gamma')\mathtt{W_{Ch}}(v_2,\gamma')\cdots\mathtt{W_{Ch}}(v_j,\gamma') \prec \mathtt{W_{Ch}}(v_1,\gamma)\mathtt{W_{Ch}}(v_2,\gamma)\cdots\mathtt{W_{Ch}}(v_l,\gamma).$$

  Thus, we have $\mathtt{W_{Ch}}(D_a^{\gamma'}(\mathcal{B}),\gamma') \prec_\alpha \mathtt{W_{Ch}}(D_a^\gamma(\mathcal{B}),\gamma)$.

We prove that for any maximal branch $v_1v_2\cdots v_k$ of G_t, $\mathtt{W_{Ch}}(D_a^{\gamma'}(\mathcal{B}),\gamma') \preceq_\alpha \mathtt{W_{Ch}}(D_a^\gamma(\mathcal{B}),\gamma)$. By definition, we have $\mathtt{W_{Ch}}(D_a^{\gamma'},\gamma') \preceq \mathtt{W_{Ch}}(D_a^\gamma,\gamma)$. ■

Theorem 4 and Lemma 18 prove that under certain circumstances, we are certain that the weight of one $\mathtt{CD}^o$ decreases. The first theorem focuses on the application of rules $\mathbb{R}_{\mathtt{Win}}$ and $\mathbb{R}_{\mathtt{FakeWin}}$, and will be useful to prove that $\mathtt{W}$ is an admissible weight function, but also to prove that the other, longer, prefixes of $\mathtt{W}$ are pre-admissible. Indeed, if in any computing step, one node executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$, we do not need to look beyond the first component to be assured that the weight does not increase, since $\preceq^k$ is a lexicographic order. The second lemma focuses

on the length of the branches of a $\texttt{CD}^o$, and is useful for us to write proofs in the following. Just as we explained before, we can now suppose in the following proofs of pre-admissibility that branches of $\texttt{CD}^o$'s never increase, since it would make the first component of $\texttt{W}$ decrease.

Theorem 4

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs. If $\exists v \in \mathbb{B}(D_a^\gamma)$ such that v executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, or $\mathbb{R}_{\texttt{ReplayUp}}$ during cs, we have $\texttt{W}_{\texttt{Ch}}(D_a^{\gamma'}, \gamma') \prec \texttt{W}_{\texttt{Ch}}(D_a^\gamma, \gamma)$.

Proof: Let us consider one node $v \in \mathbb{B}(D_a^\gamma)$ that executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, or $\mathbb{R}_{\texttt{ReplayUp}}$ during cs. By definition, $\exists p \in \mathcal{P}(v) : p \in D_a^\gamma \wedge \texttt{tok}_v^\gamma \neq \uparrow$. Let us consider $\mathcal{B} = v_1 v_2 \cdots v_k$ a branch of D_a^γ such that $v_k = p$. According to Lemma 10, $\mathcal{B}$ is a branch of $D_a^{\gamma'}$.

According to Lemma 16, $\texttt{W}_{\texttt{Ch}}(p, \gamma') < \texttt{W}_{\texttt{Ch}}(p, \gamma)$, and according to Theorem 3, $\texttt{W}_{\texttt{Ch}}(D_a^{\gamma'}(\mathcal{B}), \gamma') \preceq_\alpha \texttt{W}_{\texttt{Ch}}(D_a^\gamma(\mathcal{B}), \gamma)$. Consequently, $\texttt{W}_{\texttt{Ch}}(D_a^{\gamma'}(\mathcal{B}), \gamma') \prec_\alpha \texttt{W}_{\texttt{Ch}}(D_a^\gamma(\mathcal{B}), \gamma)$, and thus according to Theorem 3, $\texttt{W}_{\texttt{Ch}}(D_a^{\gamma'}, \gamma') \prec \texttt{W}_{\texttt{Ch}}(D_a^\gamma, \gamma)$. ■

Lemma 18

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs. If there exists $\mathcal{B} = v_1 v_2 \cdots v_k$ a branch of G_a such that $D_a^{\gamma'}(\mathcal{B})$ is longer than $D_a^\gamma(\mathcal{B})$, then $\texttt{W}_{\texttt{Ch}}(D_a^{\gamma'}, \gamma') \prec \texttt{W}_{\texttt{Ch}}(D_a^\gamma, \gamma)$.

Proof: Let us consider $v_1 v_2 \cdots v_j = D_a^\gamma(\mathcal{B})$. Let us prove that $v = v_{j+1}$ satisfies the conditions of Theorem 4.

Remark first that $\texttt{tok}_{v_i}^\gamma \neq \uparrow$. Indeed, if $\texttt{tok}_{v_i}^\gamma = \uparrow$ then $\texttt{tok}_{v_i}^{\gamma'} \in \{\uparrow, \bot\}$, which is contradictory since $v \in \mathcal{C}_{D_a^{\gamma'}}(v_i)$. Then, we have $\texttt{tok}_v^\gamma = \bot$ which implies $v \in \mathbb{B}(D_a^\gamma)$, otherwise we would have $v \in \mathcal{C}_{D_a^\gamma}(v_i)$. But since $v \in D_a^{\gamma'}$, we have $\texttt{tok}_v^{\gamma'} \neq \bot$, which is possible only if v executes $\mathbb{R}_{\texttt{Win}}$ during cs.

We can apply Theorem 4 to our situation, and consequently $\texttt{W}_{\texttt{Ch}}(D_a^{\gamma'}, \gamma') \prec \texttt{W}_{\texttt{Ch}}(D_a^\gamma, \gamma)$. ■

C.2.5 Second component of W: $\texttt{W}_{\texttt{Er}}$ errors descending from v

Explanations One major quantity in our algorithm that should decrease during an execution is the number of nodes v such that $\texttt{Er}(v)$. Indeed, we expect that after some convergence time, no node will execute $\mathbb{E}_{\texttt{TrustChild}}$, and that our $\texttt{CD}^o$ will have the expected shape. Although that last sentence will be proven true in the following, the number of nodes v such that $\texttt{Er}(v)$ might occasionally increases. Indeed when a node executes $\mathbb{E}_{\texttt{TrustChild}}$, it might happen that it creates an error in some of its parents. For example, if node v has p as a parent, and $\texttt{tok}_v^\gamma = \bigstar, \texttt{tok}_p^\gamma = \downdownarrows$. We have $\neg\texttt{Er}^\gamma(p)$, but if in $\gamma \to \gamma', v$ executes $\mathbb{E}_{\texttt{TrustChild}}$, then $\texttt{tok}_v^{\gamma'} = \downarrow$ and thus $\texttt{Er}^{\gamma'}(p)$. In the described case, the number of nodes in error does not increase, but if node v has several parents p with $\texttt{tok}_p^\gamma \in \{\downdownarrows, \circlearrowleft\}$ then that number will actually increase.

One first idea to prevent this from happening is that each node counts the number of descendants it has that are in error. This almost works, but not totally. Indeed, suppose v has two parents p, that both have the same parent a, and we have $\texttt{tok}_v^\gamma = \bigstar$, $\texttt{tok}_p^\gamma = \downdownarrows$, $\texttt{tok}_a^\gamma = \downarrow$. In γ, a only has one descendant in error, but has two once v executes $\mathbb{E}_{\texttt{TrustChild}}$. The fact that errors can split through several branches forces us to design $\texttt{W}_{\texttt{Er}}$ such that it counts a descendant in error as many times as there exists paths to that node.

Fortunately, we can stop the count in our descendants as soon as we reach a node v such that $\texttt{tok}_v \in \{\bot, \uparrow, \downarrow\}$. Indeed, such nodes cannot execute $\mathbb{E}_{\texttt{TrustChild}}$, which implies that errors cannot ride up through such nodes.

There exists another situation that increases the number of nodes in error. Suppose one node v executes $\mathbb{R}_{\texttt{Win}}$ during $\gamma \to \gamma'$. We have $\texttt{W}_{\texttt{Er}}(v,\gamma) = 0$ by definition. Yet, it might happen that one child u of v is such that $\texttt{tok}_u^{\gamma'} \neq \bot$. It might even happen that $\texttt{W}_{\texttt{Er}}(u,\gamma') > 0$, with an arbitrary value. In such a situation, the weight of v, and thus $\texttt{W}_{\texttt{Er}}(D_a^\gamma)$ can increase arbitrarily. Hopefully, we proved in the previous section that at the same moment, $\texttt{W}_{\texttt{Ch}}(D_a^\gamma)$ decreases, and thus no problem is raised.

Definition According to what precedes, we define $\texttt{W}_{\texttt{Er}}$ the second component of our weight function by 0 on nodes v such that $\texttt{tok}_v \in \{\downarrow, \uparrow, \bot\}$, and by the number of nodes in error they can see within range, themselves included, otherwise.

$$\texttt{W}_{\texttt{Er}}(v,\gamma) = \begin{cases} 0 & \text{if } \texttt{tok}_v^\gamma \in \{\downarrow, \uparrow, \bot\} \\ \sum_{u \in \mathcal{C}(v)} \texttt{W}_{\texttt{Er}}(u,\gamma) + 1 & \text{if } \texttt{Er}^\gamma(v) \\ \sum_{u \in \mathcal{C}(v)} \texttt{W}_{\texttt{Er}}(u,\gamma) & \text{otherwise} \end{cases} \tag{38}$$

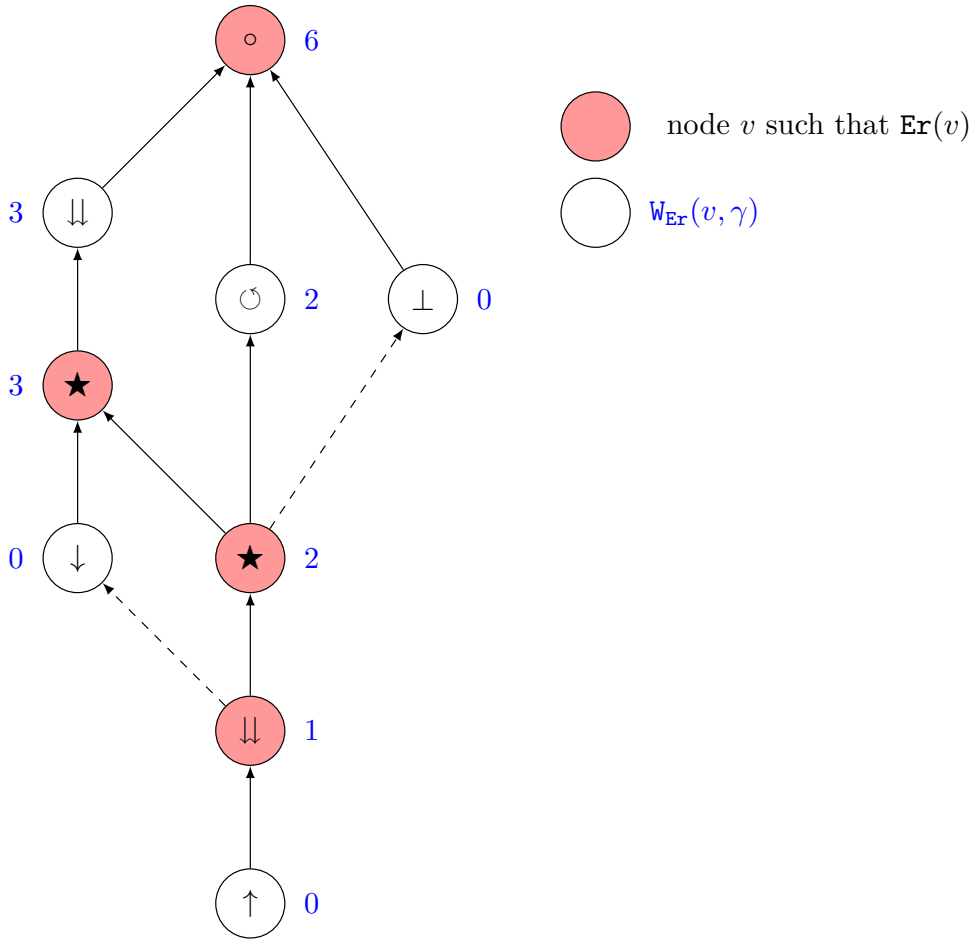

Figure 25: Example of the definition of $\texttt{W}_{\texttt{Er}}$ on a $\texttt{D}^o$.

Proofs Lemma 19 describes under which conditions one node that is not in error before a computing step can become in error after that computing step. It states that either it makes $\texttt{W}_{\texttt{Ch}}$ decrease, either it comes from the fact that one child of that node executed $\mathbb{E}_{\texttt{TrustChild}}$,

which does not lead to any increase of $\mathtt{W_{Er}}$. This lemma is the main argument to establish that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}})$ is a pre-admissible weight function.

Lemma 19

Let $v \in V$ be a node and let $cs = \gamma \to \gamma'$ be a computing step. If $\neg\mathtt{Er}^{\gamma}(v)$ and $\mathtt{Er}^{\gamma'}(v)$, then $\mathtt{W_{Ch}}(v,\gamma') < \mathtt{W_{Ch}}(v,\gamma)$ or v executes $\mathbb{R}_{\mathtt{Win}}$ during cs or $\exists u \in \mathcal{C}(v)$ such that u executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs.

Proof: We prove the equivalent statement: if $\neg\mathtt{Er}^{\gamma}(v)$ then $\neg\mathtt{Er}^{\gamma'}(v)$ or $\mathtt{W_{Ch}}(v,\gamma') < \mathtt{W_{Ch}}(v,\gamma)$ or, during cs, v executes $\mathbb{R}_{\mathtt{Win}}$ or $\exists u \in \mathcal{C}(v)$ such that u executes $\mathbb{E}_{\mathtt{TrustChild}}$.

Since $\neg\mathtt{Er}^{\gamma}(v)$, only few cases must be considered.

1. If $\mathtt{tok}_v^{\gamma} = \bot$ then either v executes $\mathbb{R}_{\mathtt{Win}}$ during cs, either it does not, but then $\mathtt{tok}_v^{\gamma'} = \bot$ and thus $\neg\mathtt{Er}^{\gamma'}(v)$.
2. If $\mathtt{tok}_v^{\gamma} = \uparrow$ then $\mathtt{tok}_v^{\gamma'} \in \{\uparrow, \bot\}$ and thus $\neg\mathtt{Er}^{\gamma'}(v)$.
3. If $\mathtt{tok}_v^{\gamma} = \downarrow$ then if v does not execute $\mathbb{R}_{\mathtt{Receive}}$ during cs, $\mathtt{tok}_v^{\gamma'} = \downarrow$ and thus $\neg\mathtt{Er}^{\gamma'}(v)$. Otherwise, v executes $\mathbb{R}_{\mathtt{Receive}}$ during cs and then $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} \in \{\bot, \uparrow\}$. Either one child u of v executes $\mathbb{R}_{\mathtt{Win}}$, and then $\mathtt{W_{Ch}}(v,\gamma') < \mathtt{W_{Ch}}(v,\gamma)$ according to Lemma 16, either no one does, but then $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} \in \{\bot, \uparrow\}$, and since $\mathtt{tok}_v^{\gamma'} = \circ$, we have $\neg\mathtt{Er}^{\gamma'}(v)$.
4. If $\mathtt{tok}_v^{\gamma} = \bigstar \wedge \forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} = \bot$ then either one child u of v executes $\mathbb{R}_{\mathtt{Win}}$, and then $\mathtt{W_{Ch}}(v,\gamma') < \mathtt{W_{Ch}}(v,\gamma)$ according to Lemma 16, either no one does, but then $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} = \bot$ which means that $\neg\mathtt{Er}^{\gamma'}(v)$.
5. If $\mathtt{tok}_v^{\gamma} = \bullet \wedge \forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} = \bot$ then children u of v cannot execute $\mathbb{R}_{\mathtt{Win}}$ during cs: either u has one other parent p such that $\mathtt{tok}_p^{\gamma} = \downdownarrows$, and then $\neg\mathtt{OkNeg}^{\gamma}(u)$, either it has not, and thus it cannot execute $\mathbb{R}_{\mathtt{Win}}$ either. Thus, $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} = \bot$, so $\neg\mathtt{Er}^{\gamma'}(v)$.
6. If $\mathtt{tok}_v^{\gamma} = \downdownarrows \wedge \forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} \in \{\bot, \bigstar\}$ then children u of v cannot execute $\mathbb{R}_{\mathtt{Nego}}$ since they have one parent v with $\mathtt{tok}_v^{\gamma} = \downdownarrows$. If no children u of v executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs, then $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} \in \{\bot, \bigstar\}$. Furthermore, $\mathtt{tok}_p^{\gamma'} \in \{\downdownarrows, \circlearrowleft\}$ since $\neg\mathtt{Er}^{\gamma}(v)$. Thus, if no children u of v executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs, $\neg\mathtt{Er}^{\gamma'}(v)$.
7. If $\mathtt{tok}_v^{\gamma} = \circlearrowleft \wedge \forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} \in \{\bot, \bigstar\}$ then children u of v cannot execute $\mathbb{R}_{\mathtt{Nego}}$ since they have one parent v with $\mathtt{tok}_v^{\gamma} = \circlearrowleft$. If no children u of v executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs, then $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} \in \{\bot, \bigstar\}$. Furthermore, $\mathtt{tok}_v^{\gamma'} \in \{\circlearrowleft, \downarrow\}$. Thus, if no children u of v executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs, $\neg\mathtt{Er}^{\gamma'}(v)$.
8. If $\mathtt{tok}_v^{\gamma} = \circ \wedge \forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} \in \{\bot, \uparrow\}$ then children u of v cannot execute $\mathbb{R}_{\mathtt{Win}}$ during cs: either u has one other parent p such that $\mathtt{tok}_p^{\gamma} = \downdownarrows$, and then $\neg\mathtt{OkNeg}^{\gamma}(u)$, either it has not, and thus it cannot execute $\mathbb{R}_{\mathtt{Win}}$ either. This implies that $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} \in \{\bot, \uparrow\}$ and thus $\neg\mathtt{Er}^{\gamma'}(v)$.

■

Theorem 5 establishes that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}})$ is pre-admissible.

Theorem 5

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs.

We have $(\mathtt{W_{Ch}}, \mathtt{W_{Er}})(D_a^{\gamma'}, \gamma') \preceq^2 (\mathtt{W_{Ch}}, \mathtt{W_{Er}})(D_a^{\gamma}, \gamma)$.

Proof: According to Theorem 3, we already have $\mathtt{W_{Ch}}(D_a^{\gamma'}, \gamma') \preceq \mathtt{W_{Ch}}(D_a^{\gamma}, \gamma)$. Consequently, by definition of the lexicographic order, we only have to establish that if $\mathtt{W_{Er}}(D_a^{\gamma}, \gamma) \prec \mathtt{W_{Er}}(D_a^{\gamma'}, \gamma')$ then $\mathtt{W_{Ch}}(D_a^{\gamma'}, \gamma') \prec \mathtt{W_{Ch}}(D_a^{\gamma}, \gamma)$.

Let us first remark that if there exists $\mathcal{B}$ a branch of G_a such that $D_a^{\gamma'}(\mathcal{B})$ is longer than $D_a^{\gamma}(\mathcal{B})$ then according to Lemma 18, $\mathtt{W_{Ch}}(D_a^{\gamma'}, \gamma') \prec \mathtt{W_{Ch}}(D_a^{\gamma}, \gamma)$.

We suppose now that for any branch $\mathcal{B}$ of G_a, $D_a^{\gamma'}(\mathcal{B})$ is not longer than $D_a^{\gamma}(\mathcal{B})$. Let us establish that $\forall v \in D_a^{\gamma'}, \mathtt{W_{Er}}(v,\gamma') \leq \mathtt{W_{Er}}(v,\gamma)$, or $\mathtt{W_{Ch}}(D_a^{\gamma'},\gamma') \prec \mathtt{W_{Ch}}(D_a^{\gamma},\gamma)$. Since the branches of $D_a^{\gamma'}$ are included in the branches of D_a^{γ}, this becomes $\mathtt{W_{Er}}(D_a^{\gamma'},\gamma') \preceq \mathtt{W_{Er}}(D_a^{\gamma},\gamma)$ or $\mathtt{W_{Ch}}(D_a^{\gamma'},\gamma') \prec \mathtt{W_{Ch}}(D_a^{\gamma},\gamma)$, and thus this actually proves the theorem.

Let us prove that if $\exists v \in D_a^{\gamma'} : \mathtt{W_{Er}}(v,\gamma') > \mathtt{W_{Er}}(v,\gamma)$ then $\mathtt{W_{Ch}}(D_a^{\gamma'},\gamma') \prec \mathtt{W_{Ch}}(D_a^{\gamma},\gamma)$, and let us consider one such v with highest depth. Let us first prove that $\mathtt{tok}_v^{\gamma} \notin \{\bot,\uparrow,\downarrow\}$. Since $v \in D_a^{\gamma}$, $\mathtt{tok}_v^{\gamma} \neq \bot$. If $\mathtt{tok}_v^{\gamma} = \uparrow$ then $\mathtt{tok}_v^{\gamma'} \in \{\uparrow,\bot\}$, and thus $\mathtt{W_{Er}}(v,\gamma') = 0$ which is contradictory with our hypothesis, so $\mathtt{tok}_v^{\gamma} \neq \uparrow$. Finally, if $\mathtt{tok}_v^{\gamma} = \downarrow$ then $\mathtt{W_{Er}}(v,\gamma') > 0$ is possible only if v executes $\mathbb{R}_{\mathtt{Receive}}$ during cs, which implies that $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma} \in \{\bot,\uparrow\}$. None of these children executes $\mathbb{R}_{\mathtt{Win}}$ during cs, because if one u does then the edge (v,u) belongs to $D_a^{\gamma'}$ (because $\mathtt{tok}_v^{\gamma'} = \circ$) while it does not belong to D_a^{γ}, which is contradictory with our hypothesis on branches. Consequently, $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma'} \in \{\bot,\uparrow\}$, and thus $\neg\mathtt{Er}^{\gamma'}(v)$, and $\mathtt{W_{Er}}(v,\gamma') = 0$. Thus, we can indeed assume that $\mathtt{tok}_v^{\gamma} \notin \{\bot,\uparrow,\downarrow\}$.

Since all branches of $D_a^{\gamma'}$ are also branches of D_a^{γ}, and since v is one deepest node of $D_a^{\gamma'}$ such that $\mathtt{W_{Er}}(v,\gamma') > \mathtt{W_{Er}}(v,\gamma)$, we have $\forall u \in \mathcal{C}(v), \mathtt{W_{Er}}(u,\gamma') \leq \mathtt{W_{Er}}(u,\gamma)$. As a consequence, $\mathtt{W_{Er}}(v,\gamma') > \mathtt{W_{Er}}(v,\gamma)$ becomes possible only if $\neg\mathtt{Er}^{\gamma}(v)$ and $\mathtt{Er}^{\gamma'}(v)$.

According to Lemma 19, this is possible only if during cs, v executes $\mathbb{R}_{\mathtt{Win}}$, which cannot happen since $\mathtt{tok}_v^{\gamma} \neq \bot$, or if $\mathtt{W_{Ch}}(v,\gamma') < \mathtt{W_{Ch}}(v,\gamma)$ which implies that $\mathtt{W_{Ch}}(D_a^{\gamma'},\gamma') \prec \mathtt{W_{Ch}}(D_a^{\gamma},\gamma)$, or if $\exists u \in \mathcal{C}(v)$ such that u executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs. In that last case, we have $\mathtt{Er}^{\gamma}(u)$ so $\mathtt{W_{Er}}(u,\gamma) \geq 1$, and $\mathtt{tok}_u^{\gamma'} = \downarrow$ and so $\mathtt{W_{Er}}(u,\gamma') = 0$. Since we took v one deepest node such that $\mathtt{W_{Er}}(v,\gamma') > \mathtt{W_{Er}}(v,\gamma)$, we have $\forall u \in \mathcal{C}(v), \mathtt{W_{Er}}(u,\gamma') \leq \mathtt{W_{Er}}(u,\gamma)$, so, by considering the child that executes $\mathbb{E}_{\mathtt{TrustChild}}$, we have $\sum_{u\in\mathcal{C}(v)} \mathtt{W_{Er}}(u,\gamma') < \sum_{u\in\mathcal{C}(v)} \mathtt{W_{Er}}(u,\gamma)$, and thus $\mathtt{W_{Er}}(v,\gamma') \leq \mathtt{W_{Er}}(v,\gamma)$. ■

Theorem 6 proves that under certain circumstances, we are certain that the weight of one $\mathtt{CD}^o$ decreases. It will be useful to prove that $\mathtt{W}$ is an admissible function, but also to prove that the other, longer, prefixes of $\mathtt{W}$ are pre-admissible. Indeed, if in any computing step one node executes $\mathbb{E}_{\mathtt{TrustChild}}$ then we do not need to look beyond the second component to be assured that the weight does not increase, since $\preceq^k$ is a lexicographic order.

Theorem 6

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs. If $\exists v \in D_a^{\gamma}$ such that v executes $\mathbb{E}_{\mathtt{TrustChild}}$ during cs, then $(\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma'},\gamma') \prec^2 (\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma},\gamma)$.

Proof: Let us consider one such node $v \in D_a^{\gamma}$, and one branch $\mathcal{B}$ of G_a such that $v \in D_a^{\gamma}(\mathcal{B})$. If $v \notin D_a^{\gamma'}(\mathcal{B})$, then according to Lemma 18, we have $(\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma'},\gamma') \prec^2 (\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma},\gamma)$.

Else, remark that $\mathtt{W_{Er}}(v,\gamma) > 1$ since $\mathtt{Er}^{\gamma}(v)$, and $\mathtt{W_{Er}}(v,\gamma') = 0$ since $\mathtt{tok}_v^{\gamma} = \downarrow$. Thus, $\mathtt{W_{Er}}(v,\gamma') < \mathtt{W_{Er}}(v,\gamma)$, and so $\mathtt{W_{Er}}(D_a^{\gamma'}(\mathcal{B}),\gamma') \prec_{\alpha} \mathtt{W_{Er}}(D_a^{\gamma}(\mathcal{B}),\gamma)$ since $v \in D_a^{\gamma'}(\mathcal{B})$. Since $(\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma'},\gamma') \preceq^2 (\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma},\gamma)$, we conclude $(\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma'},\gamma') \prec^2 (\mathtt{W_{Ch}},\mathtt{W_{Er}})(D_a^{\gamma},\gamma)$. ■

C.2.6 Third component of $\mathtt{W}$: $\mathtt{W_{Circ}}$, circulation of variable tok

Explanations Now that we have considered the cases where nodes execute $\mathbb{E}_{\mathtt{TrustChild}}$, we can consider that variable $\mathtt{tok}$ is updated in accordance with Figure 3. The third component of the weight function should treat all the transitions of variable $\mathtt{tok}$ depicted in this diagram. We could think at first sight that since the diagram is circular, we will have difficulty defining a weight function that decreases along the diagram. Yet, if the algorithm behaves correctly, when a node v is updated from $\bullet$ to $\downarrow\downarrow$, one child of v is supposed to execute $\mathbb{R}_{\mathtt{Win}}$ in the next computing steps, and then $\mathtt{W_{Ch}}$ will decrease on the observed $\mathtt{CD}^o$. Since $\mathtt{W_{Ch}}$ has higher priority in the lexicographic order than $\mathtt{W_{Circ}}$, we are going to design $\mathtt{W_{Circ}}$ so that it increases exactly

when $\mathtt{W_{Ch}}$ decreases. Therefore, we can cycle as many times as necessary through the transition diagram of Figure 3 and having the 3-tuple $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})$ decrease at each step.

To do this, we must associate a different weight to a node v with $\mathtt{tok}_v = \downdownarrows$ depending on whether it still has one child that is going to execute $\mathbb{R}_{\mathtt{Win}}$ (or $\mathbb{R}_{\mathtt{FakeWin}}$) in the next computing steps. More precisely, if there still exists one child u of v that belongs to $\mathtt{Players}(v)$, we associate a lower weight to v that if such child u does not exist. When v executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$, $\mathtt{W_{Circ}}$ increases on v, but $\mathtt{W_{Ch}}$ decreases at the same time.

But since we work in the framework of self-stabilization, it might happen that registers are incorrectly initialized, and that when node v executes $\mathbb{R}_{\mathtt{Give}}$, and updates $\mathtt{tok}_v$ from $\bullet$ to $\downdownarrows$, v immediately has no children susceptible to execute $\mathbb{R}_{\mathtt{Win}}$. Indeed, if in the initial configuration, $\mathtt{id}_v$ is inconsistent with the $\mathtt{id}$ of its children, it might happen that after some computations, all the children u of v such that $\mathtt{play}_u = \mathtt{P}$ eventually lose the negotiation. In such a situation, after v executes $\mathbb{R}_{\mathtt{Give}}$, we arrive in a configuration in which $\mathtt{W_{Circ}}(v, \gamma')$ is already at its high value. But this high value is precisely designed to be the highest assigned value of the diagram. To overcome that issue, we add one possible value for $\mathtt{W_{Circ}}(v, \gamma)$ when $\mathtt{tok}_v^{\gamma} = \bullet$, higher than the high value for nodes with $\mathtt{tok}_v^{\gamma} = \downdownarrows$. This value is taken by $\mathtt{W_{Circ}}(v, \gamma)$ only if we detect that after few computations, we fall into a situation where no children of v can execute $\mathbb{R}_{\mathtt{Win}}$. It is crucial that this situation cannot happen consecutively to an execution of $\mathbb{R}_{\mathtt{ReNego}}$, otherwise we should also design two possible outcomes for the weight of some node v with $\mathtt{tok}_v = \circ$. The predicate $\mathtt{FullRound}$ achieves that detection for node v such that $\mathtt{tok}_v = \bullet$.

For the other values that $\mathtt{tok}_v$ might take, we only associate integers that decrease along the classical scheme depicted in Figure 3. The definition of $\mathtt{W_{Circ}}$ is summarized in Figure 26

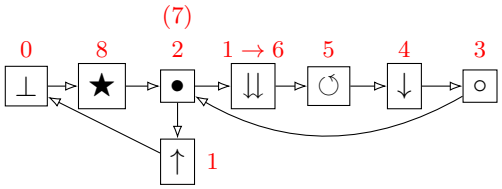

Figure 26: Value of $\mathtt{W_{Circ}}$ depending on the variable $\mathtt{tok}_v$.

Definitions The weight of one node v such that $\mathtt{tok}_v = \downdownarrows$ depends on if it has children susceptible to take the token. If it has, then we set its weight to one low value, so that when the child takes the token, $\mathtt{W_{Ch}}$ decreases on v at the same time that we make $\mathtt{W_{Circ}}$ increases on v. If it has not, then it must be a high value so that the weight of v can decrease through the transitions of Figure 26.

$$\mathtt{W}^{\downdownarrows}_{\mathtt{Circ}}(v, \gamma) = \begin{cases} 6 & \text{if } \mathtt{Players}^{\gamma}(v) = \emptyset \\ 1 & \text{if } \mathtt{Players}^{\gamma}(v) \neq \emptyset \end{cases} \tag{39}$$

Let us now define the weight of one node v such that $\mathtt{tok}_v = \bullet$. The weight of such a node can also be taken between two values that are 2 and 7, so that it can easily be bigger than $\mathtt{W}^{\downdownarrows}_{\mathtt{Circ}}$. We must design $\mathtt{W}^{\bullet}_{\mathtt{Circ}}$ such that, if after v executes $\mathbb{R}_{\mathtt{Give}}$ we have $\mathtt{W}^{\downdownarrows}_{\mathtt{Circ}}(v, \gamma') = 6$ then we have $\mathtt{W}^{\bullet}_{\mathtt{Circ}}(v, \gamma) = 7$ just before, otherwise the weight would increase. But we also require that we never have $\mathtt{W}^{\bullet}_{\mathtt{Circ}}(v, \gamma) = 2 \wedge \mathtt{W}^{\bullet}_{\mathtt{Circ}}(v, \gamma') = 7$. In other words, if $\mathtt{W}^{\downdownarrows}_{\mathtt{Circ}}$ is set to 6 immediately after the execution of $\mathbb{R}_{\mathtt{Give}}$, it must be detected at the very beginning of the execution (or as soon as $\mathtt{tok}_v$ is set to $\bullet$).

The situation described just above happens if before v executes $\mathbb{R}_{\mathtt{Give}}$, all the nodes $u \in \mathtt{Players}^{\gamma}(v)$ execute $\mathbb{R}_{\mathtt{Lose}}$. In such a situation, no children u of v wins the negotiation, and

v executes a full cycle of the cycle of Figure 26, which indeed forces us to attribute 7 to the weight of v. Hopefully, the design of our algorithm guarantees that this can only happen if the values of $\mathtt{id}_v$ and/or $\mathtt{id}_u$ are inconsistent. In other words, as soon as v executes one action, we guarantee that this situation cannot occur anymore, this will be proven in Lemma 23. Thus, we only have to deduce if $u \in \mathtt{Players}(v)$ will execute $\mathbb{R}_{\mathtt{Lose}}$ before any action of v, from the values of $\mathtt{id}_v$ and $\mathtt{id}_u$. This strongly depends on the value of $\mathtt{b}_v^\gamma$.

If $\mathtt{b}_v^\gamma \neq \bot$ then v announces one value that defines the partial winners of that turn of the negotiation. It the configuration is inconsistent, it is very possible that no children $u \in \mathtt{Players}^\gamma(v)$ has produced, neither will produce that answer. All of the nodes that have announced a different value than $\mathtt{b}_v^\gamma$, or that will announce a different value than $\mathtt{b}_v^\gamma$ will execute $\mathbb{R}_{\mathtt{Lose}}$, immediately for the first ones, and after one execution of $\mathbb{R}_{\mathtt{NewBit}}$ for the second ones

$$\mathtt{Losers}^\gamma_{\neg\bot}(v) = \{u \in \mathtt{Players}^\gamma(v), \left\{ \begin{array}{l} \mathtt{ph}_u^\gamma = \mathtt{ph}_v^\gamma \\ \mathtt{b}_u^\gamma \neq \mathtt{b}_v^\gamma \end{array} \right. \vee \left\{ \begin{array}{l} \mathtt{ph}_u^\gamma \neq \mathtt{ph}_v^\gamma \\ \mathtt{Bit}_u(\mathtt{ph}_v^\gamma) \neq \mathtt{b}_v^\gamma \end{array} \right. \} \tag{40}$$

If $\mathtt{b}_v^\gamma = \bot$ then v is waiting for all of its children $u \in \mathtt{Players}^\gamma(v)$ to announce $\mathtt{Bit}_u(\mathtt{ph}_v^\gamma)$. Therefore, no children $u \in \mathtt{Players}^\gamma(v)$ can execute $\mathbb{R}_{\mathtt{Lose}}$ before an action of v. We define the set $\mathtt{Losers}(v)$ that depends on the value of $\mathtt{b}_v$.

$$\mathtt{Losers}^\gamma(v) = \left\{ \begin{array}{ll} \mathtt{Losers}^\gamma_{\neg\bot}(v) & \text{if } \mathtt{b}_v^\gamma \neq \bot \\ \emptyset & \text{if } \mathtt{b}_v^\gamma = \bot \end{array} \right. \tag{41}$$

We are now tempted to say that $\mathtt{W}^\gamma_{\mathtt{Circ}}(v) = 7$ if and only if $\mathtt{Losers}^\gamma(v) = \mathtt{Players}^\gamma(v)$. Yet, one subtlety must be treated before. Indeed, when all the children u of v have received the token, then u will not execute $\mathbb{R}_{\mathtt{Give}}$ but $\mathbb{R}_{\mathtt{OfferUp}}$, and thus we do not have to reason based on the constraint of $\mathtt{W}^{\downdownarrows}_{\mathtt{Circ}}$. Moreover, in that situation, we must not attribute a weight of 7 to v since it would be an increase when v executes $\mathbb{R}_{\mathtt{ReNego}}$, and updates $\mathtt{tok}_v$ from $\circ$ to $\bullet$. This situation cannot happen as long as there remains players, because after the execution of $\mathtt{StartNego}(v)$, we have $\mathtt{Losers}(v) = \emptyset$. The only case to consider is therefore when no node children of v is susceptible to receive the token, and in that situation we force $\mathtt{W}^\bullet_{\mathtt{Circ}}(v, \gamma)$ to be 2.

In order to make the proof easier to approach, we first define the set $\mathtt{Candidates}(v)$ that contains all the nodes that still have to negotiate and win the token before the circulation from v is over. This set reminds one of the different cases depicted in Figure 23.

$$\mathtt{Candidates}^\gamma(v) = \{u \in \mathcal{C}_{\mathtt{neq}}{}^\gamma(v) : \mathtt{tok}_u^\gamma = \bot \wedge \mathtt{play}_u^\gamma \in \{\mathtt{P}, \mathtt{L}\}\} \tag{42}$$

We can now define the main predicate if that section, $\mathtt{FullRound}$, which decides whether v has to execute one full cycle before any of its children receives the token.

$$\mathtt{FullRound}^\gamma(v) \equiv (\mathtt{Losers}^\gamma(v) = \mathtt{Players}^\gamma(v)) \wedge (\mathtt{Candidates}^\gamma(v) \neq \emptyset) \tag{43}$$

We deduce function $\mathtt{W}^\bullet_{\mathtt{Circ}}$ that allows us to define $\mathtt{W}_{\mathtt{Circ}}$.

$$\mathtt{W}^\bullet_{\mathtt{Circ}}(v, \gamma) = \left\{ \begin{array}{ll} 7 & \text{if } \mathtt{FullRound}^\gamma(v) \\ 2 & \text{otherwise} \end{array} \right. \tag{44}$$

$$\mathtt{W}_{\mathtt{Circ}}(v,\gamma) = \begin{cases} 0 & \text{if } \mathtt{tok}_v^\gamma = \bot \\ 8 & \text{if } \mathtt{tok}_v^\gamma = \bigstar \\ \mathtt{W}^{\bullet}_{\mathtt{Circ}}(v,\gamma) & \text{if } \mathtt{tok}_v^\gamma = \bullet \\ \mathtt{W}^{\downdownarrows}_{\mathtt{Circ}}(v,\gamma) & \text{if } \mathtt{tok}_v^\gamma = \downdownarrows \\ 5 & \text{if } \mathtt{tok}_v^\gamma = \circlearrowleft \\ 4 & \text{if } \mathtt{tok}_v^\gamma = \downarrow \\ 3 & \text{if } \mathtt{tok}_v^\gamma = \circ \\ 1 & \text{if } \mathtt{tok}_v^\gamma = \uparrow \end{cases} \tag{45}$$

Proofs Lemma 20 establishes that actions that make the negotiation progress are non simultaneous on the parent and on its children.

Lemma 20

Let γ be a configuration and let $v \in V$ be a node such that $\mathtt{Synch}^\gamma(v) \wedge (\mathtt{PhComplete}^\gamma(v) \vee \mathtt{NextPlay}^\gamma(v))$.

Then $\forall u \in \mathtt{Players}^\gamma(v), \neg\mathtt{LosePar}^\gamma(u,v) \wedge \neg\mathtt{AnswerPar}^\gamma(u,v)$

Proof: Since we are only interested in properties at γ, no confusion can be made, and thus we do not precise the configuration in which the predicates and sets are evaluated in that proof.

By definition of $\mathtt{Synch}(v)$, we have $\mathtt{ph}_u = \mathtt{ph}_v$. Consequently, we have:

$$\mathtt{LosePar}(u,v) \equiv \mathtt{b}_v \neq \mathtt{b}_u \wedge \mathtt{b}_v \neq \bot, \text{ and}$$

$$\mathtt{AnswerPar}(u,v) \equiv \mathtt{ph}_v = 1 \wedge \mathtt{b}_v = \bot \wedge \mathtt{b}_u = \bot.$$

Let us first suppose that $\mathtt{PhComplete}(v)$. Then by definition, $\mathtt{b}_u = \mathtt{b}_v$, and thus $\neg\mathtt{LosePar}(u,v)$. Furthermore, we also have $\mathtt{ph}_v \neq 1 \vee \mathtt{b}_v \neq \bot$, which can be rewritten $\neg(\mathtt{ph}_v = 1 \wedge \mathtt{b}_v = \bot)$ and thus $\neg\mathtt{AnswerPar}(u,v)$.

Let us now suppose that $\mathtt{NextPlay}(v)$. We deduce that $\mathtt{b}_v = \bot$, and thus $\neg\mathtt{LosePar}(u,v)$. Furthermore, we have $\mathtt{AnswerPar}(u,v)$ only if $\mathtt{ph}_v = 1 \wedge \mathtt{b}_u = \bot$. But if $\mathtt{ph}_v = 1$ then $\mathtt{NextPlay}(v)$ requires that $\mathtt{b}_u \neq \bot$, which is contradictory. Thus, $\neg\mathtt{AnswerPar}(u,v)$. ■

Lemmas 21 and 22 prove that the definition of predicate `Losers` is adapted to the algorithm. We prove that all the nodes that execute $\mathbb{R}_{\mathtt{Lose}}$ were actually counted in the set `Losers`, and that execution of $\mathbb{R}_{\mathtt{NewBit}}$ does not bring any new node in the set `Losers`.

Lemma 21

Let $cs = \gamma \to \gamma'$ be a computing step, and let $v \in V$ be a node such that $\mathtt{tok}_v^\gamma = \bullet$.

$\forall u \in \mathtt{Players}^\gamma(v)$, *if u executes $\mathbb{R}_{\mathtt{Lose}}$ during cs, then* $u \in \mathtt{Losers}^\gamma(v)$

Proof: Let u be one such node. Since u executes $\mathbb{R}_{\mathtt{Lose}}$ during cs, $\mathtt{OkNeg}^\gamma(u) = \mathtt{true}$ which means that v is the only parent of u such that $\mathtt{tok}_v = \bullet$.

Thus, we have $\mathtt{ph}_v^\gamma = \mathtt{ph}_u^\gamma \wedge \mathtt{b}_v^\gamma \neq \mathtt{b}_u^\gamma \wedge \mathtt{b}_v^\gamma \neq \bot$. Then $u \in \mathtt{Losers}^\gamma_{\neg\bot}(v) = \mathtt{Losers}^\gamma(v)$. ■

Lemma 22

Let $cs = \gamma \to \gamma'$ be a computing step, and let $v \in V$ be a node such that $\mathtt{tok}_v^\gamma = \bullet$ *and such that v is not activated during cs.*

Let $u \in \mathtt{Players}^\gamma(v)$ be a node that executes $\mathbb{R}_{\mathtt{NewBit}}$ during cs. If $u \in \mathtt{Losers}^{\gamma'}(v)$ then $u \in \mathtt{Losers}^\gamma(v)$.

Proof: Since u executes $\mathbb{R}_{\texttt{NewBit}}$ during cs, $\texttt{OkNeg}^{\gamma}(v) = \texttt{true}$ so v is the only parent of u such that $\texttt{tok}_p v^{\gamma} = \bullet$. Thus, the execution of `Announce` produces $\texttt{ph}_u^{\gamma'} = \texttt{ph}_v^{\gamma} = \texttt{ph}_v^{\gamma'}$ (because v is not activated during cs). Since $u \in \texttt{Losers}^{\gamma'}(v)$, we necessarily have $\texttt{b}_v^{\gamma'} \neq \bot$ and $\texttt{b}_u^{\gamma'} \neq \texttt{b}_v^{\gamma'}$.

Since v is not activated during cs, this means that $\texttt{b}_v^{\gamma} \neq \bot$ and that $\texttt{b}_u^{\gamma'} \neq \texttt{b}_v^{\gamma}$. But by definition of `Announce`, we have $\texttt{b}_u^{\gamma'} = \texttt{Bit}_u(\texttt{ph}_v^{\gamma})$, and then we conclude that $\texttt{Bit}_u(\texttt{ph}_v^{\gamma}) \neq \texttt{b}_v^{\gamma} \wedge \texttt{ph}_u^{\gamma} \neq \texttt{ph}_v^{\gamma} \wedge \texttt{b}_v^{\gamma} \neq \bot$ which means that $u \in \texttt{Losers}^{\gamma}(v)$. ■

Lemma 23 proves that the definition of `FullRound` is adapted to our algorithm. Indeed, we establish that one node v such that $\texttt{tok}_v = \bullet$ both before and after a computing step does not see its weight by $\texttt{W}_{\texttt{Circ}}$ increase, unless it makes decrease one higher component of `W` at the same time.

Lemma 23

Let $cs = \gamma \to \gamma'$ be a computing step, and let $v \in V$ be a node such that $\texttt{tok}_v^{\gamma} = \texttt{tok}_v^{\gamma'} = \bullet$. Suppose that $\neg\texttt{FullRound}^{\gamma}(v)$. Then $\neg\texttt{FullRound}^{\gamma'}(v)$ or there exists one child u of v that executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, $\mathbb{R}_{\texttt{ReplayUp}}$, or $\mathbb{E}_{\texttt{TrustChild}}$ during cs.

Proof: Let us first remark that by hypothesis, v does not execute $\mathbb{E}_{\texttt{TrustChild}}$, $\mathbb{R}_{\texttt{Give}}$, or $\mathbb{R}_{\texttt{OfferUp}}$, since it would update $\texttt{tok}_v$.

Suppose first that $\texttt{Losers}^{\gamma}(v) \neq \texttt{Players}^{\gamma}(v)$, and let us consider the different cases depending on which rule v executes during cs.

- If v is not activated during cs, then let us consider one node $u \in \texttt{Players}^{\gamma}(v) \setminus \texttt{Losers}^{\gamma}(v)$. If u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ or $\mathbb{E}_{\texttt{TrustChild}}$, then we obtain the theorem. Lemma 21 states that u does not execute $\mathbb{R}_{\texttt{Lose}}$ during cs. Lemma 22 states that if u execute $\mathbb{R}_{\texttt{NewBit}}$ during cs, then $u \in \texttt{Players}^{\gamma'}(v) \setminus \texttt{Losers}^{\gamma'}(v)$, and the same happens if u is not activated during cs. Thus, either u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ or $\mathbb{E}_{\texttt{TrustChild}}$, either $\texttt{Losers}^{\gamma'}(v) \neq \texttt{Players}^{\gamma'}(v)$, and then $\neg\texttt{FullRound}^{\gamma'}(v)$.
- If v executes $\mathbb{R}_{\texttt{maxPos}}$ or $\mathbb{R}_{\texttt{NewPh}}$ during cs then according to Lemma 20, nodes $u \in \texttt{Players}^{\gamma}(v)$ do not execute $\mathbb{R}_{\texttt{Lose}}$ nor $\mathbb{R}_{\texttt{NewBit}}$. If one such node executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ or $\mathbb{E}_{\texttt{TrustChild}}$, then we obtain the theorem. Thus, we can now suppose that $\forall u \in \texttt{Players}^{\gamma}(v)$, u is not activated during cs. As a corollary, we have $\texttt{Players}^{\gamma'}(v) = \texttt{Players}^{\gamma}(v)$.
- If v executes $\mathbb{R}_{\texttt{maxPos}}$ during cs then $\texttt{MaxPos}(v)$ updates $\texttt{b}_v$ to the maximal value of $\texttt{b}_u$ where $u \in \texttt{Players}^{\gamma}(v)$. Let u be one node with maximal value of `b` among $\texttt{Players}^{\gamma}(v)$. Since $u \in \texttt{Players}^{\gamma'}(v)$ and $\texttt{ph}_u^{\gamma'} = \texttt{ph}_v^{\gamma'} \wedge \texttt{b}_u^{\gamma'} = \texttt{b}_v^{\gamma'}$, we have $u \in \texttt{Players}^{\gamma'}(v) \setminus \texttt{Losers}^{\gamma'}(v)$ and thus $\neg\texttt{FullRound}^{\gamma'}(v)$.
- If v executes $\mathbb{R}_{\texttt{NewPh}}$ during cs then $\texttt{PhasePlus}(v)$ updates $\texttt{b}_v$ to $\bot$, which means that $\texttt{Losers}^{\gamma'}(v) = \emptyset$. Since v executes $\mathbb{R}_{\texttt{NewPh}}$ during cs then $\texttt{Players}^{\gamma}(v) \neq \emptyset$, and by what precedes, $\texttt{Players}^{\gamma'}(v) \neq \emptyset$, which implies that $\texttt{Players}^{\gamma'}(v) \neq \texttt{Losers}^{\gamma'}(v)$ and thus $\neg\texttt{FullRound}^{\gamma'}(v)$.

Let us now suppose that $\texttt{Losers}^{\gamma}(v) = \texttt{Players}^{\gamma}(v)$, which implies $\texttt{Candidates}^{\gamma}(v) = \emptyset$. Since $\texttt{Players}^{\gamma}(v) \subseteq \texttt{Candidates}^{\gamma}(v)$, we have $\texttt{Players}^{\gamma}(v) = \emptyset$, so v cannot execute $\mathbb{R}_{\texttt{maxPos}}$ or $\mathbb{R}_{\texttt{NewPh}}$. Thus, v is not activated during cs (recall that by hypothesis, $\texttt{tok}_v^{\gamma'} = \bullet$). Let us consider one node $u \in \mathcal{C}(v)$. Since $\texttt{Candidates}^{\gamma}(v) = \emptyset$, either $\texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma}$, either $\texttt{tok}_u^{\gamma} \neq \bot$, either $\texttt{play}_u^{\gamma} \in \{\texttt{W}, \texttt{F}\}$.

- If $\texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma}$, then either u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, and then the theorem is established, either not, and then $\texttt{c}_u^{\gamma'} = \texttt{c}_v^{\gamma'}$ so $u \notin \texttt{Candidates}^{\gamma'}(v)$.
- If $\texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma} \wedge \texttt{tok}_u^{\gamma} \neq \bot$, then u does not execute $\mathbb{R}_{\texttt{Return}}$ during cs since v is a parent of u and $\texttt{tok}_v^{\gamma} \notin \{\bot, \downarrow, \uparrow, \circ\}$. Thus, $\texttt{tok}_u^{\gamma'} \neq \bot$, and so $u \notin \texttt{Candidates}^{\gamma'}(v)$.
- If $\texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma} \wedge \texttt{tok}_u^{\gamma} = \bot \wedge \texttt{play}_u^{\gamma} \in \{\texttt{W}, \texttt{F}\}$ then either u executes $\mathbb{R}_{\texttt{ReplayUp}}$ and then the theorem is established, either it does not and then $\texttt{play}_u^{\gamma'} \in \{\texttt{W}, \texttt{F}\}$ so $u \notin \texttt{Candidates}^{\gamma'}(v)$.

Thus, either one child u of v executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, or $\mathbb{R}_{\texttt{ReplayUp}}$, either not but then $\forall u \in \mathcal{C}(v), u \notin \texttt{Candidates}^{\gamma'}(v)$ so $\texttt{Candidates}^{\gamma'}(v) = \emptyset$ and thus $\neg\texttt{FullRound}^{\gamma'}(v)$. ■

Lemma 24 establishes under which conditions we can have an increase of $\texttt{W}_{\texttt{Circ}}$ on one node v. It proves that if this happens, then we fall into the conditions of Theorem 4 or of Theorem 6. This lemma, which largely uses Lemma 23, is the main argument to establish that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})$ is pre-admissible.

Lemma 24

Let $cs = \gamma \to \gamma'$ be a computing step, let a be an anchor at γ', and let $v \in D_a^\gamma$ be a node.
If $\texttt{W}_{\texttt{Circ}}(v,\gamma') > \texttt{W}_{\texttt{Circ}}(v,\gamma)$ then $\texttt{W}_{\texttt{Ch}}(v,\gamma') < \texttt{W}_{\texttt{Ch}}(v,\gamma)$, or v executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{E}_{\texttt{TrustChild}}$, or $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs, or $\texttt{tok}_v^\gamma \notin \{\bot, \uparrow, \downarrow\}$ and one child u of v executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, $\mathbb{R}_{\texttt{ReplayUp}}$ or $\mathbb{E}_{\texttt{TrustChild}}$ during cs.

Proof: Let us consider different cases depending on the value of $\texttt{tok}_v^\gamma$:

- If $\texttt{tok}_v^\gamma = \bot$, then $\texttt{W}_{\texttt{Circ}}(v,\gamma') > \texttt{W}_{\texttt{Circ}}(v,\gamma)$ if and only if v executes $\mathbb{R}_{\texttt{Win}}$ during cs.
- If $\texttt{tok}_v^\gamma = \bigstar$, then $\texttt{W}_{\texttt{Circ}}(v,\gamma') \leq \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \bullet \wedge \texttt{FullRound}^\gamma(v)$ then $\texttt{W}_{\texttt{Circ}}(v,\gamma') \leq \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \bullet \wedge \neg\texttt{FullRound}^\gamma(v) \wedge \texttt{tok}_v^{\gamma'} = \bullet$ then according to Lemma 23 either $\neg\texttt{FullRound}^{\gamma'}(v)$ and thus $\texttt{W}_{\texttt{Circ}}(v,\gamma') = \texttt{W}_{\texttt{Circ}}(v,\gamma)$, either there exists one child u of v that executes $\mathbb{R}_{\texttt{Win}}$, $\mathbb{R}_{\texttt{FakeWin}}$, $\mathbb{R}_{\texttt{ReplayUp}}$, or $\mathbb{E}_{\texttt{TrustChild}}$ during cs.
- If $\texttt{tok}_v^\gamma = \bullet \wedge \neg\texttt{FullRound}^\gamma(v) \wedge \texttt{tok}_v^{\gamma'} \neq \bullet$ then either v executes $\mathbb{E}_{\texttt{TrustChild}}$, either v executes $\mathbb{R}_{\texttt{OfferUp}}$ and then $\texttt{W}_{\texttt{Circ}}(v,\gamma') = 1 < \texttt{W}_{\texttt{Circ}}(v,\gamma)$, either v executes $\mathbb{R}_{\texttt{Give}}$. Let us consider that last case.

  Remark first that if $\texttt{Candidates}^\gamma(v) = \emptyset$ then $\neg\texttt{Give}^\gamma(v)$, and thus v cannot execute $\mathbb{R}_{\texttt{Give}}$ during cs. As a consequence, we have $\texttt{Candidates}^\gamma(v) \neq \emptyset$. Then, since $\neg\texttt{FullRound}^\gamma(v)$ we must have $\texttt{Losers}^\gamma(v) \neq \texttt{Players}^\gamma(v)$, which is possible only if $\texttt{Players}^\gamma(v) \neq \emptyset$ (recall that $\texttt{Losers}^\gamma(v) \subseteq \texttt{Players}^\gamma(v)$).

  Thus, we necessarily have $\texttt{Players}^\gamma(v) \neq \emptyset$, and actually since v executes $\mathbb{R}_{\texttt{Give}}$ during cs, there $\left|\texttt{Players}^\gamma(v)\right| = 1$. Let u be that node in $\texttt{Players}^\gamma(v)$. If u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, we have our result, let us now consider that it does not. Since $\texttt{Losers}^\gamma(v) \neq \texttt{Players}^\gamma(v)$, $u \notin \texttt{Losers}^\gamma(v)$ and thus, according to Lemma 21, u does not execute $\mathbb{R}_{\texttt{Lose}}$ during cs. As a consequence, u either executes $\mathbb{R}_{\texttt{NewBit}}$, either is not activated during cs. In both cases, variables $\texttt{tok}$, $\texttt{c}$, and $\texttt{play}$ are not updated on u during cs, and thus $u \in \texttt{Players}^{\gamma'}(v)$, so $\texttt{W}_{\texttt{Circ}}(v,\gamma') = 1 < \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \downdownarrows \wedge \texttt{Players}^\gamma(v) = \emptyset$ then $\texttt{W}_{\texttt{Circ}}(v,\gamma') \leq \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \downdownarrows \wedge \texttt{Players}^\gamma(v) \neq \emptyset$ then let us consider $u \in \texttt{Players}^\gamma(v)$. By definition, $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) \wedge (\texttt{play}_u^\gamma = \texttt{P} \wedge \texttt{tok}_u^\gamma \neq \bigstar)$. As a consequence, v does not execute $\mathbb{R}_{\texttt{NewPlay}}$ during cs. If v executes $\mathbb{E}_{\texttt{TrustChild}}$ during cs, then the desired property holds.

  Let us now suppose that v does not execute any rule during cs. Since $\texttt{tok}_v^\gamma = \downdownarrows$, u cannot execute $\mathbb{R}_{\texttt{Lose}}$ or $\mathbb{R}_{\texttt{NewBit}}$ during cs, since it requires that one other parent p of u is such that $\texttt{tok}_p^\gamma = \bullet$, and then $\neg\texttt{OkNeg}^\gamma(u)$. Thus, either u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ during cs and then our property holds, either it does not, but then it is not activated and thus $u \in \texttt{Players}^{\gamma'}(v)$ which means that $\texttt{W}_{\texttt{Circ}}^{\gamma'} = 1 = \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \circlearrowleft$, then $\texttt{tok}_v^{\gamma'} \in \{\circlearrowleft, \downarrow\}$ and thus $\texttt{W}_{\texttt{Circ}}(v,\gamma') \leq \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \downarrow$ then $\texttt{tok}_v^{\gamma'} \in \{\downarrow, \circ\}$ and thus $\texttt{W}_{\texttt{Circ}}(v,\gamma') \leq \texttt{W}_{\texttt{Circ}}(v,\gamma)$.
- If $\texttt{tok}_v^\gamma = \circ$, then $\texttt{W}_{\texttt{Circ}}(v,\gamma') > \texttt{W}_{\texttt{Circ}}(v,\gamma)$ only if v executes $\mathbb{E}_{\texttt{TrustChild}}$ or $\mathbb{R}_{\texttt{ReNego}}$ during cs. If v executes $\mathbb{E}_{\texttt{TrustChild}}$, then our property holds, let us now suppose that v executes $\mathbb{R}_{\texttt{ReNego}}$ during cs.

We have $\forall u \in \mathcal{C}_{\texttt{neq}}{}^{\gamma}(v), \texttt{play}_u^{\gamma} \neq \texttt{L}$. Let us first prove that $\forall u \in \mathcal{C}_{\texttt{neq}}{}^{\gamma'}(v), \texttt{play}_u^{\gamma'} \neq \texttt{L}$, and let us consider $u \in \mathcal{C}^{\gamma}(v)$. Since v executes $\mathbb{R}_{\texttt{ReNego}}$ during cs, we have $\texttt{tok}_u^{\gamma} = \bot$. If $\texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma}$ then u switches its color only if it executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, and thus the property holds. Otherwise if $\texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma}$ then $\texttt{play}_u^{\gamma} \neq \texttt{L}$ and since $\texttt{tok}_v^{\gamma} = \circ\, u$ cannot execute $\mathbb{R}_{\texttt{Lose}}$ during cs, and thus $\texttt{play}_u^{\gamma'} \neq \texttt{L}$.

Since $\texttt{b}_v^{\gamma'} = \bot$, we have $\texttt{Losers}^{\gamma'}(v) = \emptyset$. Thus, either $\exists u \in \texttt{Players}^{\gamma'}(v)$ and thus $\texttt{Losers}^{\gamma'} \neq \texttt{Players}^{\gamma'}(v)$ so $\neg\texttt{FullRound}^{\gamma'}(v)$, either $\texttt{Players}^{\gamma'}(v) = \emptyset$ but then according to what we established just above, we have $\texttt{Candidates}^{\gamma'}(v) = \emptyset$ and once again $\neg\texttt{FullRound}^{\gamma'}(v)$. In both cases, $\texttt{W}_{\texttt{Circ}}^{\gamma'}(v) = 2 < \texttt{W}_{\texttt{Circ}}^{\gamma}(v)$.

- If $\texttt{tok}_v^{\gamma} = \uparrow$ then either $v \neq r$ and thus $\texttt{tok}_v^{\gamma} \in \{\uparrow, \bot\}$ so $\texttt{W}_{\texttt{Circ}}(v, \gamma') \leq \texttt{W}_{\texttt{Circ}}(v, \gamma)$, either $v = r$ and thus, if activated, v executes $\mathbb{R}^r_{\texttt{NewDFS}}$ and the property holds. ■

Theorem 7 establishes that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})$ is pre-admissible.

Theorem 7

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs.

We have $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma'}, \gamma') \preceq^3 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma}, \gamma)$.

Proof: According to Theorem 5, we already have $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma'}, \gamma') \preceq (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_t^{\gamma}, \gamma)$. Consequently, we only have to prove that if $\texttt{W}_{\texttt{Circ}}(D_a^{\gamma'}, \gamma') > \texttt{W}_{\texttt{Circ}}(D_a^{\gamma}, \gamma)$, then we have $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma'}, \gamma') \prec^2 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma}, \gamma)$.

In the same way as we did in the proof of Theorem 5, we can suppose that for any branch $\mathcal{B}$ of G_a, $D_a^{\gamma'}(\mathcal{B})$ is not longer than $D_a^{\gamma}(\mathcal{B})$. In such circumstances, we have $\texttt{W}_{\texttt{Circ}}(D_a^{\gamma'}, \gamma') > \texttt{W}_{\texttt{Circ}}(D_a^{\gamma}, \gamma)$ only if there exists a branch $\mathcal{B}$ of G_a such that $\exists v \in D_a^{\gamma}(\mathcal{B}) : \texttt{W}_{\texttt{Circ}}(v, \gamma') > \texttt{W}_{\texttt{Circ}}(v, \gamma)$. Let us now apply Lemma 24. If v executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{E}_{\texttt{TrustChild}}$ (it cannot execute $\mathbb{R}^r_{\texttt{NewDFS}}$ by hypothesis), then Theorem 3 or Theorem 5 guarantees that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma'}, \gamma') \prec^2 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma}, \gamma)$.

Otherwise, $\texttt{tok}_v^{\gamma} \notin \{\bot, \uparrow, \downarrow\}$ and $\exists u \in \mathcal{C}(v)$ that executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ or $\mathbb{R}_{\texttt{ReplayUp}}$ or $\mathbb{E}_{\texttt{TrustChild}}$, and once again Theorem 3 or Theorem 5 guarantees that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma'}, \gamma') \prec^2 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}})(D_a^{\gamma}, \gamma)$. ■

Theorem 8 proves that under certain circumstances, we are certain that the weight of one $\texttt{CD}^o$ decreases. It will be useful to prove that $\texttt{W}$ is an admissible function, but also to prove that the other, longer, prefixes of $\texttt{W}$ are pre-admissible. Indeed, if in any computing step one node updates its variable $\texttt{tok}$ then we do not need to look beyond the third component to be assured that the weight does not increase, since $\preceq^k$ is a lexicographic order.

Theorem 8

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs.

If $\exists v \in D_a^{\gamma}$ such that v executes one rule among $\mathbb{R}_{\texttt{Give}}$, $\mathbb{R}_{\texttt{NewPlay}}$, $\mathbb{R}_{\texttt{Drop}}$, $\mathbb{R}_{\texttt{Nego}}$, $\mathbb{R}_{\texttt{OfferUp}}$, $\mathbb{R}_{\texttt{Receive}}$, $\mathbb{R}_{\texttt{Return}}$, $\mathbb{R}_{\texttt{ReNego}}$ during cs, then $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma'}, \gamma') \prec^3 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma}, \gamma)$.

Proof: Let us consider one such node $v \in D_a^{\gamma}$, and one branch $\mathcal{B}$ of G_a such that $v \in D_a^{\gamma}(\mathcal{B})$. If $v \notin D_a^{\gamma'}(\mathcal{B})$ then according to Lemma 18 we have $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma'}, \gamma') \prec^3 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma}, \gamma)$.

Else, since any of these rules guarantee that $\texttt{W}_{\texttt{Circ}}(v, \gamma') \neq \texttt{W}_{\texttt{Circ}}(v, \gamma)$ and since we established in Theorem 7 that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma'}, \gamma') \preceq^3 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma}, \gamma)$, we have $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma'}, \gamma') \prec^3 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma}, \gamma)$. ■

C.2.7 Fourth component of W: $\mathtt{W_{Play}}$, circulation of variable play on children

Explanations The three weight functions that we previously defined, $\mathtt{W_{Ch}}$, $\mathtt{W_{Er}}$, and $\mathtt{W_{Circ}}$ totally embrace the rules that affect $\mathtt{tok}_v$ and $\mathtt{c}_v$. Thus, we can now consider that both these variable are constant on nodes, and focus on the two other variables that are play and id. In this section, we only treat variable play. Remark that the rules $\mathbb{R}_{\mathtt{Win}}$, $\mathbb{R}_{\mathtt{FakeWin}}$, and $\mathbb{R}_{\mathtt{Return}}$ have an impact on variable play, but were already treated in Sections C.2.4 and C.2.6. Thus, although we discussed rule $\mathbb{R}_{\mathtt{ReplayUp}}$ in Section C.2.4, we did not treat all the cases where this rule might be executed, and thus we must consider it in this section. Consequently, the rules that interest us in this section are $\mathbb{R}_{\mathtt{Lose}}$, $\mathbb{R}_{\mathtt{ReplayD}}$, $\mathbb{R}_{\mathtt{ReplayUp}}$, $\mathbb{R}_{\mathtt{Fake}}$, and $\mathbb{R}_{\mathtt{ReWin}}$.

Depending on its variable $\mathtt{tok}_v$, it happens that one node v is waiting for some of its children to update their variable play, before it executes a rule that updates $\mathtt{tok}_v$ to the next state. For example, if $\mathtt{tok}_v = \downdownarrows$, then v does not execute $\mathbb{R}_{\mathtt{NewPlay}}$ until all of its children u such that $\mathtt{tok}_u = \mathtt{P}$ updates $\mathtt{play}_u$ to L, W, or F. In order to prove that our algorithm progresses, and that circulation might terminate, we need to prove that the number of children u of v such that $\mathtt{play}_u = \mathtt{P}$ never increases, at least as long as $\mathtt{tok}_v = \downdownarrows$. Indeed, as soon as $\mathtt{tok}_v$ is updated, then we already know that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})$ decreases. Actually, when $\mathtt{tok}_v = \downdownarrows$, children u of v such that $\mathtt{play}_u = \mathtt{P}$ will update $\mathtt{play}_u$ with rule $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$, both cases have already been treated in Section C.2.4, so we do not need to elaborate any further on this case.

There exists other situations where one node v waits until its children u have some specific values for their variable $\mathtt{play}_u$ before v itself updates $\mathtt{tok}_v$. For each of those situations, we want to define $\mathtt{W_{Play}}$ such that any time one child u of v updates its variable $\mathtt{play}_u$ in a sense that allows v to update $\mathtt{tok}_v$, $\mathtt{W_{Play}}$ decreases on v. Thus, we define $\mathtt{W_{Play}}$ such that each node v counts the number of children it has that prevent it from updating its variable $\mathtt{tok}_v$. In order to have a function that is pre-admissible, we also require that $\mathtt{W_{Play}}$ never increases, and thus that no action taken by one node u might add u to the set of children that prevent one of its parents v to update $\mathtt{tok}_v$.

The situations where one node v waits for its children u to update $\mathtt{play}_u$ occurs when $\mathtt{tok}_v \in \{\bullet, \circlearrowleft, \circ, \uparrow\}$. Let us give some details:

- If $\mathtt{tok}_v = \bullet$ and $\mathtt{b}_v \neq \bot$, then v waits for some of its children u to execute $\mathbb{R}_{\mathtt{Lose}}$ and update $\mathtt{play}_u$ from P to L.

  Although rule $\mathbb{R}_{\mathtt{Lose}}$ could be treated in the following section that focuses on negotiation, we treat it here for two reasons. First, it concerns variable play as well, and thus it remains consistent here. Second, having already treated executions of rule $\mathbb{R}_{\mathtt{Lose}}$ will help us greatly in the section on negotiation.

- If $\mathtt{tok}_v = \circ$ or $\mathtt{tok}_v = \circlearrowleft$, then v does not execute $\mathbb{R}_{\mathtt{Drop}}$ until all of its children u such that $\mathtt{tok}_u = \mathtt{L}$ execute $\mathbb{R}_{\mathtt{ReplayD}}$ and update $\mathtt{play}_u$ to P.

- If $\mathtt{tok}_v = \uparrow$, then before executing $\mathbb{R}_{\mathtt{Return}}$, v waits for all of its children u such that $\mathtt{play}_u = \mathtt{W}$ execute $\mathbb{R}_{\mathtt{ReplayUp}}$ or $\mathbb{R}_{\mathtt{Fake}}$, and update $\mathtt{play}_u$ from W to P or F, depending on whether u still has parents in any $\mathtt{CD}^o$.

From what precedes, if one node u executes $\mathbb{R}_{\mathtt{Lose}}$, $\mathbb{R}_{\mathtt{ReplayD}}$, $\mathbb{R}_{\mathtt{ReplayUp}}$, or $\mathbb{R}_{\mathtt{Fake}}$, it will make decrease the weight of the $\mathtt{CD}^o$'s that contain the parents v of u that have the corresponding value of $\mathtt{tok}_v$. Indeed, all of these four rules are conditioned by the existence of at least one parent v with the adapted value of $\mathtt{tok}_v$.

Remark that we did not mentioned rule $\mathbb{R}_{\mathtt{ReWin}}$. It is not difficult to define a weight function that decreases on v each time one child u of v execute $\mathbb{R}_{\mathtt{ReWin}}$. The difficulty comes from proving that the weight function we defined that way is pre-admissible. Indeed, $\mathbb{R}_{\mathtt{ReWin}}$ might be executed

in very various situations: as soon as u has one no parent v such that $\texttt{tok}_v = \uparrow$. Unfortunately, there exists situations in which we can give no guarantee that there won't be executions of $\mathbb{R}_{\texttt{Fake}}$ that cancels executions of $\mathbb{R}_{\texttt{ReWin}}$, and thus we cannot treat both $\mathbb{R}_{\texttt{Fake}}$ and $\mathbb{R}_{\texttt{ReWin}}$ at the same layer of $\texttt{W}$.

However, the design of the algorithm allows us design $\texttt{W}_{\texttt{Play}}$ that is pre-admissible and such that it decreases any time one node u executes $\mathbb{R}_{\texttt{Lose}}$, $\mathbb{R}_{\texttt{ReplayD}}$, $\mathbb{R}_{\texttt{ReplayUp}}$, or $\mathbb{R}_{\texttt{Fake}}$. This is stated in Theorems 9 and 10.

Figure 27 summarizes the different transitions that might be taken by variable $\texttt{play}_u$ on one node u.

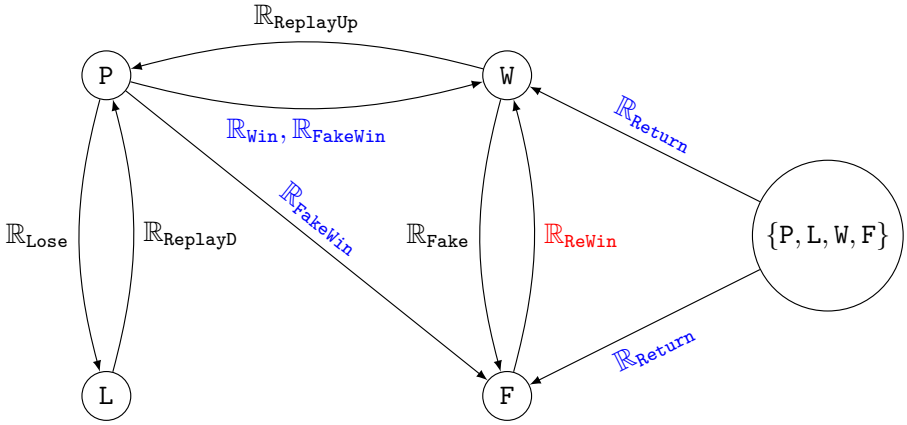

Figure 27: Diagram of variable $\texttt{play}_u$.

We represented in blue the rules that have already been treated in the previous sections, and in red the rule $\mathbb{R}_{\texttt{ReWin}}$ that is not treated in that section.

Definitions Let us first define the different sets that contain the children u of v that we are going to count in the different cases described above.

If $\texttt{tok}_v^\gamma = \bullet$, then we want to count children $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v)$ that might execute $\mathbb{R}_{\texttt{Lose}}$. One node u can execute $\mathbb{R}_{\texttt{Lose}}$ only if $\texttt{play}_u^\gamma = \texttt{P}$, so nodes $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) : \texttt{play}_u^\gamma = \texttt{P}$ must obviously be counted. Yet, we cannot count only those children of v. Indeed, it might happen that one node $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) : \texttt{play}_u^\gamma = \texttt{W}$ execute $\mathbb{R}_{\texttt{ReplayUp}}$, which produces $\texttt{play}_u^{\gamma'} = \texttt{P}$, and would make $\texttt{W}_{\texttt{Play}}$ increase on v. Although it would at the same time make $\texttt{W}_{\texttt{Play}}$ decrease on one other node w such that $\texttt{play}_w = \uparrow$, since we require that $\texttt{W}$ decrease on each branch of each $\texttt{CD}^o$, we must prevent this from happening. Thus, we must include nodes $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) : \texttt{play}_u^\gamma = \texttt{W}$ in our count for $\texttt{tok}_v = \bullet$. For the same reason, due to rule $\mathbb{R}_{\texttt{ReWin}}$, we must also include nodes $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) : \texttt{play}_u^\gamma = \texttt{F}$ in that set. This leads to the following predicate:

$$\texttt{ChLose}^\gamma(v) = \{u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) \mid \texttt{tok}_u^\gamma = \bot \wedge \texttt{play}_u^\gamma \neq \texttt{L}\} \tag{46}$$

If $\texttt{tok}_v^\gamma = \circlearrowleft$ or $\texttt{tok}_v^\gamma = \circ$, then we want to count children $u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v)$ that might execute $\mathbb{R}_{\texttt{ReplayD}}$. One node u can execute $\mathbb{R}_{\texttt{ReplayD}}$ only if $\texttt{play}_u^\gamma = \texttt{L}$. Furthermore, due to predicate $\texttt{OkNeg}$, no node u that has a parent v with $\texttt{tok}_v^\gamma \in \{\circlearrowleft, \circ\}$ can execute $\mathbb{R}_{\texttt{Lose}}$. This leads to the following predicate:

$$\texttt{ChReplay}^\gamma(v) = \{u \in \mathcal{C}_{\texttt{neq}}{}^\gamma(v) \mid \texttt{tok}_u^\gamma = \bot \wedge \texttt{play}_u^\gamma = \texttt{L}\} \tag{47}$$

If $\texttt{tok}_v^\gamma = \uparrow$ then we want to count children $u \in \mathcal{C}_{\texttt{eq}}{}^\gamma(v)$ that might execute $\mathbb{R}_{\texttt{ReplayUp}}$ or $\mathbb{R}_{\texttt{Fake}}$. One node might execute $\mathbb{R}_{\texttt{ReplayUp}}$ or $\mathbb{R}_{\texttt{Fake}}$ only if $\texttt{play}_u^\gamma = \texttt{W}$. One node u might updates

its variable $\texttt{play}_u$ from P to W only if it executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, cases that have been treated previously. Furthermore, due to predicate StopFaking, no node u that has a parent $v \in \mathcal{P}_{\texttt{eq}}(u)$ such that $\texttt{tok}_v = \uparrow$ can execute $\mathbb{R}_{\texttt{ReWin}}$, and thus update its variable $\texttt{play}_u$ from F to W. This leads to the following predicate:

$$\texttt{ChEnd}^{\gamma}(v) = \{u \in \mathcal{C}_{\texttt{eq}}{}^{\gamma}(v) \mid \texttt{tok}_u^{\gamma} = \bot \wedge \texttt{play}_u^{\gamma} = \texttt{W}\} \tag{48}$$

We can now define $\texttt{W}_{\texttt{Play}}(v, \gamma)$ depending on the value of $\texttt{tok}_v^{\gamma}$.

$$\texttt{W}_{\texttt{Play}}(v,\gamma) = \begin{cases} |\texttt{ChLose}^{\gamma}(v)| & \text{if } \texttt{tok}_v^{\gamma} = \bullet \\ |\texttt{ChReplay}^{\gamma}(v)| & \text{if } \texttt{tok}_v^{\gamma} \in \{\circ, \circlearrowleft\} \\ |\texttt{ChEnd}^{\gamma}(v)| & \text{if } \texttt{tok}_v^{\gamma} = \uparrow \\ 0 & \text{if } \texttt{tok}_v \in \{\bot, \downdownarrows, \downarrow, \bigstar\} \end{cases} \tag{49}$$

Proofs Lemmas 25, 26 and 27 prove that the definitions of ChLose, ChReplay, and ChEnd are consistent with our algorithm. For $\texttt{ChLose}(v)$ and $\texttt{ChReplay}(v)$ we prove in Lemmas 25 and 26 that both these sets do not increase unless it is by one node $u \in \mathcal{C}(v)$ that executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, which makes decrease one more important quantity. We cannot use the same reasoning for $\texttt{ChEnd}(v)$. Indeed, since we consider one node v such that $\texttt{tok}_v = \uparrow$, $\texttt{W}_{\texttt{Ch}}(v)$ is not affected by any action taken by one child $u \in \mathcal{C}(v)$. Thus, we prove in Lemmas 27 that $\texttt{ChEnd}(v)$ never increases, as long as $\texttt{tok}_v = \uparrow$.

We also establish in these lemmas some particular cases which necessarily decrease $\texttt{W}_{\texttt{Play}}$.

Lemma 25

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^{\gamma} = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$.

If $\forall u \in \mathcal{C}(v)$, u does not execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, then $\texttt{ChLose}^{\gamma'}(v) \subseteq \texttt{ChLose}^{\gamma}(v)$. Furthermore, if $\exists u \in \mathcal{C}_{\texttt{neq}}{}^{\gamma}(v)$ that executes $\mathbb{R}_{\texttt{Lose}}$ during cs, then $\texttt{W}_{\texttt{Play}}(v, \gamma') < \texttt{W}_{\texttt{Play}}(v, \gamma)$.

Proof: For the first part, we prove the equivalent statement, under the same conditions:

$$\mathcal{C}(v) \setminus \texttt{ChLose}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{ChLose}^{\gamma'}(v)$$

Let us consider $u \in \mathcal{C}(v) \setminus \texttt{ChLose}^{\gamma}(v)$. If $\texttt{tok}_u^{\gamma} \neq \bot$ then since $\texttt{tok}_v^{\gamma} = \bullet$, u does not execute $\mathbb{R}_{\texttt{Return}}$ and thus $\texttt{tok}_u^{\gamma'} \neq \bot$ so $u \notin \texttt{ChLose}^{\gamma'}(v)$. Otherwise, $\texttt{tok}_u^{\gamma} = \bot$. If $u \in \mathcal{C}_{\texttt{eq}}{}^{\gamma}(v)$, then since u does not execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, $u \in \mathcal{C}_{\texttt{eq}}{}^{\gamma'}(v)$ and thus $u \notin \texttt{ChLose}^{\gamma'}(v)$. Otherwise, $u \in \mathcal{C}_{\texttt{neq}}{}^{\gamma}(v)$, and thus $\texttt{play}_u^{\gamma} = \texttt{L}$. Since $v \in \mathcal{P}_{\texttt{neq}}{}^{\gamma}(v)$ and $\texttt{tok}_v = \bullet$, $\neg\texttt{ReplayL}^{\gamma}(u)$ so u does not execute $\mathbb{R}_{\texttt{ReplayD}}$ so $\texttt{play}_u^{\gamma'} = \texttt{L}$ and thus $u \notin \texttt{ChLose}^{\gamma'}(v)$.

Let us now prove the second part. By definition of $\mathbb{R}_{\texttt{Lose}}$ we have $\texttt{play}_u^{\gamma} = \texttt{P} \wedge \texttt{tok}_v^{\gamma} = \bot$, and thus, $u \in \texttt{ChLose}^{\gamma}(v)$. But since u executes $\mathbb{R}_{\texttt{Lose}}$ during cs, $\texttt{play}_u^{\gamma'} \neq \texttt{P}$ and thus $u \notin \texttt{ChLose}^{\gamma'}(v)$. Then, what precedes implies that $\texttt{ChLose}^{\gamma'}(v) \subsetneq \texttt{ChLose}^{\gamma}(v)$, which leads to $\texttt{W}_{\texttt{Play}}(v, \gamma') < \texttt{W}_{\texttt{Play}}(v, \gamma)$. ■

Lemma 26

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^{\gamma} \in \{\circlearrowleft, \circ\}$ and $\texttt{tok}_v^{\gamma'} \in \{\circlearrowleft, \circ\}$.

If $\forall u \in \mathcal{C}(v)$, u does not execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, then $\texttt{ChReplay}^{\gamma'}(v) \subseteq \texttt{ChReplay}^{\gamma}(v)$. Furthermore, if $\exists u \in \mathcal{C}_{\texttt{neq}}{}^{\gamma}(v)$ that executes $\mathbb{R}_{\texttt{ReplayD}}$ during cs, then $\texttt{W}_{\texttt{Play}}(v, \gamma') < \texttt{W}_{\texttt{Play}}(v, \gamma)$.

Proof: For the first part, we prove the equivalent statement, under the same conditions:

$$\mathcal{C}(v) \setminus \texttt{ChReplay}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{ChReplay}^{\gamma'}(v)$$

Let us consider $u \in \mathcal{C}(v) \setminus \texttt{ChReplay}^{\gamma}(v)$. If $\texttt{tok}_u^{\gamma} \neq \bot$ then since $\texttt{tok}_v^{\gamma} = \bullet$, u does not execute $\mathbb{R}_{\texttt{Return}}$ and thus $\texttt{tok}_u^{\gamma'} \neq \bot$ so $u \notin \texttt{ChReplay}^{\gamma'}(v)$. Otherwise, $\texttt{tok}_u^{\gamma} = \bot$. If $u \in {\mathcal{C}_{\texttt{eq}}}^{\gamma}(v)$, then since u does not execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$, then $u \in {\mathcal{C}_{\texttt{eq}}}^{\gamma'}(v)$ and thus $u \notin \texttt{ChReplay}^{\gamma'}(v)$. Otherwise, $u \in {\mathcal{C}_{\texttt{neq}}}^{\gamma}(v)$, and thus $\texttt{play}_u^{\gamma} \in \{\texttt{P}, \texttt{W}, \texttt{F}\}$. If $\texttt{play}_u^{\gamma} \in \{\texttt{W}, \texttt{F}\}$ no rule can induce $\texttt{play}_u^{\gamma'} = \texttt{L}$. If $\texttt{play}_u^{\gamma} = \texttt{P}$, then since $v \in \texttt{ParNeg}^{\gamma}(u) \wedge \texttt{tok}_v^{\gamma} \neq \bullet$, then due to $\texttt{OkNeg}$ u cannot execute $\mathbb{R}_{\texttt{Lose}}$ during cs, and thus $u \notin \texttt{ChReplay}^{\gamma'}(v)$.

Let us now prove the second part. By definition of $\mathbb{R}_{\texttt{ReplayD}}$ we have $\texttt{play}_u^{\gamma} = \texttt{L} \wedge \texttt{tok}_v^{\gamma} = \bot$, and thus, $u \in \texttt{ChReplay}^{\gamma}(v)$. But since u executes $\mathbb{R}_{\texttt{ReplayD}}$ during cs, $\texttt{play}_u^{\gamma'} \neq \texttt{L}$ and thus $u \notin \texttt{ChReplay}^{\gamma'}(v)$. Then, what precedes implies that $\texttt{ChReplay}^{\gamma'}(v) \subsetneq \texttt{ChReplay}^{\gamma}(v)$, which leads to $\texttt{W}_{\texttt{Play}}(v, \gamma') < \texttt{W}_{\texttt{Play}}(v, \gamma)$. ■

Lemma 27

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^{\gamma} = \uparrow$ and $\texttt{tok}_v^{\gamma'} = \uparrow$.

We have $\texttt{ChEnd}^{\gamma'}(v) \subseteq \texttt{ChEnd}^{\gamma}(v)$. Furthermore, if $\exists u \in {\mathcal{C}_{\texttt{eq}}}^{\gamma}(v)$ that executes $\mathbb{R}_{\texttt{Fake}}$ or $\mathbb{R}_{\texttt{ReplayUp}}$ during cs, then $\texttt{W}_{\texttt{Play}}(v, \gamma') < \texttt{W}_{\texttt{Play}}(v, \gamma)$.

Proof: For the first part, we prove the equivalent statement:

$$\mathcal{C}(v) \setminus \texttt{ChEnd}^{\gamma}(v) \subseteq \mathcal{C}(v) \setminus \texttt{ChEnd}^{\gamma'}(v)$$

Let us consider $u \in \mathcal{C}(v) \setminus \texttt{ChEnd}^{\gamma}(v)$. If $u \in {\mathcal{C}_{\texttt{neq}}}^{\gamma}(v)$ we have $u \in \texttt{ChEnd}^{\gamma'}(v)$ only if u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ to switch its color. If u executes $\mathbb{R}_{\texttt{Win}}$ then $\texttt{tok}_u^{\gamma'} = \bigstar$ so $u \notin \texttt{ChEnd}^{\gamma'}(v)$. If u executes $\mathbb{R}_{\texttt{FakeWin}}$ then since $v \in {\mathcal{P}_{\texttt{neq}}}^{\gamma}(v) \wedge \texttt{tok}_v^{\gamma} = \uparrow$, the execution of $\texttt{FakeWin}(v)$ induces $\texttt{play}_u^{\gamma'} = \texttt{F}$, and then $u \notin \texttt{ChEnd}^{\gamma'}(v)$.

Suppose now $u \in {\mathcal{C}_{\texttt{eq}}}^{\gamma}(v)$. If $\texttt{tok}_u^{\gamma} \neq \bot$ then $u \in \texttt{ChEnd}^{\gamma'}(v)$ only if u executes $\mathbb{R}_{\texttt{Return}}$ during cs, but if it does since $v \in {\mathcal{P}_{\texttt{eq}}}^{\gamma}(v) \wedge \texttt{tok}_v^{\gamma} = \uparrow$, the execution of $\texttt{Drop}(v)$ induces $\texttt{play}_u^{\gamma'} = \texttt{F}$, and then $u \notin \texttt{ChEnd}^{\gamma'}(v)$.

Suppose now $\texttt{tok}_u^{\gamma} = \bot$. Then we have $\texttt{play}_u^{\gamma} \neq \texttt{W}$. Remark first that u cannot execute $\mathbb{R}_{\texttt{Return}}$ during cs since $\texttt{tok}_u^{\gamma} = \bot$. If $\texttt{play}_u^{\gamma} = \texttt{L}$ then we cannot have $\texttt{play}_u^{\gamma'} = \texttt{W}$, necessary condition to have $u \in \texttt{ChEnd}^{\gamma'}(v)$. If $\texttt{play}_u^{\gamma} = \texttt{P}$, then we have $\texttt{play}_u^{\gamma'} = \texttt{W}$ only if u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ during cs, but if it does, then it switches its color$\texttt{c}_u$, so $u \in {\mathcal{C}_{\texttt{neq}}}^{\gamma}(v)$ and then $u \notin \texttt{ChEnd}^{\gamma'}(v)$. Finally, if $\texttt{play}_u^{\gamma} = \texttt{F}$ then u cannot execute $\mathbb{R}_{\texttt{ReWin}}$ since $v \in {\mathcal{P}_{\texttt{eq}}}^{\gamma}(v) \wedge \texttt{tok}_v^{\gamma} = \uparrow$, we have $\neg\texttt{FakeWin}^{\gamma}(u)$, and thus $u \notin \texttt{ChEnd}^{\gamma'}(v)$.

Let us now prove the second part. By definition of $\mathbb{R}_{\texttt{Fake}}$ and $\mathbb{R}_{\texttt{ReplayUp}}$ we have $\texttt{play}_u^{\gamma} = \texttt{W} \wedge \texttt{tok}_v^{\gamma} = \bot$, and thus, $u \in \texttt{ChEnd}^{\gamma}(v)$. But since u executes $\mathbb{R}_{\texttt{Fake}}$ or $\mathbb{R}_{\texttt{ReplayUp}}$ during cs, $\texttt{play}_u^{\gamma'} \neq \texttt{W}$ and thus $u \notin \texttt{ChEnd}^{\gamma'}(v)$. Then, what precedes implies that $\texttt{ChEnd}^{\gamma'}(v) \subsetneq \texttt{ChEnd}^{\gamma}(v)$, which leads to $\texttt{W}_{\texttt{Play}}(v, \gamma') < \texttt{W}_{\texttt{Play}}(v, \gamma)$. ■

Lemma 28 combines the results of Lemmas 25, 26 and 27 and proves that $\texttt{W}_{\texttt{Play}}$ can decrease on one node only if the previous components of $\texttt{W}$ decrease on the $\texttt{CD}^o$'s that include that node.

Lemma 28

Let $cs = \gamma \to \gamma'$ be a computing step, let a be an anchor at γ' that does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs, and let $v \in D_a^{\gamma}$ be a node.

If $\texttt{W}_{\texttt{Play}}(v, \gamma') > \texttt{W}_{\texttt{Play}}(v, \gamma)$ then $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma'}, \gamma') \prec^3 (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})(D_a^{\gamma}, \gamma)$.

Proof: Let us first eliminate the cases in which the decrease of $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}})$ is immediate. According to Theorem 4 we can assume that, if $\texttt{tok}_v^{\gamma} \neq \uparrow$, then $\forall u \in \mathcal{C}(v)$, u does not execute $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$. Furthermore, according to Theorems 6 and 8, we can assume that v does not execute any rule among $\mathbb{E}_{\texttt{TrustChild}}$, $\mathbb{R}_{\texttt{Give}}$, $\mathbb{R}_{\texttt{NewPlay}}$, $\mathbb{R}_{\texttt{Drop}}$, $\mathbb{R}_{\texttt{Nego}}$, $\mathbb{R}_{\texttt{OfferUp}}$, $\mathbb{R}_{\texttt{Receive}}$, $\mathbb{R}_{\texttt{Return}}$, and $\mathbb{R}_{\texttt{ReNego}}$ during cs.

Furthermore since $v \in D_a^\gamma$, it is obvious that $\mathtt{tok}_v^\gamma \neq \bot$ and thus that v cannot execute $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$.

As a direct consequence, we have $\mathtt{tok}_v^{\gamma'} = \mathtt{tok}_v^\gamma$ and $\mathtt{c}_v^{\gamma'} = \mathtt{c}_v^\gamma$. Since $\mathtt{tok}_v^{\gamma'} = \mathtt{tok}_v^\gamma$ we can have $\mathtt{W_{Play}}(v,\gamma') > \mathtt{W_{Play}}(v,\gamma)$ only if $\mathtt{tok}_v^\gamma \in \{\circ, \circlearrowleft, \uparrow, \bullet\}$. Since $\mathtt{tok}_v^\gamma = \mathtt{tok}_v^{\gamma'}$ and for the cases where $\mathtt{tok}_v^\gamma \in \{\bullet, \circlearrowleft, \circ\}$, we have no children $u \in \mathcal{C}(v)$ executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$ during cs, then we fall in the preconditions of Lemmas 25, 26, and 27. Whatever the value of $\mathtt{tok}_v^\gamma$, the corresponding lemma guarantees that $\mathtt{W_{Play}}(v,\gamma') \leq \mathtt{W_{Play}}(v,\gamma)$. ■

Theorem 9 establishes that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})$ is pre-admissible.

Theorem 9

Let $cs = \gamma \to \gamma'$ be a computing step, and let a be an anchor at γ', and suppose that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs.

We have $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \preceq^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^\gamma, \gamma)$.

Proof: Theorem 7 establishes $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})(D_a^{\gamma'}, \gamma') \preceq^3 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})(D_a^\gamma, \gamma)$. Suppose now $\mathtt{W_{Play}}(D_a^{\gamma'}, \gamma') > \mathtt{W_{Play}}(D_a^\gamma, \gamma)$, and prove $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})(D_a^{\gamma'}, \gamma') \prec^3 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})(D_a^\gamma, \gamma)$.

In the same way as we did in the proof of Theorem 5, we can suppose that for any branch $\mathcal{B}$ of G_a, $D_a^{\gamma'}(\mathcal{B})$ is not longer than $D_a^\gamma(\mathcal{B})$.

In such circumstances, we have $\mathtt{W_{Play}}(D_a^{\gamma'}, \gamma') > \mathtt{W_{Play}}(D_a^\gamma, \gamma)$ only if there exists a branch $\mathcal{B}$ of G_a such that $\exists v \in D_a^\gamma(\mathcal{B}) : \mathtt{W_{Play}}(v,\gamma') > \mathtt{W_{Play}}(v,\gamma)$. But then, according to Lemma 28, we have $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})(D_a^{\gamma'}, \gamma') \prec^3 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}})(D_a^\gamma, \gamma)$. ■

Theorem 10 proves that under certain circumstances, we are certain that the weight of one $\mathtt{CD}^o$ decreases. It will be useful to prove that $\mathtt{W}$ is admissible, but also to prove that it is pre-admissible.

Theorem 10

Let $cs = \gamma \to \gamma'$ be a computing step, let a be an anchor at γ' that does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs.

If $\exists v \in D_a^\gamma$ and $u \in \mathcal{C}_{G_a}(v)$ such that:

- *$\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{c}_u^\gamma \neq \mathtt{c}_v^\gamma$ and u executes $\mathbb{R}_{\mathtt{Lose}}$ during cs, or*
- *$\mathtt{tok}_v^\gamma \in \{\circlearrowleft, \circ\}$ and $\mathtt{c}_u^\gamma \neq \mathtt{c}_v^\gamma$ and u executes $\mathbb{R}_{\mathtt{ReplayD}}$ during cs, or*
- *$\mathtt{tok}_v^\gamma = \uparrow$ and $\mathtt{c}_u^\gamma = \mathtt{c}_v^\gamma$ and u executes $\mathbb{R}_{\mathtt{Fake}}$ or $\mathbb{R}_{\mathtt{ReplayUp}}$ during cs,*

we have $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \prec^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^\gamma, \gamma)$.

Proof: Let us consider such nodes v and u, and one branch $\mathcal{B}$ of G_a such that $v \in D_a^\gamma(\mathcal{B})$. If $v \notin D_a^{\gamma'}(\mathcal{B})$, or if $\mathtt{tok}_v^{\gamma'} \neq \mathtt{tok}_v^\gamma$, or if $\mathtt{tok}_v^\gamma \in \{\bullet, \circlearrowleft, \circ\}$ and one child u of v executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$, then we according to Lemma 18, or Theorems 6 and 8, or Theorem 4, we have

$$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \prec^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^\gamma, \gamma)$$

We can now suppose that $v \in D_a^{\gamma'}(\mathcal{B})$ and $\mathtt{tok}_v^{\gamma'} = \mathtt{tok}_v^\gamma$ and if $\mathtt{tok}_v^\gamma \in \{\bullet, \circlearrowleft, \circ\}$ then no child of v executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$. Depending on which of the three situations has to be considered, Lemma 25, Lemma 26, or Lemma 27 applies and guarantees that $\mathtt{W_{Play}}(v,\gamma') < \mathtt{W_{Play}}(v,\gamma)$. From Theorem 9 we conclude

$$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \prec^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^\gamma, \gamma).$$

■

C.2.8 Fifth component of W: $\mathtt{W}_{\mathtt{Nego}}$ negotiation between one parent and its children

Explanations Only one aspect of the algorithm has not been treated yet: the negotiation between one node v that has the token $\mathtt{tok}_v = \bullet$ and its children $u \in \mathtt{Players}(v)$. This negotiation occurs only if there are several nodes in $\mathtt{Players}(v)$, otherwise v decides that the negotiation is terminated, and executes $\mathbb{R}_{\mathtt{Give}}$ or $\mathbb{R}_{\mathtt{OfferUp}}$ depending on whether there exist some candidates for the token or not. Furthermore, let us remark that if $\mathtt{Players}(v)$ evolves during an execution, whether by an addition (with $\mathbb{R}_{\mathtt{ReplayUp}}$) or a deletion (with $\mathbb{R}_{\mathtt{Lose}}$ for example), then according to Theorems 4 and 10 the weight of the $\mathtt{CD}^o$ which contains v decreases. Thus, we can suppose in the following that we work under the following hypothesis: $\mathtt{Players}(v)$ is constant, and there are at least two nodes in it.

Due to the impact of $\mathtt{PhasePlus}(v)$ on variable $\mathtt{ph}_v$, we cannot guarantee at first sight that the negotiation does not cycle. Indeed, if all the children u of v declare that they cannot give an answer for $\mathtt{ph}_v$, then v decides to restart from $\mathtt{ph}_v = 1$ so it does not eliminate all of its children. But as soon as v can restart, we must prove that it does not restart indefinitely, what would create a livelock in the negotiation phase.

To overcome this problem, we statically determine the first value $\mathtt{ph}_v$ which, when taken by v, will push one child $u \in \mathtt{Players}(v)$ to execute $\mathbb{R}_{\mathtt{Lose}}$. Such values exist since we supposed that there are at least two nodes in $\mathtt{Players}(v)$ and that the identifiers are globally unique. If we prove that ultimately, some node u can only execute $\mathbb{R}_{\mathtt{Lose}}$ and thus makes $\mathtt{W}_{\mathtt{Play}}$ decrease, we prove that the negotiation is livelock-free. Unfortunately, it is not trivial to determine the value of the first incoming value of $\mathtt{ph}_v$ which will induce an execution of $\mathbb{R}_{\mathtt{Lose}}$. Basically, we are looking for the lowest value greater or equal to $\mathtt{ph}_v$ which induces different answers on two children $u \in \mathtt{Players}(v)$.

One first subtlety comes with the fact that, since we work in the framework of self-stabilization, there might not be such greater value. Indeed, if the initial value of $\mathtt{ph}_v$ is different from 1, and if all the values of $\mathtt{ph}_v$ which might differentiate two children $u \in \mathtt{Players}(v)$ are less than this initial value, then the differentiation will not happen before a reset of $\mathtt{ph}_v$. Under such circumstances, v executes a full round of all the possible value for $\mathtt{ph}$, increasing as long as its children u produce answers different from $\mathtt{b}_u = \perp$, and when they all, at the same moment by hypothesis, cannot, v restart from $\mathtt{ph}_v = 1$. Then, v will increase $\mathtt{ph}_v$ until it reaches one value which differentiates two of its children $u \in \mathtt{Players}(v)$.

One other, trickier, subtlety coming from the self-stabilizing framework, is that the registers of the children might be incorrectly initialized. If, due to an incorrect initialization, we miss an occasion to differentiate two children in $\mathtt{Players}(v)$, a similar scheme will happen. In such situation, v increases $\mathtt{ph}_v$ until it reaches the next value which differentiates two of its children of $\mathtt{Players}(v)$. It might even happen that the initial value of $\mathtt{ph}_v$ is the only value which can differentiate any two children of $\mathtt{Players}(v)$. In this situation, node v will execute a full round of all the possible value for $\mathtt{ph}_v$, increasing as long as all of its children produce answers different from $\mathtt{b}_u = \perp$, then restarting from $\mathtt{ph}_v = 1$ until it reaches again the initial value which differentiates its children. On the contrary, an incorrect initialization of the values of $\mathtt{b}_u$ on some node $u \in \mathtt{Players}(v)$ might implies that u executes $\mathbb{R}_{\mathtt{Lose}}$ while it is not supposed to, according to its actual $\mathtt{id}$. That last case will be said to be a false positive.

Incorrect initializations of variable $\mathtt{b}_u$ might only happen at the very beginning of the execution, and might be taken into account only if u does not update it before any action if its parent, *i.e.* if $\neg\mathtt{AnswerPar}(u, v)$. Then, as soon as u takes a step, either it executes $\mathbb{R}_{\mathtt{Lose}}$, either it updates $\mathtt{b}_u$ according to its identifier, and then we will not face any incorrect initialization anymore. Thus, we have to consider the possibilities of wrong initializations only in few cases, basically only for the current value of $\mathtt{ph}_v$ and only if $\neg\mathtt{AnswerPar}(u, v)$. For any other situation,

$\texttt{Bit}_u(\texttt{ph})$ is the adapted value, which describes the value of $\texttt{b}_u$ taken into account during the negotiation.

To summarize, we can distinguish three situations.

- The first, simplest, situation, is when the current value of $\texttt{ph}_v$ actually differentiates at least two children of $\texttt{Players}(v)$. This situation includes false positive, when $\texttt{ph}_v$ is not meant to play this role regarding to identifiers, and excludes situations where an incorrect initialization of the $\texttt{b}_u$ prevents the expected differentiation to actually happen.
- The second situation is when the current value of $\texttt{ph}_v$ does not differentiate two children of $\texttt{Players}(v)$, but there exists a value greater than $\texttt{ph}_v$ which does. In such a situation, we only have to wait for $\texttt{ph}_v$ to increase until it reaches the first such value, and we fall in the previous situation.
- The third and last situation is when neither the current value of $\texttt{ph}_v$, neither any value greater than it, can differentiate two children of $\texttt{Players}(v)$. This situation includes, as an extreme case, the situation where the current value of $\texttt{ph}_v$ is the only one which might differentiate two children of $\texttt{Players}(v)$, but an incorrect initialization of some $\texttt{b}_u$ prevents the differentiation from being effective. In this third situation, we wait for $\texttt{ph}_v$ to increase until the value for which all children of $\texttt{Players}(v)$, at the same moment, answer $\perp$. At this moment, $\texttt{ph}_v$ is set to 1 and then we fall in one of the previous situations.

We want to design a function dealing with as many aspects, or as many rules, of the algorithm as possible. Thus, we intend to define $\texttt{W}_{\texttt{Nego}}$ such that any action involving $\mathbb{R}_{\texttt{maxPos}}$, $\mathbb{R}_{\texttt{NewPh}}$, or $\mathbb{R}_{\texttt{NewBit}}$, makes $\texttt{W}_{\texttt{Nego}}$ decrease. It is pretty natural to treat $\mathbb{R}_{\texttt{NewPh}}$ since it increases (or reduces down to 1) the value of $\texttt{ph}_v$, and thus bring closer to the expected value. Since any two execution of $\mathbb{R}_{\texttt{NewPh}}$ are separated by an execution of $\mathbb{R}_{\texttt{maxPos}}$, which updates $\texttt{b}_v$ from $\perp$ to a value different from $\perp$, we will not have much trouble to use those properties in the design of $\texttt{W}_{\texttt{Nego}}$. Finally, we want $\texttt{W}_{\texttt{Nego}}$ to decrease any time one child $u \in \texttt{Players}(v)$ executes $\mathbb{R}_{\texttt{NewBit}}$. In other words, we must, in a certain way, count the number of children of v which can execute $\mathbb{R}_{\texttt{NewBit}}$. But any execution of $\mathbb{R}_{\texttt{NewPh}}$ recreates the conditions under which all the nodes $u \in \texttt{Players}(v)$ can execute $\mathbb{R}_{\texttt{NewBit}}$, and thus if we simply decrease $\texttt{W}_{\texttt{Nego}}$ by one for the execution of $\mathbb{R}_{\texttt{NewPh}}$, but it increases by $|\texttt{Players}(v)|$ at the same time, we fail. Two options are left to us. First, we can define $\texttt{W}_{\texttt{Nego}}$ as a 2-tuple, the first component focuses on the progress of $\texttt{ph}_v$, and the second one focuses on the decreasing of the number of nodes which can execute $\mathbb{R}_{\texttt{NewBit}}$. The other option is to deal this in one single function, and to give some coefficients to the different components involved. In a certain sense, it boils down to the same as the previous option, but instead of using the lexicographic order, we define manually high coefficients on the first component so that it all works the same with one integer value. To avoid making this proof even more tedious, we chose the second option.

Definitions Before properly defining $\texttt{W}_{\texttt{Nego}}$, let us first formally determine the three different situations which were described above, whether we reached the expected value for $\texttt{ph}_v$, or whether not but there we will reach this value with only increasing $\texttt{ph}_v$, or whether we will have to reset $\texttt{ph}_v$ to 1 before we reach it.

Recall that to decide if the current value of $\texttt{ph}_v$ will differentiate two children $u \in \texttt{Players}(v)$, we cannot simply rely on the values of $\texttt{Bit}_u(\texttt{ph}_v)$. Indeed, we must take into account that the register of $\texttt{b}_v$ and/or $\texttt{b}_u$ might be incorrectly initialized. If this happens, and if the current state of one node u can be interpreted as an answer by its parent, then it does not update it, and thus $\texttt{b}_u$ is the value used to decide what the future of the negotiation is. Otherwise, u has to first execute $\mathbb{R}_{\texttt{NewBit}}$ before anything else, and then erase its previous state and only $\texttt{Bit}_u(\texttt{ph}_v)$

is relevant. Thus, we first formally define what actual value will be taken as the answer by the parent v of u for the current step of the negotiation, which depends for each node u on whether it will produce a new answer to its parent, or not.

$$\texttt{Answer}^\gamma(u,v) = \begin{cases} \texttt{Bit}_u(\texttt{ph}_v^\gamma) & \text{if } \texttt{AnswerPar}^\gamma(u,v) \\ \texttt{b}_u^\gamma & \text{otherwise} \end{cases} \tag{50}$$

Let us now detail the situations in which the current value of $\texttt{ph}_v$ differentiates two children $u \in \texttt{Players}^\gamma(v)$. The first, most natural, situation is when two different nodes will produce different values for $\texttt{b}$ in response to $\texttt{ph}_v^\gamma$. If this happens, then after v executes $\mathbb{R}_{\texttt{maxPos}}$ (or before, if we already have $\texttt{b}_v^\gamma \neq \bot$, at least one of them will be in a situation where it must execute $\mathbb{R}_{\texttt{Lose}}$.

$$\texttt{TwoDiffer}^\gamma(v) \equiv \exists u_1, u_2 \in \texttt{Players}^\gamma(v) : \texttt{Answer}^\gamma(u_1, v) \neq \texttt{Answer}^\gamma(u_2, v) \tag{51}$$

The previous captures almost all the situations where one node is close to execute $\mathbb{R}_{\texttt{Lose}}$. There exists one other, unusual, situation, which is not described here, which is when all the nodes $u \in \texttt{Players}^\gamma(v)$ will produce the same value for $\texttt{ph}_v$, but due to an incorrect initialization of variables, v has already picked a value, which is different from the common value proposed by its children. In this situation, all the children will actually execute $\mathbb{R}_{\texttt{Lose}}$ at very short term.

$$\texttt{OneDiffers}^\gamma(v) \equiv \texttt{b}_v^\gamma \neq \bot \wedge \exists u \in \texttt{Players}^\gamma(v) : \texttt{Answer}^\gamma(u, v) \neq \texttt{b}_v^\gamma \tag{52}$$

We can now define the first of our three cases, which corresponds to the situation where one node $u \in \texttt{Players}^\gamma$ will execute $\mathbb{R}_{\texttt{Lose}}$ before any update of $\texttt{ph}_v$.

$$\texttt{Finished}^\gamma(v) \equiv \texttt{TwoDiffer}^\gamma(v) \vee \texttt{OneDiffers}^\gamma(v) \tag{53}$$

Let us now discuss the two other situations, in which the current value $\texttt{ph}_v^\gamma$ does not differentiate any two children of $\texttt{Players}^\gamma(v)$. We need to determine whether v will reset $\texttt{ph}_v$ to 1 before we reach a value which differentiates two of its children of $\texttt{Players}^\gamma(v)$, or whether not. We could think, at first, that it is sufficient to determine whether there exists one value greater than $\texttt{ph}_v^\gamma$ which differentiates two nodes of $\texttt{Players}^\gamma(v)$ or not. If there is not such greater value, $\texttt{ph}_v$ will indeed be reset to 1 before we terminate the negotiation phase. Yet, it might happen that, although there exists such a greater value, we are nevertheless forced to reset $\texttt{ph}_v$ to 1 before we reach this greater value. Indeed, it could happen that, due to an incorrect initialization of variables $\texttt{b}_u$, all the children $u \in \texttt{Players}^\gamma(v)$ produce $\bot$ as an answer for $\texttt{ph}_v^\gamma$ while there exists a greater value which would differentiate two of them. If this happens, then the execution of $\texttt{PhasePlus}(v)$ will reset $\texttt{ph}_v = 1$. Hopefully, this is can happen only due to incorrect initialization of variables, and thus can only happen for the initial value of $\texttt{ph}_v$. Remark that if only few of the nodes produce $\bot$ as an answer for $\texttt{ph}_v^\gamma$ then we have $\texttt{Finished}^\gamma(v)$. Let us define the predicate $\texttt{ForcedReset}(v)$ which corresponds to configurations where an execution of $\mathbb{R}_{\texttt{NewPh}}$ by v will set $\texttt{ph}_v$ to 1.

$$\texttt{ForcedReset}^\gamma(v) \equiv \forall u \in \texttt{Players}^\gamma(v), \texttt{Answer}^\gamma(u, v) = \bot \tag{54}$$

If $\texttt{ForcedReset}^\gamma(v)$, then we are sure that v will first reset $\texttt{ph}_v = 1$. But if not, we must determine whether there exists one value greater than $\texttt{ph}_v^\gamma$ which differentiates two nodes of $\texttt{Players}^\gamma(v)$ or not. Let us first define $\texttt{Separators}^\gamma(v)$ the set of all the values for $\texttt{ph}_v$ which can differentiate two children $u \in \texttt{Players}^\gamma(v)$. Remark that this set is defined statically: it does not depend at all on the values of $\texttt{ph}_v, \texttt{b}_v, \texttt{ph}_u, \texttt{b}_u$, and only relies on the values of the different identifiers of children of v in $\texttt{Players}^\gamma(v)$. Yet, we consider this set only if $\neg\texttt{Finished}^\gamma(v)$, that

is to say if we must execute $\mathbb{R}_{\texttt{NewPh}}$ at least once before reaching the value that differentiates children of v. By definition, after v executes $\mathbb{R}_{\texttt{NewPh}}$, we necessarily have $\texttt{AnswerPar}(u, v)$ for all children $u \in \texttt{Players}^\gamma(v)$. Thus, in such situations, $\texttt{Bit}_u$ is the relevant tool to decide whether some value for $\texttt{ph}_v$ will differentiate children of v. Remark also that $\texttt{Separators}^\gamma(v)$ only depends on $\texttt{Players}^\gamma(v)$, which will be supposed constant in the following. Consequently, we will suppose $\texttt{Separators}^\gamma(v)$ constant too. Finally, since we are interested in considering the negotiation phase, v has at least two children, and then $\texttt{Separators}^\gamma(v)$ contains at least one value.

$$\texttt{Separators}^\gamma(v) = \{\texttt{ph} \in [0, \lceil \log n \rceil] \mid \exists u_1, u_2 \in \texttt{Players}^\gamma(v) : \texttt{Bit}_{u_1}(\texttt{ph}) \neq \texttt{Bit}_{u_2}(\texttt{ph})\} \quad (55)$$

Let us now define the two other predicates, which correspond to situations where node v will not reset $\texttt{ph}_v$ to 1 before it reaches a value which differentiates two of its children, and to the situation where it will.

$$\texttt{WontReset}^\gamma(v) \equiv \neg\texttt{Finished}^\gamma(v) \wedge \begin{cases} \exists i > \texttt{ph}_v^\gamma : i \in \texttt{Separators}^\gamma(v) \\ \wedge \\ \neg\texttt{ForcedReset}^\gamma(v) \end{cases} \quad (56)$$

$$\texttt{WillReset}^\gamma(v) \equiv \neg\texttt{Finished}^\gamma(v) \wedge \begin{cases} \forall i > \texttt{ph}_v^\gamma, i \notin \texttt{Separators}^\gamma(v) \\ \vee \\ \texttt{ForcedReset}^\gamma(v) \end{cases} \quad (57)$$

Remark that the three predicates `Finished`, `WontReset`, and `WillReset` are mutually exclusive, and that $\forall \gamma \in \Gamma, \forall v \in V, \texttt{Finished}^\gamma(v) \vee \texttt{WontReset}^\gamma(v) \vee \texttt{WillReset}^\gamma(v)$. This justifies the use of these three predicates to distinguish cases in any situation.

Let us now define, for each of these three cases, the value of `ph` which, when reach by $\texttt{ph}_v$, will provoke one execution of $\mathbb{R}_{\texttt{Lose}}$ on one child $u \in \texttt{Players}^\gamma(v)$.

$$\texttt{SepVal}^\gamma(v) = \begin{cases} \texttt{ph}_v^\gamma & \text{if } \texttt{Finished}^\gamma(v) \\ \min\{i \mid i \in \texttt{Separators}^\gamma(v) \wedge i > \texttt{ph}_v^\gamma\} & \text{if } \texttt{WontReset}^\gamma(v) \\ \min\{i \mid i \in \texttt{Separators}^\gamma(v)\} & \text{if } \texttt{WillReset}^\gamma(v) \end{cases} \quad (58)$$

Finally, we are able to define one first aspect of $\texttt{W}_{\texttt{Nego}}$, which is the distance between $\texttt{ph}_v^\gamma$ and $\texttt{SepVal}^\gamma(v)$, which corresponds to the number of executions of $\mathbb{R}_{\texttt{NewPh}}$ required for v. This definition is very simple in both situations $\texttt{Finished}^\gamma(v)$ and $\texttt{WontReset}^\gamma(v)$. It is slightly more difficult when $\texttt{WillReset}^\gamma(v)$ since we must determine how high $\texttt{ph}_v$ will reach before it is reset to 1. Two situations might occur.

If, due to an incorrect initialization of variables, all the children u of $\texttt{Players}^\gamma(v)$ produce $\perp$ as an answer for $\texttt{ph}_v^\gamma$, then the next execution of $\mathbb{R}_{\texttt{NewPh}}$ by v will set $\texttt{ph}_v$ to 1 and then $\texttt{ph}_v^\gamma$ is the maximal value which will be reached before $\texttt{ph}_v$ is reset to 1.

Otherwise, there will be at least one execution of $\mathbb{R}_{\texttt{NewPh}}$ by v before we reach the maximal value, and thus only the functions $\texttt{Bit}_u$ are relevant. In this last case, the maximal value reached before $\texttt{ph}_v$ is reset to 1 is the lowest value such that, when reached, all the children u of $\texttt{Players}^\gamma(v)$ produce $\perp$ as an answer.

$$\texttt{MaxPh}^\gamma(v) = \begin{cases} \texttt{ph}_v^\gamma & \text{if } \texttt{ForcedReset}^\gamma(v) \\ \min\{i > \texttt{ph}_v^\gamma \mid \forall u \in \texttt{Players}^\gamma(v) : \texttt{Bit}_u(i) = \perp\} & \text{if } \neg\texttt{ForcedReset}^\gamma(v) \end{cases} \quad (59)$$

Finally, the definition of $\texttt{SepDist}(v)$ which is the number of executions of $\mathbb{R}_{\texttt{NewPh}}$ v has to make before we have $\texttt{ph}_v = \texttt{SepVal}^\gamma(v)$.

$$\texttt{SepDist}^\gamma(v) = \begin{cases} 0 & \text{if } \texttt{Finished}^\gamma(v) \\ \texttt{SepVal}^\gamma(v) - \texttt{ph}_v^\gamma & \text{if } \texttt{WontReset}^\gamma(v) \\ \texttt{SepVal}^\gamma(v) + (\texttt{MaxPh}^\gamma(v) - \texttt{ph}_v^\gamma) & \text{if } \texttt{WillReset}^\gamma(v) \end{cases} \quad (60)$$

We now formally define the weight of one node v. The goal is to count the number of actions each node, among v and its children in $\texttt{Players}^\gamma(v)$, can take before the execution of $\mathbb{R}_{\texttt{Lose}}$ is necessary.

Node v will execute two actions for each increase of $\texttt{ph}_v$, since it must execute both $\mathbb{R}_{\texttt{NewPh}}$, and $\mathbb{R}_{\texttt{maxPos}}$ for each step. A node $u \in \texttt{Players}^\gamma(v)$ executes $\mathbb{R}_{\texttt{Lose}}$ when $\texttt{b}_v \neq \bot$, which means immediately after one execution of $\mathbb{R}_{\texttt{maxPos}}$, and not of $\mathbb{R}_{\texttt{NewPh}}$. Thus, v executes a sequence of $\mathbb{R}_{\texttt{NewPh}}; \mathbb{R}_{\texttt{maxPos}}$ in this order, which might be preceded by one execution of $\mathbb{R}_{\texttt{maxPos}}$ if necessary.

$$\texttt{ParSteps}^\gamma(v) = 2 \times \texttt{SepDist}^\gamma(v) + \begin{cases} 1 & \text{if } \texttt{b}_v^\gamma = \bot \\ 0 & \text{otherwise} \end{cases} \tag{61}$$

One children $u \in \texttt{Players}^\gamma(v)$ only executes $\mathbb{R}_{\texttt{NewBit}}$, by hypothesis. Every time it executes $\mathbb{R}_{\texttt{NewBit}}$, u sets $\texttt{ph}_u$ at the current value of $\texttt{ph}_v$, which will be updated exactly $\texttt{SepDist}^\gamma(v)$ times. Thus, u executes $\mathbb{R}_{\texttt{NewBit}}$ once for each updated value of $\texttt{ph}_v$, and execute one more action if, in the initial configuration, u has to produce an answer for $\texttt{ph}_v$.

$$\texttt{ChSteps}^\gamma(u,v) = \texttt{SepDist}^\gamma(v) + \begin{cases} 1 & \text{if } \texttt{AnswerPar}^\gamma(u,v) \\ 0 & \text{otherwise} \end{cases} \tag{62}$$

We finally define $\texttt{W}_{\texttt{Nego}}$.

$$\texttt{W}_{\texttt{Nego}}(v,\gamma) = \begin{cases} \texttt{ParSteps}^\gamma(v) + \sum_{u \in \texttt{Players}^\gamma(v)} \texttt{ChSteps}^\gamma(u,v) & \text{if } \texttt{tok}_v^\gamma = \bullet \\ 0 & \text{otherwise} \end{cases} \tag{63}$$

Preliminaries Our overall goal is to prove that $\texttt{W}_{\texttt{Nego}}$ behaves as expected, which is proving that associated with the four previous components, it is pre-admissible, and that in some specific cases, it is a decreasing function. Before we formally establish this, we need to establish some basic, yet numerous, properties which describe how the different predicates and sets we defined above behave during an execution. Indeed, since the definition of $\texttt{W}_{\texttt{Nego}}$ strongly depends on `Finished`, `WontReset`, and `WillReset`, among other things. In this preliminary, we focus on establishing properties of stability for those three predicates, depending on the activation of node v. To achieve this, we first prove some stability properties on other, more basic, predicates, namely `AnswerPar`, `Answer`, `ForcedReset`. Some of those preliminary lemmas will be reused as well in the demonstrations of the next section.

Our goal is to prove that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}}, \texttt{W}_{\texttt{Nego}})$ is pre-admissible. Thus, according to Theorem 9, most of the proof will be done under the hypothesis that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})$ is constant. Actually, this is hypothesis is explicitly used only in the proof of Lemma 29, which states that $\texttt{Players}(v)$ and $\texttt{Separators}(v)$ are constant. In all the other lemmas, this hypothesis on $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})$ is actually used through Lemma 29: we suppose $\texttt{Players}(v)$ and $\texttt{Separators}(v)$ are constant.

Lemma 29

> *Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^\gamma = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v,\gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v,\gamma)$.*
>
> *Then $\texttt{Players}^{\gamma'}(v) = \texttt{Players}^\gamma(v) \wedge \texttt{Separators}^{\gamma'}(v) = \texttt{Separators}^\gamma(v)$.*

Proof: Let us first remark that $\texttt{c}_v^{\gamma'} = \texttt{c}_v^\gamma$ according to the rules. Let us first consider one node $u \in \texttt{Players}^\gamma(v)$, *i.e.* $\texttt{tok}_u^\gamma = \bot \wedge \texttt{c}_u^\gamma \neq \texttt{c}_v^\gamma \wedge \texttt{play}_u^\gamma = \texttt{P}$. If u executes $\mathbb{R}_{\texttt{Lose}}$ then according to Lemma 25, we have $\texttt{W}_{\texttt{Play}}(v,\gamma') < \texttt{W}_{\texttt{Play}}(v,\gamma)$ which cannot happen by hypothesis. If u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ then according to Lemma 16, we have $\texttt{W}_{\texttt{Ch}}(v,\gamma') < \texttt{W}_{\texttt{Ch}}(v,\gamma)$ which cannot happen by hypothesis. Thus, u can only execute $\mathbb{R}_{\texttt{NewBit}}$, and whatever happens, $u \in \texttt{Players}^{\gamma'}(v)$.

Let us now consider one node $u \notin \texttt{Players}^{\gamma}(v)$. If $\texttt{c}_u^{\gamma} = \texttt{c}_v^{\gamma}$ then we can have $u \in \texttt{Players}^{\gamma'}(v)$ only if $\texttt{c}_u^{\gamma'} \neq \texttt{c}_v^{\gamma'}$ which is possible only if u executes $\mathbb{R}_{\texttt{Win}}$ or $\mathbb{R}_{\texttt{FakeWin}}$ but then according to Lemma 16, we have $\texttt{W}_{\texttt{Ch}}(v, \gamma') < \texttt{W}_{\texttt{Ch}}(v, \gamma)$ which cannot happen by hypothesis. Otherwise, if $\texttt{tok}_u^{\gamma} \neq \bot$ then we can have $u \in \texttt{Players}^{\gamma'}(v)$ only if $\texttt{tok}_u^{\gamma'} = \bot$, which is possible only if u executes $\mathbb{R}_{\texttt{Return}}$, but then the execution of $\texttt{Drop}(u)$ implies that $\texttt{play}_u^{\gamma'} = \texttt{F}$ and thus $u \notin \texttt{Players}^{\gamma'}(v)$. Finally, if $\texttt{c}_u^{\gamma} \neq \texttt{c}_v^{\gamma}$ and $\texttt{tok}_u^{\gamma} = \bot$, then $\texttt{play}_u^{\gamma} \neq \texttt{P}$. We can have $u \in \texttt{Players}^{\gamma'}(v)$ only if $\texttt{play}_u^{\gamma'} = \texttt{P}$, which is possible only if u executes $\mathbb{R}_{\texttt{ReplayD}}$ or $\mathbb{R}_{\texttt{ReplayUp}}$. Due to its parent v, u cannot execute $\mathbb{R}_{\texttt{ReplayD}}$, and if it executes $\mathbb{R}_{\texttt{ReplayD}}$, then according to Lemma 16, we have $\texttt{W}_{\texttt{Ch}}(v, \gamma') < \texttt{W}_{\texttt{Ch}}(v, \gamma)$ which cannot happen by hypothesis. As a consequence, in any case, $u \notin \texttt{Players}^{\gamma'}(v)$.

We just established that $u \in \texttt{Players}^{\gamma'}(v) \iff u \in \texttt{Players}^{\gamma}(v)$, which means that $\texttt{Players}^{\gamma'}(v) = \texttt{Players}^{\gamma}(v)$. Since $\texttt{Separators}^{\gamma}(v)$ only depends on $\texttt{Players}^{\gamma}(v)$, we can conclude that $\texttt{Separators}^{\gamma'}(v) = \texttt{Separators}^{\gamma}(v)$ as well. ■

From now on, since all the following lemmas of this section are under, at least, the hypothesis of Lemma 29, we allow ourselves to simply write $\texttt{Players}(v)$ and $\texttt{Separators}(v)$ without referring to the configuration in which the set is evaluated.

Lemma 30 proves that, unless node v asks for one new information, which concerns one new value for ph, then the value which will be taken as an answer for the negotiation does not evolve. It partly justifies the definition of Answer.

Lemma 30

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^{\gamma} = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

If v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs then $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^{\gamma}(u, v)$.

Proof: Let us consider $u \in \texttt{Players}(v)$. If u is activated, then according to Theorems 4 and 10, u executes $\mathbb{R}_{\texttt{NewBit}}$. Therefore, $\texttt{OkNeg}^{\gamma}(u)$ so v is the only parent of u with $\texttt{tok}_v \notin \{\bot, \uparrow, \downarrow\}$, and, on the other hand, $\texttt{AnswerPar}^{\gamma}(u, v)$. Therefore, by definition of the action $\texttt{Announce}(u)$, $\texttt{Answer}^{\gamma'}(u, v) = \texttt{b}_u^{\gamma'} = \texttt{Bit}_u(\texttt{ph}_v^{\gamma}) = \texttt{Answer}^{\gamma}(u, v)$. Suppose now that u is not activated during cs. If v is not activated either, then we obviously have $\texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^{\gamma}(u, v)$. Let us rather suppose that v executes $\mathbb{R}_{\texttt{maxPos}}$ during cs.

According to Lemma 20, we have $\neg\texttt{AnswerPar}^{\gamma}(u, v)$, so $\texttt{Answer}^{\gamma}(u, v) = \texttt{b}_u^{\gamma}$, and since u is not activated during cs we even have $\texttt{b}_u^{\gamma'} = \texttt{b}_u^{\gamma}$. By definition, we have $\texttt{Synch}^{\gamma}(v)$ so $\texttt{ph}_u^{\gamma} = \texttt{ph}_v^{\gamma}$, which implies, according to $\texttt{NextPlay}(v)$, that $\texttt{ph}_u^{\gamma'} = \texttt{ph}_v^{\gamma'}$. Since $\texttt{b}_v^{\gamma'} \neq \bot$, we have $\neg\texttt{AnswerPar}^{\gamma'}(u, v)$, so $\texttt{Answer}^{\gamma'}(u, v) = \texttt{b}_u^{\gamma'} = \texttt{b}_u^{\gamma} = \texttt{Answer}^{\gamma}(u, v)$. ■

Lemma 32 proves that once the predicate Finished evaluates to true, then it does in all the following configurations too. In order to prove it, we first establish Lemma 31 which proves that when we have $\texttt{Finished}^{\gamma}(v)$, node v cannot execute $\mathbb{R}_{\texttt{NewPh}}$, which justifies the name of this predicate. Lemma 32 is the first fundamental lemma which will be helpful in the next section.

Lemma 31

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^{\gamma} = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

Then $\texttt{Finished}^{\gamma}(v) \Rightarrow \neg(\texttt{Synch}^{\gamma}(v) \wedge \texttt{PhComplete}^{\gamma}(v))$.

Proof: Suppose that $\texttt{Finished}^{\gamma}(v) \wedge \texttt{Synch}^{\gamma}(v)$ and prove that $\neg\texttt{PhComplete}^{\gamma}(v)$. Whether we have $\texttt{TwoDiffer}^{\gamma}(v)$ or $\texttt{OneDiffers}^{\gamma}(v)$, there exists $u \in \texttt{Players}(v) : \texttt{Answer}^{\gamma}(u, v) \neq \texttt{b}_v^{\gamma}$. If $\texttt{AnswerPar}^{\gamma}(u, v)$ then according to Lemma 20, $\neg\texttt{PhComplete}^{\gamma}(v)$. Otherwise, $\neg\texttt{AnswerPar}^{\gamma}(u, v)$ and then $\texttt{b}_u^{\gamma} \neq \texttt{b}_v^{\gamma}$ and once again, $\neg\texttt{PhComplete}^{\gamma}(v)$. ■

Lemma 32

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^\gamma = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

Then $(\texttt{TwoDiffer}^\gamma(v) \Rightarrow \texttt{TwoDiffer}^{\gamma'}(v)) \wedge (\texttt{OneDiffers}^\gamma(v) \Rightarrow \texttt{OneDiffers}^{\gamma'}(v))$, and thus $\texttt{Finished}^\gamma(v) \Rightarrow \texttt{Finished}^{\gamma'}(v)$.

Proof: If $\texttt{TwoDiffer}^\gamma(v)$ then $\texttt{Finished}^\gamma(v)$ so according to Lemma 31, v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs, and thus according to Lemma 30, $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^\gamma(u, v)$. Thus, let us consider $u_1, u_2 \in \texttt{Players}(v)$ such that $\texttt{Answer}^\gamma(u_1, v) \neq \texttt{Answer}^\gamma(u_2, v)$. Immediately, we have $\texttt{Answer}^\gamma(u_1, v) \neq \texttt{Answer}^\gamma(u_2, v)$ and thus $\texttt{TwoDiffer}^{\gamma'}(v)$.

Similarly, if $\texttt{OneDiffers}^\gamma(v)$, then $\texttt{Finished}^\gamma(v)$ so according to Lemma 31, u does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs, and thus $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^\gamma(u, v)$. By definition of $\texttt{OneDiffers}^\gamma(v)$, we have $\texttt{b}_v^\gamma \neq \bot$, so v does not execute $\mathbb{R}_{\texttt{maxPos}}$ during cs, which implies that v is not activated during cs. Consequently, $\texttt{b}_v^{\gamma'} = \texttt{b}_v^\gamma \neq \bot$, and since $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^\gamma(u, v)$, then $\texttt{OneDiffers}^{\gamma'}(v)$.

Since $\texttt{Finished} = \texttt{TwoDiffer} \vee \texttt{OneDiffers}$, we deduce $\texttt{Finished}^\gamma(v) \Rightarrow \texttt{Finished}^{\gamma'}(v)$. ■

The two following lemmas prove that, when v does not execute $\mathbb{R}_{\texttt{NewPh}}$, then the predicates `ForcedReset` and `Finished` remain constant.

Lemma 33 proves that, as long as v does not update $\texttt{ph}_v$, then whether the next execution of $\mathbb{R}_{\texttt{NewPh}}$ will reset $\texttt{ph}_v = 1$ or not, $\texttt{ForcedReset}(v)$ keeps the same boolean value. It, partly, justifies the definition of `ForcedReset`.

Lemma 33

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^\gamma = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

If v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs then $\texttt{ForcedReset}^{\gamma'}(v) \iff \texttt{ForcedReset}^\gamma(v)$.

Proof: Since `ForcedReset` only depends on `Answer`, this is a direct consequence of Lemma 30 ■

Lemma 34 establishes that, as long as v does not update $\texttt{ph}_v$, then whether we have reached the final value for $\texttt{ph}_v$ or not, $\texttt{Finished}(v)$ keeps the same boolean value. In a certain sense, it specifies Lemma 32.

Lemma 34

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^\gamma = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

If v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs then $\texttt{Finished}^{\gamma'}(v) \iff \texttt{Finished}^\gamma(v)$.

Proof: Since v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs, $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^\gamma$. According to Lemma 30, we have $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^\gamma(u, v)$.

If v is not activated during cs, then $\texttt{b}_v^{\gamma'} = \texttt{b}_v^\gamma$ and thus we have $\texttt{Finished}^{\gamma'}(v) \iff \texttt{Finished}^\gamma(v)$. Now suppose that v executes $\mathbb{R}_{\texttt{maxPos}}$ during cs, which means $\texttt{b}_v^\gamma = \bot$ and $\texttt{b}_v^{\gamma'} \neq \bot$.

Since $\texttt{b}_v^{\gamma'} \neq \bot$, $\texttt{OneDiffers}^{\gamma'}(v) \equiv \exists u \in \texttt{Players}(v) : \texttt{Answer}^{\gamma'}(u, v) \neq \texttt{b}_v^{\gamma'}$. But since $\exists u \in \texttt{Players}(v) : \texttt{Answer}^{\gamma'}(v) = \texttt{b}_v^{\gamma'}$ due to $\texttt{MaxPos}(v)$, we have $\texttt{OneDiffers}^{\gamma'}(v) \equiv \exists u_1, u_2 \in \texttt{Players}(v) : \texttt{Answer}^{\gamma'}(u_1, v) \neq \texttt{Answer}^{\gamma'}(u_2, v)$. Thus, $\texttt{OneDiffers}^{\gamma'}(v) \iff \texttt{TwoDiffer}^{\gamma'}(v)$. Furthermore, since $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^\gamma(u, v)$ we have $\texttt{TwoDiffer}^\gamma(v) \iff \texttt{TwoDiffer}^{\gamma'}(v)$.

Finally, since $\mathtt{b}_v^\gamma = \bot$ we have $\neg\mathtt{OneDiffers}^\gamma(v)$, so

$$\begin{aligned}\mathtt{Finished}^\gamma(v) &\iff \mathtt{TwoDiffer}^\gamma(v)\\ &\iff \mathtt{TwoDiffer}^{\gamma'}(v)\\ &\iff \mathtt{OneDiffers}^{\gamma'}(v) \vee \mathtt{TwoDiffer}^{\gamma'}(v)\\ &\iff \mathtt{Finished}^{\gamma'}(v).\end{aligned}$$

■

The four following lemmas give information on how an execution of $\mathbb{R}_{\mathtt{NewPh}}$ might alter some of the predicates we consider here, or establish relations between them. Lemma 35 proves that one node v which executes $\mathbb{R}_{\mathtt{NewPh}}$ resets $\mathtt{ph}_v = 1$ if and only if it was actually detected by `ForcedReset` before its action.

Lemma 35

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

If v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs then $\mathtt{ph}_v^{\gamma'} = 1 \iff \mathtt{ForcedReset}^\gamma(v)$.

Proof: According to Lemma 20, $\forall u \in \mathtt{Players}(v), \neg\mathtt{AnswerPar}^\gamma(u, v)$. Consequently, $\forall u \in \mathtt{Players}(v), \mathtt{Answer}^\gamma(u, v) = \mathtt{b}_u^\gamma$. Thus, $\mathtt{ForcedReset}^\gamma(v) \equiv \forall u \in \mathtt{Players}(v), \mathtt{b}_u^\gamma = \bot$, which is exactly the condition which decides whether $\mathtt{ph}_v$ is set to 1 by $\mathtt{PhasePlus}(v)$. ■

Lemma 36 states that if node v verifies `WontReset`, then it will not reset $\mathtt{ph}_v = 1$, as we expect.

Lemma 36

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

If $\mathtt{WontReset}^\gamma(v)$ then v does not reset $\mathtt{ph}_v = 1$ during cs.

Proof: If v does not execute $\mathbb{R}_{\mathtt{NewPh}}$ during cs, then this result is immediate. If v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs, then since $\mathtt{WontReset}^\gamma(v)$, we have $\neg\mathtt{ForcedReset}^\gamma(v)$, and thus according to Lemma 35, v does not reset $\mathtt{ph}_v = 1$ during cs. ■

Lemma 37 states that if v executes $\mathbb{R}_{\mathtt{NewPh}}$, *i.e.* if it asks for one new information to its children, then all of them will have to produce a new answer, so after this we can trust the values of $\mathtt{Bit}_u$ in the negotiation process. Associated to Lemma 30 it fully justifies the definition of `Answer`.

Lemma 37

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

If v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs then $\forall u \in \mathtt{Players}(v), \mathtt{AnswerPar}^{\gamma'}(u, v)$.

Proof: Let us consider one node $u \in \mathtt{Players}(v)$. According to Lemma 20, u is not activated during cs. Furthermore, since v executes $\mathbb{R}_{\mathtt{NewPh}}$, we deduce $\mathtt{ph}_u^{\gamma'} = \mathtt{ph}_u^\gamma = \mathtt{ph}_v^\gamma$ and $\mathtt{b}_u^{\gamma'} = \mathtt{b}_u^\gamma = \mathtt{b}_v^\gamma$. Two cases must be considered.

If $\mathtt{b}_u^\gamma \neq \bot$ then $\mathtt{PhasePlus}(v)$ updates $\mathtt{ph}_v^{\gamma'} = \mathtt{ph}_v^\gamma + 1 \neq \mathtt{ph}_v^\gamma = \mathtt{ph}_u^{\gamma'}$, which reduces to $\mathtt{ph}_u^{\gamma'} \neq \mathtt{ph}_v^{\gamma'}$ and consequently $\mathtt{AnswerPar}^{\gamma'}(u, v)$.

Otherwise, if $\mathtt{b}_u^\gamma = \bot$, since $\forall u' \in \mathtt{Players}(v), \mathtt{b}_{u'}^\gamma = \mathtt{b}_v$, we have $\forall u' \in \mathtt{Players}(v), \mathtt{b}_{u'}^\gamma = \bot$, so $\mathtt{PhasePlus}(v)$ sets $\mathtt{ph}_v^{\gamma'} = 1$ and $\mathtt{b}_v^{\gamma'} = \bot$. Since we also have $\mathtt{b}_u^{\gamma'} = \bot$, we conclude $\mathtt{AnswerPar}^{\gamma'}(u, v)$. ■

Lemma 38 states that if one node v executes $\mathbb{R}_{\texttt{NewPh}}$, then we have reached the value on which the negotiation is finished if and only if this value belongs to the set $\texttt{Separators}(v)$. Associated with Lemmas 31 and 34 it fully specifies the situations where a computing step leads to a configuration which satisfies `Finished`.

Lemma 38

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^\gamma = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

If v executes $\mathbb{R}_{\texttt{NewPh}}$ during cs then $\texttt{Finished}^{\gamma'}(v) \iff \texttt{ph}_v^{\gamma'} \in \texttt{Separators}(v)$.

Proof: Remark first that $\texttt{b}_v^{\gamma'} = \bot$, and thus $\neg\texttt{OneDiffers}^{\gamma'}(v)$.

By definition $\forall u \in \texttt{Players}(v), \texttt{ph}_u^\gamma = \texttt{ph}_v^\gamma \wedge \texttt{b}_u^\gamma = \texttt{b}_v^\gamma$. According to Lemma 20, children u of v are not activated during cs, so $\forall u \in \texttt{Players}(v), \texttt{ph}_u^{\gamma'} = \texttt{ph}_v^\gamma \wedge \texttt{b}_u^{\gamma'} = \texttt{b}_v^\gamma$. Furthermore, according to Lemma 37, $\forall u \in \texttt{Players}(v), \texttt{AnswerPar}^{\gamma'}(v)$. Consequently, $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Bit}_u(\texttt{ph}_v^{\gamma'})$. Since $\neg\texttt{OneDiffers}^{\gamma'}(v)$, we obtain:

$$\begin{array}{rcl} \texttt{Finished}^{\gamma'}(v) & \iff & \texttt{TwoDiffer}^{\gamma'}(v) \\ & \iff & \exists u_1, u_2 : \texttt{Bit}_{u_1}(\texttt{ph}_v^{\gamma'}) \neq \texttt{Bit}_{u_1}(\texttt{ph}_v^{\gamma'}) \\ & \iff & \texttt{ph}_v^{\gamma'} \in \texttt{Separators}(v). \end{array}$$

■

The two following lemmas, the last of this section, and establish how situations where $\neg\texttt{Finished}(v)$ can evolve.

Lemma 39 states that if we are in a configuration such that we will reach the terminal value for $\texttt{ph}_v$ without resetting $\texttt{ph}_v = 1$, then it remains true. Associated with Lemmas 34 and 38, it fully describes what configuration is reached after a computing step which starts with $\texttt{WontReset}(v)$.

Lemma 39

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^\gamma = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

Then $\texttt{WontReset}^\gamma(v) \Rightarrow \texttt{WontReset}^{\gamma'}(v) \vee \texttt{Finished}^{\gamma'}(v)$.

Proof: Recall that $\texttt{WontReset}^\gamma(v)$ implies $\exists i > \texttt{ph}_v^\gamma : i \in \texttt{Separators}(v)$ and $\neg\texttt{ForcedReset}^\gamma(v)$.

Suppose first that v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs. According to Lemma 33, we have $\neg\texttt{ForcedReset}^{\gamma'}(v)$. Since $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^\gamma$, we also have $\exists i > \texttt{ph}_v^{\gamma'} : i \in \texttt{Separators}(v)$. Consequently, $\texttt{WontReset}^{\gamma'}(v) \vee \texttt{Finished}^{\gamma'}(v)$.

Now suppose that v executes $\mathbb{R}_{\texttt{NewPh}}$ during cs. According to Lemma 36, $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^\gamma + 1$, and, by definition, $\texttt{b}_v^{\gamma'} = \bot$.

If $\texttt{ph}_v^{\gamma'} \in \texttt{Separators}(v)$ then according to Lemma 38 $\texttt{Finished}^{\gamma'}(v)$ and thus we obtain the desired result.

Let us now suppose that $\texttt{ph}_v^{\gamma'} \notin \texttt{Separators}(v)$. By definition of $\texttt{WontReset}^\gamma(v)$, we have $\exists i > \texttt{ph}_v^\gamma : i \in \texttt{Separators}(v)$. Since $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^\gamma + 1 \notin \texttt{Separators}(v)$, we deduce $\exists i > \texttt{ph}_v^{\gamma'} : i \in \texttt{Separators}(v)$. We now finish the proof by establishing $\neg\texttt{ForcedReset}^{\gamma'}(v)$.

Since $\exists i > \texttt{ph}_v^{\gamma'} : i \in \texttt{Separators}(v)$, we deduce $\exists u_1, u_2 \in \texttt{Players}(v) : \texttt{Bit}_{u_1}(i) \neq \texttt{Bit}_{u_2}(i)$. At least one of them, let us say u, is such that $\texttt{Bit}_u(i) \neq \bot$. By definition of $\texttt{Bit}_u$, since $\texttt{ph}_v^{\gamma'} < i$, $\texttt{Bit}_u(\texttt{ph}_v^{\gamma'}) \neq \bot$, and by Lemma 37, we deduce $\texttt{Answer}^{\gamma'}(u, v) \neq \bot$. Thus, $\neg\texttt{ForcedReset}^{\gamma'}(v)$, and finally $\texttt{WontReset}^{\gamma'}(v) \vee \texttt{Finished}^{\gamma'}(v)$. ■

Lemma 40 states that if we are in a configuration such that we need to reset $\texttt{ph}_v = 1$ before reaching the terminal value for $\texttt{ph}_v$, then this remains true after a computing step, unless we do

reset $\mathtt{ph}_v = 1$ during this computing step. Associated with Lemmas 35 and 38, it fully describes what configuration is reached after a computing step which starts with $\mathtt{WillReset}(v)$.

Lemma 40

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

If $\mathtt{WillReset}^\gamma(v)$, we have:

$\neg\mathtt{WillReset}^{\gamma'}(v) \iff v$ executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs and $\mathtt{ph}_v^{\gamma'} = 1$.

Proof: Suppose that $\mathtt{WillReset}^\gamma(v)$, *i.e.* $\neg\mathtt{Finished}^\gamma(v) \wedge (\forall i > \mathtt{ph}_v^\gamma, i \notin \mathtt{Separators}(v) \vee \mathtt{ForcedReset}^\gamma(v))$.

If v does not execute $\mathbb{R}_{\mathtt{NewPh}}$ during cs then according to Lemma 34, $\neg\mathtt{Finished}^{\gamma'}(v)$, and according to Lemma 33, $\mathtt{ForcedReset}^{\gamma'}(v) \iff \mathtt{ForcedReset}^\gamma(v)$. Finally, since $\mathtt{ph}_v^{\gamma'} = \mathtt{ph}_v^\gamma$, we have $\forall i > \mathtt{ph}_v^{\gamma'}, i \notin \mathtt{Separators}(v) \iff \forall i > \mathtt{ph}_v^\gamma, i \notin \mathtt{Separators}(v)$. Consequently, we have $\mathtt{WillReset}^{\gamma'}(v)$.

Suppose now that v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs. According to Lemma 37 $\forall u \in \mathtt{Players}(v), \mathtt{AnswerPar}^{\gamma'}(u, v)$ which implies $\forall u \in \mathtt{Players}(u, v), \mathtt{Answer}^{\gamma'}(u, v) = \mathtt{Bit}_u(\mathtt{ph}_v^{\gamma'})$.

Let us first prove that if $\mathtt{ph}_v^{\gamma'} \neq 1$ then $\mathtt{WillReset}^{\gamma'}(v)$. Remark that $\neg\mathtt{ForcedReset}^\gamma(v)$. Indeed, according to Lemma 20 we have $\forall u \in \mathtt{Players}(v), \neg\mathtt{AnswerPar}^\gamma(u, v)$, and thus if $\mathtt{ForcedReset}^\gamma(v)$ then $\forall u \in \mathtt{Players}(v), \mathtt{b}_u^\gamma = \bot$ and thus the execution of $\mathtt{PhasePlus}(v)$ set $\mathtt{ph}_v$ to 1. But now, $\mathtt{WillReset}^\gamma(v) \iff \neg\mathtt{Finished}^{\gamma'}(v) \wedge \forall i > \mathtt{ph}_v^\gamma, i \notin \mathtt{Separators}(v)$. Thus, $\mathtt{ph}_v^{\gamma'} \notin \mathtt{Separators}(v)$, so according to Lemma 38, $\neg\mathtt{Finished}^{\gamma'}(v)$. Furthermore, we have $\forall i > \mathtt{ph}_v^{\gamma'}, i \notin \mathtt{Separators}(v)$, and thus $\mathtt{WillReset}^{\gamma'}(v)$.

Let us now prove that if $\mathtt{ph}_v^{\gamma'} = 1$ then $\neg\mathtt{WillReset}^{\gamma'}(v)$. If $1 \in \mathtt{Separators}(v)$ then according to Lemma 38, $\mathtt{Finished}^{\gamma'}(v)$ and thus $\neg\mathtt{WillReset}^{\gamma'}(v)$. Otherwise, $1 \notin \mathtt{Separators}(v)$, and since $\mathtt{Separators}(v) \neq \emptyset, \exists i > \mathtt{ph}_v^{\gamma'} : i \in \mathtt{Separators}(v)$. Let us now prove that $\neg\mathtt{ForcedReset}^{\gamma'}(v)$ to finish the proof. Since v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs, $\mathtt{ForcedReset}^{\gamma'}(v) \iff \forall u \in \mathtt{Players}(v), \mathtt{Bit}_u(1) = \bot$, which cannot be by hypothesis on the identifiers. Thus, $\exists i > \mathtt{ph}_v^{\gamma'} : i \in \mathtt{Separators}(v) \wedge \neg\mathtt{ForcedReset}^{\gamma'}(v)$ so $\mathtt{Finished}^{\gamma'}(v) \vee \mathtt{WontReset}^{\gamma'}(v)$. ■

Proofs In this section, we finally address the function $\mathtt{W_{Nego}}$, the lemmas established in the previous section are going to be extremely useful.

One first step is to establish that what we defined as the last value for $\mathtt{ph}_v$ during the negotiation phase, $\mathtt{SepVal}(v)$, is consistent through an execution. We prove in Lemma 42 that this value remains constant. Let us first prove Lemma 41 which states that unless we reset $\mathtt{ph}_v = 1$, then the maximum value for $\mathtt{ph}_v$ which we will reach before the reset remains constant as well.

Lemma 41

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

If v does not reset $\mathtt{ph}_v = 1$ during cs, we have $\mathtt{MaxPh}^{\gamma'}(v) = \mathtt{MaxPh}^\gamma(v)$.

Proof: If v does not execute $\mathbb{R}_{\mathtt{NewPh}}$ during cs then $\mathtt{ph}_v^{\gamma'} = \mathtt{ph}_v^\gamma$, and according to Lemma 33 $\mathtt{ForcedReset}^{\gamma'}(v) \iff \mathtt{ForcedReset}^\gamma(v)$, and thus $\mathtt{MaxPh}^{\gamma'}(v) = \mathtt{MaxPh}^\gamma(v)$.

Suppose now that v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs. By hypothesis, $\mathtt{ph}_v$ is not reset to 1 during cs, so $\mathtt{ph}_v^{\gamma'} = \mathtt{ph}_v^\gamma + 1$, and according to Lemma 35 we also have $\neg\mathtt{ForcedReset}^\gamma(v)$. Thus, we deduce $\mathtt{MaxPh}^\gamma(v) = \min\{i \geq \mathtt{ph}_v^\gamma \mid \forall u \in \mathtt{Players}(v), \mathtt{Bit}_u(i) = \bot\}$. On the other hand, since v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs, we know by Lemma 37 that $\forall u \in \mathtt{Players}(v), \mathtt{AnswerPar}^{\gamma'}(u, v)$, which implies $\forall u \in \mathtt{Players}(v), \mathtt{Answer}^{\gamma'}(u, v) = \mathtt{Bit}_u(\mathtt{ph}_v^{\gamma'})$. Thus, $\mathtt{ForcedReset}^{\gamma'}(v) \equiv \forall u \in \mathtt{Players}(v), \mathtt{Bit}_u(\mathtt{ph}_v^{\gamma'}) = \bot$, which is equivalent to $\mathtt{MaxPh}^\gamma(v) = \mathtt{ph}_v^{\gamma'}$.

If $\texttt{ForcedReset}^{\gamma'}(v)$ then immediately, we conclude $\texttt{MaxPh}^{\gamma'}(v) = \texttt{MaxPh}^{\gamma}(v)$. Let us suppose $\neg\texttt{ForcedReset}^{\gamma'}(v)$, which implies $\texttt{MaxPh}^{\gamma'}(v) = \min\{i \geq \texttt{ph}_v^{\gamma'} \mid \forall u \in \texttt{Players}(v), \texttt{Bit}_u(i) = \bot\}$. But we also know $\neg(\forall u \in \texttt{Players}(v), \texttt{Bit}_u(\texttt{ph}_v^{\gamma'}) = \bot)$, so we deduce $\texttt{MaxPh}^{\gamma'}(v) = \min\{i \geq \texttt{ph}_v^{\gamma'} - 1 \mid \forall u \in \texttt{Players}(v), \texttt{Bit}_u(i) = \bot\}$, and as a consequence, $\texttt{MaxPh}^{\gamma'}(v) = \texttt{MaxPh}^{\gamma}(v)$. ■

Lemma 42

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\texttt{tok}_v^{\gamma} = \bullet$ and $\texttt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma') = (\texttt{W}_{\texttt{Ch}}, \texttt{W}_{\texttt{Er}}, \texttt{W}_{\texttt{Circ}}, \texttt{W}_{\texttt{Play}})(v, \gamma)$.

Then $\texttt{SepVal}^{\gamma'}(v) = \texttt{SepVal}^{\gamma}(v)$.

Proof: Let us distinguish three cases:

- If $\texttt{Finished}^{\gamma}(v)$, then according to Lemma 31, v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs, and thus $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^{\gamma}$. Furthermore, according to Corollary 32, $\texttt{Finished}^{\gamma'}(v)$. Thus, we have $\texttt{SepVal}^{\gamma'}(v) = \texttt{ph}_v^{\gamma'} = \texttt{ph}_v^{\gamma} = \texttt{SepVal}^{\gamma}(v)$.
- If $\texttt{WontReset}^{\gamma}(v)$, two cases must be considered.
  If v does not execute $\mathbb{R}_{\texttt{NewPh}}$ during cs, then $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^{\gamma}$, and according to Lemma 34, $\neg\texttt{Finished}^{\gamma'}(v)$, and according to Lemma 30, $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Answer}^{\gamma}(u, v)$. Thus, we have $\texttt{WontReset}^{\gamma'}(v)$, and thus $\texttt{SepVal}^{\gamma'}(v) = \texttt{SepVal}^{\gamma}(v)$.
  If v executes $\mathbb{R}_{\texttt{NewPh}}$ during cs, then by definition of $\texttt{WontReset}^{\gamma}(v)$ the execution of $\texttt{PhasePlus}(v)$ sets $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^{\gamma}+1$ and $\texttt{b}_v^{\gamma'} = \bot$. According to Lemma 37, $\forall u \in \texttt{Players}(v), \texttt{AnswerPar}^{\gamma'}(u, v)$, which implies $\forall u \in \texttt{Players}(v), \texttt{Answer}^{\gamma'}(u, v) = \texttt{Bit}_u(\texttt{ph}_v^{\gamma'})$.
  - If $\texttt{ph}_v^{\gamma'} \in \texttt{Separators}(v)$, then, by definition, $\texttt{SepVal}^{\gamma}(v) = \texttt{ph}_v^{\gamma'}$. Furthermore, according to Lemma 38, $\texttt{Finished}^{\gamma'}(v)$, and thus $\texttt{SepVal}^{\gamma'}(v) = \texttt{ph}_v^{\gamma'}$. This prove $\texttt{SepVal}^{\gamma'}(v) = \texttt{SepVal}^{\gamma}(v)$.
  - If $\texttt{ph}_v^{\gamma'} \notin \texttt{Separators}(v)$ then according to Lemmas 39 and 38 $\texttt{WontReset}^{\gamma'}(v)$. Furthermore, since $\texttt{ph}_v^{\gamma'} = \texttt{ph}_v^{\gamma}+1 \notin \texttt{Separators}(v)$, we deduce $\min\{i \mid i \in \texttt{Separators}(v) \wedge i > \texttt{ph}_v^{\gamma}\} = \min\{i \mid i \in \texttt{Separators}(v) \wedge i > \texttt{ph}_v^{\gamma'}\}$. In other words, $\texttt{SepVal}^{\gamma'}(v) = \texttt{SepVal}^{\gamma}(v)$
- If $\texttt{WillReset}^{\gamma}(v)$, let us consider two cases.
  If v executes $\mathbb{R}_{\texttt{NewPh}}$ and sets $\texttt{ph}_v^{\gamma'} = 1$, then according to Lemma 40, $\texttt{Finished}^{\gamma'}(v) \vee \texttt{WontReset}^{\gamma'}(v)$, and according to Lemma 38, $\texttt{Finished}^{\gamma'}(v) \iff 1 \in \texttt{Separators}(v)$. Thus, if $1 \in \texttt{Separators}(v)$ then $\texttt{SepVal}^{\gamma}(v) = 1$ and $\texttt{SepVal}^{\gamma'}(v) = \texttt{ph}_v^{\gamma'} = 1$, and thus $\texttt{SepVal}^{\gamma'}(v) = \texttt{SepVal}^{\gamma}(v)$. Otherwise, if $1 \notin \texttt{Separators}(v)$, then $\neg\texttt{Finished}^{\gamma'}(v)$, so $\texttt{WontReset}^{\gamma'}(v)$. We have

$$\begin{aligned}\texttt{SepVal}^{\gamma}(v) &= \min\{i \mid i \in \texttt{Separators}(v)\}\\ &= \min\{i \mid i \in (\texttt{Separators}(v) \setminus \{1\})\}\\ &= \min\{i \mid i \in \texttt{Separators}(v) \wedge i > \texttt{ph}_v^{\gamma'}\}\\ &= \texttt{SepVal}^{\gamma'}(v)\end{aligned}$$

  and thus $\texttt{SepVal}^{\gamma'}(v) = \texttt{SepVal}^{\gamma}(v)$
  Let us now suppose that v does not execute $\mathbb{R}_{\texttt{NewPh}}$, or that, if it does, then it does not set $\texttt{ph}_v = 1$. According to Lemma 40, $\texttt{WillReset}^{\gamma'}(v)$, and we deduce $\texttt{SepVal}^{\gamma'}(v) = \min\{i \mid i \in \texttt{Separators}(v)\} = \texttt{SepVal}^{\gamma}(v)$. ■

From now on, for all the lemmas of this section which are stated under, at least, the hypotheses of Lemma 42, which by the way are the same as the hypothesis of Lemma 29, we allow ourselves to simply write $\texttt{SepVal}(v)$ without referring to the configuration in which this function is evaluated.

This being established, we are going to follow the different components of $\mathtt{W_{Nego}}$ introduced for its definition: SepDist, ParSteps, ChSteps, and finished $\mathtt{W_{Nego}}$ itself. For each of those four functions, we prove that it corresponds to a non-increasing function, and for each of them we present the cases where we can guarantee its decrease.

Lemma 43

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.
Then $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^{\gamma}(v)$.

Proof: Let us consider three cases.

- If $\mathtt{Finished}^\gamma(v)$ then according to Corollary 32, $\mathtt{Finished}^{\gamma'}(v)$ and thus $\mathtt{SepDist}^{\gamma'}(v) = \mathtt{SepDist}^{\gamma}(v) = 0$.
- If $\mathtt{WontReset}^\gamma(v)$, then $\mathtt{SepDist}^\gamma(v) = \mathtt{SepVal}(v) - \mathtt{ph}_v^\gamma$. If $\mathtt{Finished}^{\gamma'}(v)$ then $\mathtt{SepDist}^{\gamma'}(v) = 0 \leq \mathtt{SepDist}^\gamma(v)$. Otherwise, according to Lemma 39 $\mathtt{WontReset}^{\gamma'}(v)$. Furthermore, according to Lemma 36, $\mathtt{ph}_v^{\gamma'} \geq \mathtt{ph}_v^\gamma$, so $\mathtt{SepVal}(v) - \mathtt{ph}_v^{\gamma'} \leq \mathtt{SepVal}(v) - \mathtt{ph}_v^\gamma$. Consequently, we have $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^\gamma(v)$.
- If $\mathtt{WillReset}^\gamma(v)$, then $\mathtt{SepDist}^\gamma(v) = \mathtt{SepVal}(v) + (\mathtt{MaxPh}^\gamma(v) - \mathtt{ph}_v^\gamma)$.
  If v does not reset $\mathtt{ph}_v^{\gamma'} = 1$ during cs, then according to Lemma 40 $\mathtt{WillReset}^{\gamma'}(v)$, and according to Lemma 41 $\mathtt{MaxPh}^{\gamma'}(v) = \mathtt{MaxPh}^\gamma(v)$. Furthermore, by hypothesis $\mathtt{ph}_v^{\gamma'} \geq \mathtt{ph}_v^\gamma$, so $\mathtt{SepVal}(v) + (\mathtt{MaxPh}^{\gamma'}(v) - \mathtt{ph}_v^{\gamma'}) \leq \mathtt{SepVal}(v) + (\mathtt{MaxPh}^\gamma(v) - \mathtt{ph}_v^\gamma$. Thus, we deduce $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^\gamma(v)$.
  Otherwise, if v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs, and sets $\mathtt{ph}_v^{\gamma'} = \mathtt{ph}_v^\gamma + 1$, then according to Lemma 40 $\neg\mathtt{WillReset}^{\gamma'}(v)$, and thus we deduce $\mathtt{SepDist}^\gamma(v) \leq \mathtt{SepVal}(v)$. Since by definition $\mathtt{MaxPh}^\gamma(v) - \mathtt{ph}_v^\gamma \geq 0$, we have $\mathtt{SepDist}^\gamma(v) \geq \mathtt{SepVal}(v)$. By transitivity, $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^\gamma(v)$. ■

Lemma 44

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.
If v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs we have $\mathtt{SepDist}^{\gamma'}(v) < \mathtt{SepDist}^\gamma(v)$.

Proof: Let us consider three cases.

- If $\mathtt{Finished}^\gamma(v)$ then according to Lemma 31, v does not execute $\mathbb{R}_{\mathtt{NewPh}}$ during cs.
- If $\mathtt{WontReset}^\gamma(v)$ then if $\mathtt{Finished}^{\gamma'}(v)$, we have $\mathtt{SepDist}^{\gamma'}(v) = 0 < \mathtt{SepDist}^\gamma(v)$. Indeed, since $\mathtt{WontReset}^\gamma(v)$, we have $\mathtt{SepVal}(v) > \mathtt{ph}_v^\gamma$, and so $\mathtt{SepDist}^\gamma(v) > 0$.
  Let us now suppose that $\neg\mathtt{Finished}^{\gamma'}(v)$. According to Lemma 39 $\mathtt{WontReset}^{\gamma'}(v)$. Furthermore, according to Lemma 36, $\mathtt{ph}_v^{\gamma'} > \mathtt{ph}_v^\gamma$, and thus $\mathtt{SepVal}(v) - \mathtt{ph}_v^{\gamma'} < \mathtt{SepVal}(v) - \mathtt{ph}_v^\gamma$. Consequently, $\mathtt{SepDist}^{\gamma'}(v) < \mathtt{SepDist}^\gamma(v)$.
- If $\mathtt{WillReset}^\gamma(v)$ then let us consider two options.
  If $\mathtt{ph}_v^{\gamma'} = \mathtt{ph}_v^\gamma + 1$ then according to Lemma 40 $\mathtt{WillReset}^{\gamma'}(v)$, and according to Lemma 41 $\mathtt{MaxPh}^{\gamma'}(v) = \mathtt{MaxPh}^\gamma(v)$. Thus, we deduce $\mathtt{SepVal}(v) + (\mathtt{MaxPh}^{\gamma'}(v) - \mathtt{ph}_v^{\gamma'}) < \mathtt{SepVal}(v) + (\mathtt{MaxPh}^\gamma(v) - \mathtt{ph}_v^\gamma$, which means $\mathtt{SepDist}^{\gamma'}(v) < \mathtt{SepDist}^\gamma(v)$.
  If $\mathtt{ph}_v^{\gamma'} = 1$ then according to Lemma 40 $\neg\mathtt{WillReset}^{\gamma'}(v)$. By definition of $\mathtt{MaxPh}^\gamma(v)$, we have $\mathtt{MaxPh}^\gamma(v) \geq \mathtt{ph}_v$, so $\mathtt{SepDist}^\gamma(v) \geq \mathtt{SepVal}(v)$. On the other hand, whether $\mathtt{Finished}^{\gamma'}(v)$ or $\mathtt{WontReset}^{\gamma'}(v)$, we have $\mathtt{SepDist}^\gamma(v) < \mathtt{SepVal}(v)$. As a consequence, we obtain $\mathtt{SepDist}^{\gamma'}(v) < \mathtt{SepDist}^\gamma(v)$. ■

Lemma 45

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

Then $\mathtt{ParSteps}^{\gamma'}(v) \leq \mathtt{ParSteps}^{\gamma}(v)$, and if v executes $\mathbb{R}_{\mathtt{maxPos}}$ or $\mathbb{R}_{\mathtt{NewPh}}$ during cs, then $\mathtt{ParSteps}^{\gamma'}(v) < \mathtt{ParSteps}^{\gamma}(v)$.

Proof: Remark first that if v is not activated during cs, then according to Lemma 43 $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^{\gamma}(v)$. On the other hand, $\mathtt{b}_v^{\gamma'} = \mathtt{b}_v^{\gamma}$, so $\mathtt{ParSteps}^{\gamma'}(v) \leq \mathtt{ParSteps}^{\gamma}(v)$.

If v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs then according to Lemma 44,

$$\begin{array}{rrl} & \mathtt{SepDist}^{\gamma'}(v) + 1 & \leq \mathtt{SepDist}^{\gamma}(v) \\ \Rightarrow & 2 \times \mathtt{SepDist}^{\gamma'}(v) + 2 & \leq 2 \times \mathtt{SepDist}^{\gamma}(v) \\ \Rightarrow & 2 \times \mathtt{SepDist}^{\gamma'}(v) + 1 & < 2 \times \mathtt{SepDist}^{\gamma}(v). \end{array}$$

Furthermore, $\mathtt{ParSteps}^{\gamma'}(v) \leq 2 \times \mathtt{SepDist}^{\gamma'}(v) + 1$ and $\mathtt{ParSteps}^{\gamma}(v) \geq 2 \times \mathtt{SepDist}^{\gamma}(v)$. By transitivity, we deduce $\mathtt{ParSteps}^{\gamma'}(v) < \mathtt{ParSteps}^{\gamma}(v)$.

Finally, if v executes $\mathbb{R}_{\mathtt{maxPos}}$ during cs, then according to Lemma 43 $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^{\gamma}(v)$. Furthermore, we have $\mathtt{b}_v^{\gamma} = \bot$ and $\mathtt{b}_v^{\gamma'} \neq \bot$, so $\mathtt{ParSteps}^{\gamma'}(v) = 2 \times \mathtt{SepDist}^{\gamma'}(v) \leq 2 \times \mathtt{SepDist}^{\gamma}(v) < 2 \times \mathtt{SepDist}^{\gamma}(v) + 1 = \mathtt{ParSteps}^{\gamma}(v)$, which reduces to $\mathtt{ParSteps}^{\gamma'}(v) < \mathtt{ParSteps}^{\gamma}(v)$. ■

Lemma 46

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

Let us consider $u \in \mathtt{Players}(v)$. Then $\mathtt{ChSteps}^{\gamma'}(v) \leq \mathtt{ChSteps}^{\gamma}(v)$, and if u executes $\mathbb{R}_{\mathtt{NewBit}}$ during cs, then $\mathtt{ChSteps}^{\gamma'}(v) < \mathtt{ChSteps}^{\gamma}(v)$.

Proof: Suppose first that u does not execute $\mathbb{R}_{\mathtt{NewBit}}$.

If v is not activated, then $\mathtt{ChSteps}^{\gamma'}(v) = \mathtt{ChSteps}^{\gamma}(v)$.

If v executes $\mathbb{R}_{\mathtt{NewPh}}$ during cs then according to Lemma 44, $\mathtt{SepDist}^{\gamma'}(v) < \mathtt{SepDist}^{\gamma}(v)$, and thus $\mathtt{ChSteps}^{\gamma'}(u, v) \leq \mathtt{ChSteps}^{\gamma}(u, v)$.

If v executes $\mathbb{R}_{\mathtt{maxPos}}$ during cs then no children of v is activated during cs according to Lemma 20, and thus, $\mathtt{Synch}^{\gamma}(v)$. Consequently, $\mathtt{ph}_u^{\gamma'} = \mathtt{ph}_u^{\gamma} = \mathtt{ph}_v^{\gamma} = \mathtt{ph}_v^{\gamma'}$. Furthermore, since $\mathtt{b}_v^{\gamma'} \neq \bot$, we obtain $\neg\mathtt{AnswerPar}^{\gamma'}(u, v)$. According to Lemma 43, $\mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^{\gamma}(v)$ so we deduce $\mathtt{ChSteps}^{\gamma'}(u, v) \leq \mathtt{ChSteps}^{\gamma}(u, v)$.

Now suppose that u executes $\mathbb{R}_{\mathtt{NewBit}}$ during cs. According to Lemma 20, v is not activated during cs. Furthermore, we have, by definition, $\mathtt{AnswerPar}^{\gamma}(u, v)$. The execution of $\mathtt{Announce}(u)$ sets $\mathtt{ph}_u^{\gamma'} = \mathtt{ph}_u^{\gamma} = \mathtt{ph}_u^{\gamma'}$ and $\mathtt{b}_u^{\gamma'} = \mathtt{Bit}_u(\mathtt{ph}_v^{\gamma})$. If $\mathtt{ph}_v^{\gamma} = 1$ then $\mathtt{Bit}_u(\mathtt{ph}_v^{\gamma}) \neq \bot$, so $\neg\mathtt{AnswerPar}^{\gamma'}(v)$, and if $\mathtt{ph}_v^{\gamma} \neq 1$ then $\mathtt{ph}_v^{\gamma'} \neq 1$ and thus $\neg\mathtt{AnswerPar}^{\gamma'}(v)$. Thus, we conclude $\mathtt{ChSteps}^{\gamma'}(u, v) = \mathtt{SepDist}^{\gamma'}(v) \leq \mathtt{SepDist}^{\gamma}(v) < \mathtt{SepDist}^{\gamma}(v) + 1 = \mathtt{ChSteps}^{\gamma}(u, v)$. By transitivity, $\mathtt{ChSteps}^{\gamma'}(u, v) < \mathtt{ChSteps}^{\gamma}(u, v)$. ■

Lemma 47

Let $cs = \gamma \to \gamma'$ be a computing step and let v be a node such that $\mathtt{tok}_v^\gamma = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, and suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(v, \gamma)$.

Then $\mathtt{W_{Nego}}(v, \gamma') \leq \mathtt{W_{Nego}}(v, \gamma)$, and if v executes $\mathbb{R}_{\mathtt{NewPh}}$ or $\mathbb{R}_{\mathtt{maxPos}}$ during cs, or if $\exists u \in \mathtt{Players}(v)$ such that u executes $\mathbb{R}_{\mathtt{NewBit}}$ during cs, then $\mathtt{W_{Nego}}(v, \gamma') < \mathtt{W_{Nego}}(v, \gamma)$.

Proof: This is a direct consequence of Lemmas 45 and 46. ■

Theorem 11 establishes that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})$ is pre-admissible.

Theorem 11

Let $cs = \gamma \to \gamma'$ be a computing step and let a be an anchor at γ' such that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs.

$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma'}, \gamma') \preceq^5 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma}, \gamma)$.

Proof: According to Theorem 9,

$$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \preceq^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma}, \gamma).$$

If $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \prec^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma}, \gamma)$, then the result is trivial. Let us now suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma}, \gamma)$, and prove that $\mathtt{W_{Nego}}(D_a^{\gamma'}, \gamma') \preceq \mathtt{W_{Nego}}(D_a^{\gamma}, \gamma)$.

In the same way we did in the proof of Theorem 5, we can suppose that for any branch $\mathcal{B}$ of G_a, $D_a^{\gamma'}(\mathcal{B})$ is not longer than $D_a^{\gamma}(\mathcal{B})$. In such circumstances, we have $\mathtt{W_{Nego}}(D_a^{\gamma'}, \gamma') \succ \mathtt{W_{Nego}}(D_a^{\gamma}, \gamma)$ only if there exists a branch $\mathcal{B}$ of G_a such that $\exists v \in D_a^{\gamma'}(\mathcal{B})$ such that $\mathtt{W_{Nego}}(v, \gamma') > \mathtt{W_{Nego}}(v, \gamma)$.

Let us consider one such node $v \in D_a^{\gamma'}(\mathcal{B})$. According to Theorems 6 and 8, we have $\mathtt{tok}_v^{\gamma'} = \mathtt{tok}_v^{\gamma}$. If $\mathtt{tok}_v^{\gamma} \neq \bullet$, then $\mathtt{W_{Nego}}(v, \gamma') = 0 = \mathtt{W_{Nego}}(v, \gamma)$. If $\mathtt{tok}_v^{\gamma} = \bullet$ and $\mathtt{tok}_v^{\gamma'} = \bullet$, then we fall under the hypothesis of Lemma 47 and thus $\mathtt{W_{Nego}}(v, \gamma') \leq \mathtt{W_{Nego}}(v, \gamma)$.

Thus, for any branch $\mathcal{B}$ of G_a, for any node $v \in D_a^{\gamma'}(\mathcal{B})$, $\mathtt{W_{Nego}}(v, \gamma') \leq \mathtt{W_{Nego}}(v, \gamma)$, and thus $\mathtt{W_{Nego}}(D_a^{\gamma'}, \gamma') \preceq \mathtt{W_{Nego}}(D_a^{\gamma}, \gamma)$. ■

Theorem 12 establishes that under certain circumstances, we are certain that the weight of one $\mathtt{CD}^o$ decreases. It will we useful to prove the admissibility of $\mathtt{W}$.

Theorem 12

Let $cs = \gamma \to \gamma'$ be a computing step and let a be an anchor at γ' such that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs.

If $\exists v \in D_a^{\gamma}$ such that $\mathtt{tok}_v^{\gamma} = \bullet$ and either v executes $\mathbb{R}_{\mathtt{NewPh}}$ or $\mathbb{R}_{\mathtt{maxPos}}$ during cs, either $\exists u \in \mathtt{Players}^{\gamma}(v)$ such that u executes $\mathbb{R}_{\mathtt{NewBit}}$ during cs, then

$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma'}, \gamma') \prec^5 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma}, \gamma)$

Proof: According to Theorem 11,

$$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma'}, \gamma') \preceq^5 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma}, \gamma).$$

If $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') \prec^4 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma}, \gamma)$, then the result is trivial. Let us now suppose that $(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma'}, \gamma') = (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}})(D_a^{\gamma}, \gamma)$, and prove that $\mathtt{W_{Nego}}(D_a^{\gamma'}, \gamma') \prec \mathtt{W_{Nego}}(D_a^{\gamma}, \gamma)$.

Let us consider $v \in D_a^{\gamma}$ such that $\mathtt{tok}_v^{\gamma} = \bullet$ and either v executes $\mathbb{R}_{\mathtt{NewPh}}$ or $\mathbb{R}_{\mathtt{maxPos}}$ during cs, either $\exists u \in \mathtt{Players}^{\gamma}(v)$ such that u executes $\mathbb{R}_{\mathtt{NewBit}}$ during cs. According to Theorems 6 and 8, we have $\mathtt{tok}_v^{\gamma'} = \bullet$. Thus, we fall under the hypothesis of Lemma 47 and we have $\mathtt{W_{Nego}}(v, \gamma') \prec \mathtt{W_{Nego}}(v, \gamma)$. Thus, $\mathtt{W_{Nego}}(D_a^{\gamma'}, \gamma') \neq \mathtt{W_{Nego}}(D_a^{\gamma}, \gamma)$ so according to Theorem 11, we conclude that

$$(\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma'}, \gamma') \prec^5 (\mathtt{W_{Ch}}, \mathtt{W_{Er}}, \mathtt{W_{Circ}}, \mathtt{W_{Play}}, \mathtt{W_{Nego}})(D_a^{\gamma}, \gamma).$$

■

C.2.9 Conclusions

In this section we combine the results of Sections C.2.4 to C.2.8 to provide a suitable definition of *admissible potential function*. More precisely, we provide a definition of admissibility which is wide enough to include our weight function $\mathtt{W}$, and tight enough to be convenient in the following proofs.

The final theorem of this section states that our algorithm satisfies Condition 2 of Specification 1, but other intermediate theorems will be used in the following section.

From now on, to ease the reading, when we use comparison operations on 5-tuple induced by $\mathtt{W}$, we simply use the symbol $\prec$ rather than $\prec^5$ to denote the lexicographic order on 5-tuples of $\mathtt{WD}^o$'s.

Theorem 13 states that $\mathtt{W}$ is pre-admissible.

Theorem 13

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots \to \gamma_m$ be an execution. If a is an anchor at γ_m, and if a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during ϵ, then $\mathtt{W}(D_a^{\gamma_m}) \preceq \mathtt{W}(D_a^{\gamma_0})$.

Proof: By definition, $\forall t \in [0, m-1], a$ does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during $\gamma_t \to \gamma_{t+1}$, so according to Theorem 11, $\forall t \in [0, m-1], \mathtt{W}(D_a^{\gamma_{t+1}}) \preceq \mathtt{W}(D_a^{\gamma_t})$. By transitivity, we conclude $\mathtt{W}(D_a^{\gamma_m}) \preceq \mathtt{W}(D_a^{\gamma_0})$. ■

As hinted above, to define admissibility we must first interest to what an activation of a $\mathtt{CD}^o$ is.

Definition 25 (*Activation of a* $\mathtt{CD}^o$)

Let $cs = \gamma \to \gamma'$ be a computing step, let a be an anchor at γ', and let $u \in V$ be a node. We say that u activates D_a^γ during cs if:

- *$u = a$ and u executes $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs, or*
- *$u \in \mathbb{B}(D_a^\gamma)$ and u executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$ during cs, or*
- *$u \in D_a^\gamma$ and u executes $\mathbb{E}_{\mathtt{TrustChild}}$ or $\mathbb{R}_{\mathtt{Give}}$ or $\mathbb{R}_{\mathtt{NewPlay}}$ or $\mathbb{R}_{\mathtt{Drop}}$ or $\mathbb{R}_{\mathtt{Nego}}$ or $\mathbb{R}_{\mathtt{OfferUp}}$ or $\mathbb{R}_{\mathtt{Receive}}$ or $\mathbb{R}_{\mathtt{Return}}$ or $\mathbb{R}_{\mathtt{ReNego}}$ or $\mathbb{R}_{\mathtt{NewPh}}$ or $\mathbb{R}_{\mathtt{maxPos}}$ during cs, or*
- *$\exists v \in D_a^\gamma, v \in \mathcal{P}(u), \mathtt{tok}_v^\gamma = \bullet$ and u executes $\mathbb{R}_{\mathtt{Lose}}$ or $\mathbb{R}_{\mathtt{NewBit}}$ during cs, or*
- *$\exists v \in D_a^\gamma, v \in \mathcal{P}_{\mathtt{neq}}(u), \mathtt{tok}_v^\gamma \in \{\circlearrowleft, \circ\}$ and u executes $\mathbb{R}_{\mathtt{ReplayD}}$ during cs, or*
- *$\exists v \in D_a^\gamma, v \in \mathcal{P}_{\mathtt{eq}}(u), \mathtt{tok}_v^\gamma = \uparrow$ and u executes $\mathbb{R}_{\mathtt{Fake}}$ or $\mathbb{R}_{\mathtt{ReplayUp}}$ during cs.*

We say that D_a^γ is activated during cs if there exists one node $u \in V$ which activates D_a^γ during cs.

We now prove that the definition of activation matches the evolution of $\mathtt{CD}^o$'s. Namely, if one $\mathtt{CD}^o$ is not activated during one computing step, either it is unchanged, either it disappears, its anchor becoming an intern node of one other $\mathtt{CD}^o$.

Lemma 48

Let $cs = \gamma \to \gamma'$ be a computing step, and let $a \in \mathbb{A}(\gamma)$ be an anchor at γ. If D_a^γ is not activated during cs then either $D_a^{\gamma'} = D_a^\gamma$, either $a \notin \mathbb{A}(\gamma')$.

Proof: Let us suppose that D_a^γ is not activated during cs and that $a \in \mathbb{A}(\gamma')$, and let us prove $D_a^{\gamma'} = D_a^\gamma$. To achieve this, let us prove that for all branch $\mathcal{B} = v_1 \cdots v_k$ of G_a, $D_a^{\gamma'}(\mathcal{B}) = D_a^\gamma(\mathcal{B})$.

Remark first that $\forall i \in [1,k]$, $\mathtt{tok}_{v_i}^{\gamma'} = \mathtt{tok}_{v_i}^{\gamma}$, which implies $D_a^{\gamma}(\mathcal{B}) \subset D_a^{\gamma'}(\mathcal{B})$. We have already seen in the proof of Lemma 18 that if $D_a^{\gamma'}(\mathcal{B})$ is longer than $D_a^{\gamma}(\mathcal{B})$ then there exists $v \in \mathbb{B}(D_a^{\gamma})$ which executes $\mathbb{R}_{\mathtt{Win}}$ during cs, which would activate D_a^{γ}. Consequently, $D_a^{\gamma'}(\mathcal{B}) = D_a^{\gamma}(\mathcal{B})$.

This, being true for all branch $\mathcal{B}$ of G_a, proves $D_a^{\gamma'} = D_a^{\gamma}$. ■

We now prove that the definition of activation captures almost all computing steps of our algorithm. The only situation in which no $\mathtt{CD}^o$ is activated is when all activated nodes execute $\mathbb{R}_{\mathtt{ReWin}}$, rule which is not considered by $\mathtt{W}$, but which cannot be responsible of livelock only by itself.

Lemma 49

Let $cs = \gamma \to \gamma'$ be a computing step. There exists an anchor $a \in \mathbb{A}(\gamma)$ such that D_a^{γ} is activated during cs, or all nodes activated during cs execute $\mathbb{R}_{\mathtt{ReWin}}$.

Proof: Let us suppose that there exists at least one node $u \in V$ such that u executes one rule different from $\mathbb{R}_{\mathtt{ReWin}}$ during cs.

- If u executes $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs then u is the root, and u activates D_r^{γ}.
- If u executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$ during cs then $\exists u \in \mathcal{P}(v) : \mathtt{tok}_p^{\gamma} \notin \{\bot, \uparrow, \downarrow\}$, so according to Lemma 8, $\exists a \in \mathbb{A}(\gamma)$ such that $v \in D_a^{\gamma}$ and thus $u \in \mathbb{B}(D_a^{\gamma})$, and u activates D_a^{γ}.
- If u executes $\mathbb{E}_{\mathtt{TrustChild}}$ or $\mathbb{R}_{\mathtt{Give}}$ or $\mathbb{R}_{\mathtt{NewPlay}}$ or $\mathbb{R}_{\mathtt{Drop}}$ or $\mathbb{R}_{\mathtt{Nego}}$ or $\mathbb{R}_{\mathtt{OfferUp}}$ or $\mathbb{R}_{\mathtt{Receive}}$ or $\mathbb{R}_{\mathtt{Return}}$ or $\mathbb{R}_{\mathtt{ReNego}}$ or $\mathbb{R}_{\mathtt{NewPh}}$ or $\mathbb{R}_{\mathtt{maxPos}}$ during cs then $\mathtt{tok}_u^{\gamma} \neq \bot$ so according to Lemma 8, there exists $a \in \mathbb{A}(\gamma)$ such that $u \in D_a^{\gamma}$ and thus u activates D_a^{γ}.
- If u executes $\mathbb{R}_{\mathtt{Lose}}$ or $\mathbb{R}_{\mathtt{NewBit}}$ during cs then $\mathtt{LoseNeg}^{\gamma}(u) \vee \mathtt{AnswerNeg}^{\gamma}(u)$. In both cases, $\exists p \in \mathcal{P}(u) : \mathtt{tok}_p^{\gamma} = \bullet$, so according to Lemma 8, there exists $a \in \mathbb{A}(\gamma)$ such that $v \in D_a^{\gamma}$ and thus u activates D_a^{γ}.
- If u executes $\mathbb{R}_{\mathtt{ReplayD}}$ during cs then $\mathtt{ReplayL}^{\gamma}(u)$ and thus $\exists p \in \mathcal{P}_{\mathtt{neq}}(u) : \mathtt{tok}_p^{\gamma} \in \{\circlearrowleft, \circ\}$. According to Lemma 8, there exists $a \in \mathbb{A}(\gamma)$ such that $v \in D_a^{\gamma}$ and thus u activates D_a^{γ}.
- If u executes $\mathbb{R}_{\mathtt{Fake}}$ or $\mathbb{R}_{\mathtt{ReplayUp}}$ during cs then $\exists v \in \mathcal{P}_{\mathtt{neq}}{}^{\gamma}(u) : \mathtt{tok}_v^{\gamma} = \uparrow$. According to Lemma 8, there exists $a \in \mathbb{A}(\gamma)$ such that $v \in D_a^{\gamma}$ and thus u activates D_a^{γ}. ■

We can finally provide a definition of admissibility, which corresponds to our weight function $\mathtt{W}$ according to Theorem 14.

Definition 26 (*Admissible weight function*)

Let $\mathtt{W}_1, \ldots, \mathtt{W}_k : V \times \Gamma \to M$ be k weight functions on nodes such that $(\mathtt{W}_1, \ldots, \mathtt{W}_k)$ is pre-admissible. We say that $(\mathtt{W}_1, \ldots, \mathtt{W}_k)$ is admissible if for all computing step $cs = \gamma \to \gamma'$, for all $\mathtt{CD}^o$ $D_a^{\gamma'}$, if D_a^{γ} is activated during cs and if a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs, then

$$(\mathtt{W}_1, \ldots, \mathtt{W}_k)(D_a^{\gamma'}, \gamma') \prec^k (\mathtt{W}_1, \ldots, \mathtt{W}_k)(D_a^{\gamma}, \gamma)$$

Theorem 14

$\mathtt{W}$ is admissible.

Proof: We established in Theorem 13 that $\mathtt{W}$ is pre-admissible. Let us now consider one computing step $\gamma \to \gamma'$, one $\mathtt{CD}^o$ $D_a^{\gamma'}$ which is activated during cs, and suppose that a does not execute $\mathbb{R}^r_{\mathtt{NewDFS}}$ during cs.

Let us consider one node u which activates D_a^{γ} during cs.

- If $u \in \mathbb{B}(D_a^{\gamma})$ and u executes $\mathbb{R}_{\mathtt{Win}}$ or $\mathbb{R}_{\mathtt{FakeWin}}$ during cs, then according to Theorem 4, $\mathtt{W}(D_a^{\gamma'}, \gamma') \prec \mathtt{W}(D_a^{\gamma}, \gamma)$.

- If $u \in D_a^\gamma$ and u executes $\mathbb{E}_{\texttt{TrustChild}}$ or $\mathbb{R}_{\texttt{Give}}$ or $\mathbb{R}_{\texttt{NewPlay}}$ or $\mathbb{R}_{\texttt{Drop}}$ or $\mathbb{R}_{\texttt{Nego}}$ or $\mathbb{R}_{\texttt{OfferUp}}$ or $\mathbb{R}_{\texttt{Receive}}$ or $\mathbb{R}_{\texttt{Return}}$ or $\mathbb{R}_{\texttt{ReNego}}$ or $\mathbb{R}_{\texttt{NewPh}}$ or $\mathbb{R}_{\texttt{maxPos}}$ during cs, then according to Theorems 6, 8, and 12, $\texttt{W}(D_a^{\gamma'}, \gamma') \prec \texttt{W}(D_a^\gamma, \gamma)$.
- $\exists v \in D_a^\gamma, v \in \mathcal{P}(u), \texttt{tok}_v^\gamma = \bullet$ and u executes $\mathbb{R}_{\texttt{Lose}}$ or $\mathbb{R}_{\texttt{NewBit}}$ during cs, then since $\texttt{OkNeg}^\gamma(u)$, we deduce $v \in \mathcal{P}_{\texttt{neq}}{}^\gamma(v)$, and thus according to Theorems 10 and 12 $\texttt{W}(D_a^{\gamma'}, \gamma') \prec \texttt{W}(D_a^\gamma, \gamma)$.
- $\exists v \in D_a^\gamma, v \in \mathcal{P}_{\texttt{neq}}(u), \texttt{tok}_v^\gamma \in \{\circlearrowleft, \circ\}$ and u executes $\mathbb{R}_{\texttt{ReplayD}}$ during cs, then according to Theorem 12 $\texttt{W}(D_a^{\gamma'}, \gamma') \prec \texttt{W}(D_a^\gamma, \gamma)$.
- $\exists v \in D_a^\gamma, v \in \mathcal{P}_{\texttt{eq}}(u), \texttt{tok}_v^\gamma = \uparrow$ and u executes $\mathbb{R}_{\texttt{Fake}}$ or $\mathbb{R}_{\texttt{ReplayUp}}$ during cs, then according to Theorem 12 $\texttt{W}(D_a^{\gamma'}, \gamma') \prec \texttt{W}(D_a^\gamma, \gamma)$.

■

We now prove that if the anchor of a $\texttt{CD}^o$ does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$, then the $\texttt{CD}^o$ can only be activated a finite number of time.

Theorem 15

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be an infinite execution, and let $a \in V$ such that $\forall t \geq 0, a \in \mathbb{A}(\gamma_t)$, and such that a does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during ϵ.

There only exists a finite number of computing steps $cs_t = \gamma_t \to \gamma_{t+1}$ such that $D_a^{\gamma_t}$ is activated during cs_t.

Proof: According to Theorem 14, W is admissible, so if $D_a^{\gamma_t}$ is activated during cs_t, we have $\texttt{W}(D_a^{\gamma_{t+1}}, \gamma_{t+1}) \prec \texttt{W}(D_a^{\gamma_t}, \gamma_t)$.

Let us now reason by contradiction and suppose that there exists an infinity of such computing steps, $cs_{t_0}, cs_{t_1}, \ldots$. We have $\forall i \geq 0, \texttt{W}(D_a^{\gamma_{t_i+1}}, \gamma_{t_i+1}) \prec \texttt{W}(D_a^{\gamma_{t_i}}, \gamma_{t_i})$. Furthermore, according to Theorem 13, we also have $\forall i \geq 0, \texttt{W}(D_a^{\gamma_{t_{i+1}}}, \gamma_{t_{i+1}}) \preceq \texttt{W}(D_a^{\gamma_{t_i+1}}, \gamma_{t_i+1})$. Thus, we deduce $\forall i \geq 0, \texttt{W}(D_a^{\gamma_{t_{i+1}}}, \gamma_{t_{i+1}}) \prec \texttt{W}(D_a^{\gamma_{t_i}}, \gamma_{t_i})$.

Since $\prec$ is a well-founded order on $\texttt{WD}^o$'s, such infinite decreasing sequence does not exist, and thus we raised a contradiction. Consequently, there only exists a finite number of computing steps $cs_t = \gamma_t \to \gamma_{t+1}$ such that $D_a^{\gamma_t}$ is activated during cs_t. ■

The following theorem is crucial, since it states that at some point, the $\texttt{CD}^o$ which are not anchored at the root cease being activated, and cease disappearing too. It is a key theorem to prove the last result of this section, but will also be determining in the following section to prove the fairness of our algorithm.

Theorem 16

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be an infinite execution. There exists $t_0 \geq 0$ such that for all $t \geq t_0$, $\mathbb{A}^*(\gamma_t) = \mathbb{A}^*(\gamma_{t_0})$ and $\forall a \in \mathbb{A}^*(\gamma_t), D_a^{\gamma_t}$ is not activated during $\gamma_t \to \gamma_{t+1}$.

Proof: Let us first define t_0, and let us consider $a \in \mathbb{A}^*(\gamma_0)$.

If $\exists t \geq 0 : a \notin \mathbb{A}^*(\gamma_t)$ then let us set $t_a = t$. According to Lemma 9, $\forall t \geq t_a, a \notin \mathbb{A}(\gamma_t)$.

Otherwise, we have $\forall t \geq 0, a \in \mathbb{A}^*(\gamma_t)$. According to Theorem 15, there exists only a finite number of computing steps $cs_t = \gamma_t \to \gamma_{t+1}$ such that $D_a^{\gamma_t}$ is activated during cs_t (since $a \neq r$, it cannot execute $\mathbb{R}^r_{\texttt{NewDFS}}$). Let us consider t' the highest value of t such that $D_a^{\gamma_t}$ is activated during cs_t, and let us set $t_a = t' + 1$.

Let us now set $t_0 = \max_{a \in \mathbb{A}^*(\gamma_0)} t_a$. Let us consider one anchor $a \in \mathbb{A}^*(\gamma_{t_0})$. We have $\forall t \geq t_0, a \in \mathbb{A}^*(\gamma_t)$, since if not, we would have $t_a > t_0$ which is contradictory with the definition of t_0. Therefore we have $\forall t \geq t_0, \mathbb{A}^*(\gamma_{t_0}) \subseteq \mathbb{A}^*(\gamma_t)$, and according to Lemma 9 we conclude $\forall t \geq t_0, \mathbb{A}^*(\gamma_{t_0}) = \mathbb{A}^*(\gamma_t)$.

Furthermore, since $\forall t \geq t_0, a \in \mathbb{A}^*(\gamma_t)$, t_a has been defined as one more than the highest value such that $D_a^{\gamma_t}$ is activated during cs_t. Therefore, $\forall t \geq t_0, D_a^{\gamma_t}$ is not activated during cs_t.

■

We now formally prove that our algorithm satisfies Condition 2 of Specification 1.

Theorem 17

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be a maximal execution, and let r be the root of G.
ϵ is infinite, and can be divided into an infinite number of circulation rounds.

Proof: According to Theorem 1, we only have to prove that there exists an infinite number of computing steps $cs_i = \gamma_i \to \gamma_{i+1}$ such that r executes $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs_i.

According to Theorem 16, there exists $t_0 \geq 0$ such that $\forall a \in \mathbb{A}(\gamma_t) \setminus \{r\}, \forall t' \geq t_0, a \in \mathbb{A}(\gamma_{t'})$ and $D_a^{\gamma_{t'}}$ is not activated during $\gamma_{t'} \to \gamma_{t'+1}$.

Let us reason by contradiction and suppose that there only exists a finite number of computing steps $cs_i = \gamma_i \to \gamma_{i+1}$ such that r executes $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs_i. Then, there exists $t_1 \geq 0$ such that $\forall t \geq t_1, r$ does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs_t. Thus, according to Theorem 15, there exists $t_2 \geq t_1$ such that $\forall t \geq t_2, D_r^{\gamma_t}$ is not activated during cs_t.

Let us consider $t_3 = \max(t_0, t_2)$, and the infinite execution $\epsilon_3 = \gamma_{t_3} \to \gamma_{t_3+1} \to \cdots$. By definition, no $\texttt{CD}^o$ is activated during ϵ_3. Thus, during ϵ_3, nodes only execute $\mathbb{R}_{\texttt{ReWin}}$. But once a node v executes $\mathbb{R}_{\texttt{ReWin}}$, it cannot execute it again before the execution of one other rule sets $\texttt{play}_v = \texttt{F}$. Thus, there does not exist such execution, so we reached a contradiction.

Consequently, there exists an infinite number of computing steps $cs_i = \gamma_i \to \gamma_{i+1}$ such that r executes $\mathbb{R}^r_{\texttt{NewDFS}}$ during cs_i. ■

C.3 Convergence

In this section, we establish that our algorithm behaves correctly, by largely using results established in Section C.2, and especially in Section C.2.9. We use in particular Theorems 16 and 17, that allow us to start the reasoning at a moment where the algorithm has already partly converged. Namely, we consider executions in which the set of all the anchors remain constant, and such that all the $\texttt{CD}^o$'s which are not anchored at the root of the $\texttt{D}^o$ are not activated.

Once this is considered, the proof is organized according to the following scheme. We first establish some additional stability properties on the $\texttt{CD}^o$ anchored at the root of the $\texttt{D}^o$. Then, we give a formal definition of the notion of having the token, for one node. The formal notion we use to describe the part of an execution where one node is involved in the token circulation is *circulation round*, introduced in Theorem 1 and Definition 28. After that, we prove that if one node executes a circulation round, then all of its children which might take the token (depending on their variables `play` and `c`) actually do it too, and execute a circulation round as well. We can thus reason by induction an prove that, starting at the root, the token browses a part of the $\texttt{D}^o$. Finally we prove that the alternation of the color of the root, which occurs each time it executes $\mathbb{R}^r_{\texttt{NewDFS}}$, associated to properties of the algorithm, especially its effect on the variables `play` of the children of a node, guarantee that the part of the $\texttt{D}^o$ which is being browsed by the token grows each time the root executes $\mathbb{R}^r_{\texttt{NewDFS}}$, and eventually become the entire $\texttt{D}^o$.

As we said before, we consider executions which start after the value of t_0 guaranteed by Theorem 16. In particular, the set of anchors does not evolve during the executions we consider, and according to Lemma 48, no $\texttt{CD}^o$ other than the one anchored at r will be updated. Consequently, to enhance readability, we allow ourselves to simply write $\mathbb{A}$ (resp. $\mathbb{A}^*$) without referring to the configuration in which this set is evaluated, and to simply write D_a when $a \in \mathbb{A}^*$ without referring to the configuration in which we consider the $\texttt{CD}^o$ anchored at a.

Lemma 50

Let ϵ be a maximal execution. We have $\forall t \geq 0, \forall a \in \mathbb{A}^*, D_r^{\gamma_t} \cap D_a = \emptyset$.

Proof: Let us reason by contradiction and suppose that $\exists v \in D_r^{\gamma_t} \cap D_a$. Remark first that since $v \in D_a$ and $a \neq r$, we have $v \neq r$.

Since $v \in D_r^{\gamma_t}$, there exists $\mathcal{B} = v_1 \cdots v_k$ a branch of $D_r^{\gamma_t}$ such that $v_1 = r$ and $v_k = v$. According to Theorem 17, there exists $t' \geq t$ such that r executes $\mathbb{R}^r_{\texttt{NewDFS}}$ during $\gamma_{t'} \to \gamma_{t'+1}$. By definition, at $\gamma_{t'}$, only r is a branch of $D_r^{\gamma_{t'}}$, and in particular, $\mathcal{B}$ is not. Let us consider t_0 the smallest value greater than t such that $\mathcal{B}$ is a branch of $D_r^{\gamma_{t_0}}$ and $\mathcal{B}$ is not a branch of $D_r^{\gamma_{t_0+1}}$. According to Lemma 10, we deduce that v executes $\mathbb{R}_{\texttt{Return}}$ during $\gamma_{t_0} \to \gamma_{t_0+1}$.

By definition, we deduce that v activates D_a during $\gamma_{t_0} \to \gamma_{t_0+1}$, which is contradictory. ■

Definition 27 (*branch*-$\texttt{CD}^o$)

Let D_a^γ be a $\texttt{CD}^o$. We say that D_a^γ is a branch-$\texttt{CD}^o$ if the vertices of D_a^γ are $\{v_1, v_2, \ldots, v_k\}$ with $v_1 = a$ and $\forall i \in [2,k], v_i \in \mathcal{C}_{D_a^\gamma}(v_{i-1})$ and $\forall i \in [2,k], \texttt{c}_{v_i} = \texttt{c}_a$.

We say that a branch-$\texttt{CD}^o$ is clean if $\forall i \in [1,k], \neg\texttt{Er}^\gamma(v_i)$, and if $\texttt{tok}^\gamma_{v_i} = \downdownarrows$, then $|\texttt{Players}^\gamma(v_i)| + |\{u \in \mathcal{C}^\gamma(v_i) : \texttt{tok}^\gamma_u = \bigstar\}| \leq 1$.

Lemma 51

Let D_a^γ be a clean branch-$\texttt{CD}^o$, and let us label its vertices $v_1, v_2 \ldots v_k$ such that $\forall i \in [2,k], v_i \in \mathcal{C}_{D_a^\gamma}(v_{i-1})$.

$\forall i \in [1, k-2], \texttt{tok}_{v_i} = \downarrow$, and if $\texttt{tok}_{v_{k-1}} \neq \downarrow$ then $(\texttt{tok}_{v_{k-1}}, \texttt{tok}_{v_k}) \in \{(\downdownarrows, \bigstar), (\circlearrowleft, \bigstar), (\circ, \uparrow)\}$.

Proof: This is a direct consequence of the definition of predicate $\texttt{Er}$, since nodes are not in error and form a branch in G. ■

Lemma 52

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be an execution such that r execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during $\gamma_0 \to \gamma_1$. Then $\forall t$, $D_r^{\gamma_t}$ is a clean branch-$\texttt{CD}^o$.

Proof: We reason by induction. The base case is true since r executes $\mathbb{R}^r_{\texttt{NewDFS}}$ in $\gamma_0 \to \gamma_1$, thus $\texttt{tok}_r^{\gamma_0} = \uparrow$ and thus $D_r^{\gamma_0} = \{r\}$, which is a clean branch-$\texttt{CD}^o$. Let us also prove that $D_r^{\gamma_1}$ is a clean branch-$\texttt{CD}^o$. We have $\texttt{tok}_r^{\gamma_1} = \bigstar$. Let us prove by contradiction that $D_r^{\gamma_1} = \{r\}$, and suppose r has one child u with $\texttt{tok}_u^{\gamma_1} \neq \bot$. Since $u \notin D_r^{\gamma_0}$, u is not activated during $\gamma_0 \to \gamma_1$, and thus $\texttt{tok}_u^{\gamma_0} \neq \bot$. According to Lemma 8, $\exists a \in \mathbb{A}^* : u \in D_a$. As a consequence, $u \in D_a \cap D_r^{\gamma_1}$ which cannot be according to Lemma 50, and therefore $D_r^{\gamma_1}$ is a clean branch-$\texttt{CD}^o$.

Let us now suppose that the property holds at γ_t, and prove it at γ_{t+1}. Remark first that if no node u joins D_r by executing $\mathbb{R}_{\texttt{Win}}$, then the guards of the rules that update $\texttt{tok}$ guarantee that no error can be generated.

Furthermore, following the same idea as what was presented for $D_r^{\gamma_1}$, a node u can join D_r only if it has a parent v in D_r such that $\texttt{tok}_v = \downdownarrows$, otherwise u would be in the intersection of D_r and one other $\texttt{CD}^o$.

Therefore, we only consider the different cases which imply $\downdownarrows$, the others being trivial. Let us consider v_k the last node of the clean branch-$\texttt{CD}^o$ $D_r^{\gamma_t}$.

- If $\texttt{tok}^\gamma_{v_k} = \downdownarrows$ then $\forall i \in [1, k-1], \texttt{tok}_{v_i} = \downarrow$ and are not enabled. If the player of v, u, executes $\mathbb{R}_{\texttt{Win}}$, then $\texttt{c}_u^{\gamma_{t+1}} = \texttt{c}_v^{\gamma_t} = \texttt{c}_r^{\gamma_{t+1}}$ and therefore $D_r^{\gamma_{t+1}}$ is a branch-$\texttt{CD}^o$.
  Furthermore, no node reaches $\texttt{Players}^\gamma(v_k)$ since $\mathbb{R}_{\texttt{ReplayUp}}$ implies an activation of another $\texttt{CD}^o$, and $\mathbb{R}_{\texttt{ReplayD}}$ is not enabled due to $\neg\texttt{ReplayL}$. Therefore,
  $\texttt{Players}^{\gamma_{t+1}} \cup \{u \in \mathcal{C}(v_k) : \texttt{tok}_u^{\gamma_{t+1}} = \bigstar\} \subseteq \texttt{Players}^{\gamma_t} \cup \{u \in \mathcal{C}(v_k) : \texttt{tok}_u^{\gamma_t} = \bigstar\}$, and thus the branch-$\texttt{CD}^o$ is clean.

- If $\texttt{tok}^\gamma_{v_k} = \bigstar \wedge \texttt{tok}^\gamma_{v_{k-1}} = \downdownarrows$ then $\forall i \in [1, k-2], \texttt{tok}_{v_i} = \downarrow$, and are not enabled, neither is v_k. Furthermore, $\texttt{Players}^\gamma(v_{k-1}) = \emptyset$ and no node reaches $\texttt{Players}^\gamma(v_{k-1})$ for the same reason as previously. Therefore, $D_r^{\gamma_{t+1}}$ is a clean branch-$\texttt{CD}^o$.
- If $\texttt{tok}^\gamma_v = \bullet$ then $\forall i \in [1, k-1], \texttt{tok}_{v_i} = \downarrow$ and are not enabled. Children u of v_k can only execute rules that update $\texttt{play}_u$ from P to $\{\texttt{L}, \texttt{W}, \texttt{F}\}$ and thus $\texttt{Players}^{\gamma'}(v) \subseteq \texttt{Players}^\gamma(v)$. If v executes $\mathbb{R}_{\texttt{OfferUp}}$ then nothing has to be proven. If v executes $\mathbb{R}_{\texttt{Give}}$ then $|\texttt{Players}^{\gamma_t}(v)| + |\{u \in \mathcal{C}^\gamma(v) : \texttt{tok}^\gamma_u = \bigstar\}| \leq 1$, and thus $D_r^{\gamma_{t+1}}$ is a clean branch-$\texttt{CD}^o$. ■

Corollary 2

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots$ be an execution such that r execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during $\gamma_0 \to \gamma_1$. Then no node executes $\mathbb{R}_{\texttt{FakeWin}}$ during ϵ.

Proof: Let us consider $cs = \gamma \to \gamma'$ one computing step of ϵ, and let us consider one node v such that $\texttt{tok}^\gamma_v = \perp$ and $\texttt{play}^\gamma_v = \texttt{P}$.

Suppose first that $\forall p \in \mathcal{P}(v)$ such that $\texttt{tok}^\gamma_p \notin \{\perp, \uparrow\}$ we have $p \in D^\gamma_r$. According to Lemmas 52 and 51, there exists at most 2 parents of v such that $\texttt{tok}^\gamma_p \neq \downarrow$, and if there are 2 then we have $\texttt{tok}_{p_1} = \circlearrowleft$ and $\texttt{tok}_{p_2} = \bigstar$. In any case, we have $\texttt{OkNeg}^\gamma(v)$ and thus v does not execute $\mathbb{R}_{\texttt{FakeWin}}$ during cs.

Let us now suppose that $\exists p \in \mathcal{P}(v)$ such that $\texttt{tok}^\gamma_p \notin \{\perp, \uparrow\}$ and $p \notin D^\gamma_r$. Since $\texttt{tok}_p \neq \perp$, according to Lemma 8 there exists $a \in \mathbb{A}^*$ such that $p \in D_a$. But then, since $v \in \mathbb{B}(D_a)$, v does not execute $\mathbb{R}_{\texttt{FakeWin}}$ since it would activate D_a. ■

From now on, we suppose that the executions start after at least one $\mathbb{R}^r_{\texttt{NewDFS}}$ and thus we suppose that no node executes $\mathbb{R}_{\texttt{FakeWin}}$ during the executions.

We presented in Definition 14 and in Theorem 1 the notion of circulation rounds for the root. Hence, for the sake of the proof, we also introduce the notion of circulation round on nodes that are not the root. This will allow us to consider circulation round on any node $v \in V$, root or not. Namely, this corresponds to the part of a circulation round in which one particular node holds the token, or in which the token is in one of its descendants.

Definition 28 (*Circulation Round for $v \neq r$*)

Let $\epsilon = \gamma_0 \to \cdots \to \gamma_k$ be a finite execution.

We say that $v \neq r$ executes a circulation round during ϵ if v executes $\mathbb{R}_{\texttt{Win}}$ and $\mathbb{R}_{\texttt{Return}}$, exactly once during ϵ, in that order.

We call circulation round of v during ϵ, and denote $\epsilon_{rt}(v, \epsilon)$ the subexecution of ϵ : $\gamma_{i+1} \to \cdots \to \gamma_j$ where $\gamma_i \to \gamma_{i+1}$ (resp. $\gamma_j \to \gamma_{j+1}$) is the computing step of ϵ where r executes $\mathbb{R}_{\texttt{Win}}$ (resp. $\mathbb{R}_{\texttt{Return}}$).

Lemma 53

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots \to \gamma_k$ be a circulation round of v. Then

1. $\forall i \in [0, k], \texttt{c}^{\gamma_i}_v = \texttt{c}^{\gamma_0}_v$
2. $\forall u \in \mathcal{C}(v), \texttt{tok}^{\gamma_0}_u = \perp$
3. $\texttt{tok}^{\gamma_k}_v = \uparrow$ *and* $\texttt{MayDropUp}^{\gamma_k}(v)$.

Proof: Let us prove the three items separately.

1. If $v = r$ then v updates $\texttt{c}_v$ only with the action of $\mathbb{R}^r_{\texttt{NewDFS}}$, and it does not execute $\mathbb{R}^r_{\texttt{NewDFS}}$ during ϵ by definition.

If $v \neq r$ then v updates $\mathtt{c}_v$ only with the action of $\mathbb{R}_{\mathtt{FakeWin}}$, which is not executed according to Corollary 2, or with the action of $\mathbb{R}_{\mathtt{Win}}$, which is not executed by v during ϵ by definition.

2. Since v executes a circulation round, it is activated and thus belongs to D_r. According to Lemma 52, $\neg\mathtt{Er}^{\gamma_0}(v)$. Furthermore, by definition of circulation round, we have $\mathtt{tok}_v^{\gamma_0} \in \{\bullet, \bigstar\}$. Thus, $\forall u \in \mathcal{C}(v), \mathtt{tok}_u^{\gamma_0} = \perp$.
3. By definition of a circulation round, v executes $\mathbb{R}^r_{\mathtt{NewDFS}}$ or $\mathbb{R}_{\mathtt{Return}}$ during $\gamma_k \to \gamma_{k+1}$ and thus $\mathtt{tok}_v^{\gamma_k} = \uparrow$ and $\mathtt{MayDropUp}^{\gamma_k}(v)$.

■

In the following lemma, we establish that if one node v executes a circulation round, then its playing children also execute a circulation round. This will allow us in the latter to reason by induction.

Lemma 54

Let us consider one node $v \in V$, and let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots \to \gamma_k$ be a circulation round of v. Then $\forall u \in \mathtt{Candidates}^{\gamma_0}(v), u$ executes a circulation round during ϵ.

Proof: Let us consider $u \in \mathtt{Candidates}^{\gamma_0}(v)$. According to Lemma 53, $\forall i \in [0,k], \mathtt{c}_v^{\gamma_i} = \mathtt{c}_v^{\gamma_0}$ Since $\mathtt{c}_u^{\gamma_0} \neq \mathtt{c}_v^{\gamma_0}$, we deduce that $\forall i \in [0,k], \mathtt{c}_v^{\gamma_i} \neq \mathtt{c}_u^{\gamma_0}$. Lemma 53 also assures that $\mathtt{tok}_u^{\gamma_0} = \perp$ which implies that, during ϵ, u cannot execute $\mathbb{R}_{\mathtt{Return}}$ before it executes $\mathbb{R}_{\mathtt{Win}}$. Let us first prove that u executes $\mathbb{R}_{\mathtt{Win}}$ during ϵ.

Whether $v = r$ or $v \neq r$ has no incidence on the fact that, necessarily, v executes $\mathbb{R}_{\mathtt{OfferUp}}$ during ϵ. Otherwise, we would have $\mathtt{tok}_v^{\gamma_k} \notin \{\perp, \uparrow\}$ which is inconsistent. Let us name $cs_s = \gamma_s \to \gamma_{s+1}$ the computing step during which v executes $\mathbb{R}_{\mathtt{OfferUp}}$. We deduce that $\mathtt{c}_u^{\gamma_s} = \mathtt{c}_v^{\gamma_s} (\neq \mathtt{c}_u^{\gamma_0}) \vee \mathtt{play}_u^{\gamma_s} \in \{\mathtt{W}, \mathtt{F}\}$. Since we consider executions where nodes do not execute $\mathbb{R}_{\mathtt{FakeWin}}$, $\mathbb{R}_{\mathtt{Win}}$ is the only rule whereby u can update $\mathtt{c}_u$, and also the only rule whereby u can update $\mathtt{play}_u$ from $\{\mathtt{P}, \mathtt{L}\}$ to $\{\mathtt{W}, \mathtt{F}\}$. We conclude that u executes $\mathbb{R}_{\mathtt{Win}}$ during $\gamma_0 \to \cdots \to \gamma_s$, let us say during $cs_t = \gamma_t \to \gamma_{t+1}$.

Let us now prove that u executes $\mathbb{R}_{\mathtt{Win}}$ only once during ϵ. Let us reason by contradiction and suppose there exists a second computing step $cs = \gamma \to \gamma'$ where u executes $\mathbb{R}_{\mathtt{Win}}$. Since D_r^γ is a branch-$\mathtt{CD}^o$, then $\forall w \in D_r^\gamma, \mathtt{c}_w^\gamma = \mathtt{c}_v^\gamma = \mathtt{c}_u^\gamma$. Indeed, u does not change its color between the first and the second execution of $\mathbb{R}_{\mathtt{Win}}$ by definition. But because u executes $\mathbb{R}_{\mathtt{Win}}$, $\exists p \in \mathcal{P}_{\mathtt{neq}}{}^{\gamma}(u) : \mathtt{tok}_p^\gamma = \downdownarrows$. This parent is necessarily in one other $\mathtt{CD}^o$, which is activated by u during cs, which is contradictory.

Since v executes $\mathbb{R}_{\mathtt{OfferUp}}$ during cs_s, we have $\mathtt{tok}_v^{\gamma_s} = \bullet \wedge \neg\mathtt{Er}^{\gamma_s}(v)$ which implies that $\mathtt{tok}_u^{\gamma_s} = \perp$, and because u executes $\mathbb{R}_{\mathtt{Win}}$ during $\gamma_0 \to \cdots \to \gamma_a$, it means that u executes $\mathbb{R}_{\mathtt{Return}}$ after having executed $\mathbb{R}_{\mathtt{Win}}$ and before γ_s.

Consequently, u executes one circulation round during ϵ. ■

We now introduce the notion of Free DODAG. This notion is defined only on reset points, *i.e.* configuration immediately subsequent to an execution of $\mathbb{R}^r_{\mathtt{NewDFS}}$ (see Definition 14). A Free DODAG corresponds to all the nodes which can reach the root by a path in G, such that no node of this path has any parent in one other $\mathtt{CD}^o$. It corresponds to nodes which have the possibility to receive the token sent from the root by executing $\mathbb{R}_{\mathtt{Win}}$ (we do not take into account the color of the nodes on this path in this definition).

Definition 29 (*Free DODAG*)

Let γ be a reset point of an execution ϵ. We call free DODAG in γ, and denote $\mathtt{FD}^o(\gamma)$, the smallest set that contains r and such that $\forall w \neq r$:

$$w \in \mathtt{FD}^o(\gamma) \iff \mathtt{tok}_w = \perp \wedge \exists p \in \mathcal{P}(w) : p \in \mathtt{FD}^o(\gamma) \wedge \forall p \in \mathcal{P}(w), (p = r \vee \mathtt{tok}_p = \perp)$$

Remark 9

$\texttt{FD}^o(\gamma)$ is a sub-$\texttt{D}^o$ of G anchored at r: $\forall u, v \in \texttt{FD}^o(\gamma), u \in \mathcal{C}_{\texttt{FD}^o{}_{(\gamma)}}(v) \iff u \in \mathcal{C}_G(v)$

Lemma 55

Let γ^1 and γ^2 be two reset points of an execution. We have $\texttt{FD}^o(\gamma^1) = \texttt{FD}^o(\gamma^2)$.

Proof: The set of nodes v such that $\texttt{tok}_v^{\gamma^1} \neq \bot$ is the union of $D_r^{\gamma^1} = \{r\}$ and of all $\texttt{CD}^o$'s $D_a^{\gamma^1}, a \in \mathbb{A}(\gamma^1)$. By hypothesis, the $\texttt{CD}^o$'s anchored to a node of $\mathbb{A}$ are not activated during the execution, and thus the set of nodes v such that $\texttt{tok}_v \neq \bot$ is the same in γ^1 as in γ^2. As a consequence, $\texttt{FD}^o(\gamma^1) = \texttt{FD}^o(\gamma^2)$. ■

$\texttt{FD}^o$ corresponds to all the nodes that may receive the token from the root, without considering variables $\texttt{c}$ and $\texttt{play}$. In order to gain in precision, we define the set of all nodes which will actually receive the token during the starting circulation round.

Definition 30 (*Reachable area from r*)

Let γ be a reset point. We define the reachable area from r at γ, and denote $\texttt{Reach}_r(\gamma)$:

$$\texttt{Reach}_r(\gamma) = \{v_k \mid \exists v_1 \cdots v_k \in \texttt{FD}^o(\gamma) : \left\{ \begin{array}{l} v_1 = r \wedge \forall i \geq 2, v_i \in \mathcal{C}(v_{i-1}) \\ \forall i \geq 2, (\texttt{play}_{v_i}^{\gamma} \in \{\texttt{P}, \texttt{L}\} \wedge \texttt{c}_{v_i}^{\gamma} \neq \texttt{c}_r^{\gamma}) \end{array} \right. \}$$

We prove in the following Lemma that nodes in the Reachable area actually receive the token, and execute exactly one circulation round during the circulation round of the root.

Lemma 56

Let $\epsilon = \gamma_0 \to \gamma_1 \to \cdots \to \gamma_k$ be a circulation round of r. Then $\forall u \in \texttt{Reach}_r(\gamma)$, u executes exactly one circulation round during ϵ.

Proof: Let us prove that lemma by induction on the depth of node u in $\texttt{Reach}_r(\gamma)$. The base case comes immediately, since r is the only node with depth 0 and by definition, r executes a circulation round during ϵ.

Let us now establish the induction step and suppose that for any node u with depth less that d of $\texttt{Reach}_r(\gamma)$, u executes a circulation round during ϵ. Let us consider one node $v \in \texttt{Reach}_r(\gamma)$ with depth $d+1$, and one of its ancestors $u \in \texttt{Reach}_r(\gamma)$ with depth d. By hypothesis, u executes a circulation round during ϵ. Let us consider $\epsilon_{rt}(u, \epsilon) = \gamma_i \to \cdots \to \gamma_j$ the circulation round of u during the circulation round of r. If $v \in \texttt{Candidates}^{\gamma_i}(u)$ then we can apply Lemma 54 and thus v executes a circulation round during $\epsilon_{rt}(u, \epsilon)$. If $v \notin \texttt{Candidates}^{\gamma_i}(u)$ then $\texttt{play}_v^{\gamma_i} \notin \{\texttt{P}, \texttt{L}\} \vee \texttt{c}_v^{\gamma_i} \neq \texttt{c}_u^{\gamma_i}$. This is possible only if, before γ_i v executes $\mathbb{R}_{\texttt{Win}}$. According to Lemma 53, $\texttt{tok}_v^{\gamma_i} = \bot$. Thus v executes at least one circulation round during ϵ.

Furthermore, v cannot execute several circulation round during ϵ, since v cannot activate any other $\texttt{CD}^o$ than D_r^{γ}, and after its first circulation round it has the same color as the root, and thus the same color as any node of the branch-$\texttt{CD}^o$. ■

The token does not only circulate through the reachable area of r indefinitely, it also makes its reachable area increase at each circulation round. Indeed, suppose one node v has the token, and has one child u with the same color as v. Then u is already seen as visited, which means that it won't receive the token, as being out of the reachable area. Yet, when sending the token upwards to its parent, v forces u to reset $\texttt{play}_u$ to $\texttt{P}$, which means that in the next circulation round, when the color of the token has switched, u will be able to receive the token. Note that this only works if $u \in \texttt{FD}^o(\gamma)$, otherwise u has other parents, and therefore updates $\texttt{play}_u$ to $\texttt{F}$.

Also note that if one child u does not receive the token for its variable play is already set to W or F, but had the appropriate color to receive it, then u will have to wait two circulation rounds before it actually receives the token. To embrace this behavior, we define the expansion areas of r, which constitutes all nodes in $\texttt{FD}^o(\gamma)$ which have a parent in $\texttt{Reach}_r(\gamma)$ without being in $\texttt{Reach}_r(\gamma)$ themselves.

Definition 31 (*Expansion areas of r*)

Let γ be a reset point. We define the expansion area of order 1 and of order 2 of r at γ, and denote $\texttt{Exp}_r^1(\gamma)$ and $\texttt{Exp}_r^2(\gamma)$:

$$\texttt{Exp}_r^1(\gamma) = \{u \in \texttt{FD}^o(\gamma) \mid \exists p \in \mathcal{P}(u) : p \in \texttt{Reach}_r(\gamma) \wedge \texttt{c}_u^\gamma = \texttt{c}_r^\gamma\}$$

$$\texttt{Exp}_r^2(\gamma) = \{u \in \texttt{FD}^o(\gamma) \mid \exists p \in \mathcal{P}(u) : p \in \texttt{Reach}_r(\gamma) \wedge \left\{ \begin{array}{l} \texttt{play}_u^\gamma \in \{\texttt{W},\texttt{F}\} \\ \texttt{c}_u^\gamma \neq \texttt{c}_r^\gamma \end{array} \right. \}$$

The following Lemma proves that the three sets defined above are jointly increasing, as intended. This will allow us to prove that at some point, the reachable area contains the entire $\texttt{FD}^o$.

Lemma 57

Let γ^1 and γ^2 be two consecutive reset points of an execution, and let $\epsilon = \gamma_0 \to \cdots \to \gamma_k$ be the circulation round of r between γ^1 and γ^2.

1. $\texttt{Reach}_r(\gamma^1) \subset \texttt{Reach}_r(\gamma^2)$
2. $\texttt{Exp}_r^1(\gamma^1) \subset \texttt{Reach}_r(\gamma^2)$
3. $\texttt{Exp}_r^2(\gamma^1) \subset \texttt{Exp}_r^1(\gamma^2)$

Proof: 1. Remark that $r \in \texttt{Reach}_r(\gamma^2)$ by definition. Let us consider one node $u \in \texttt{Reach}_r(\gamma^1), u \neq r$, and prove that $\texttt{play}_u^{\gamma^2} \in \{\texttt{P},\texttt{L}\} \wedge \texttt{c}_u^{\gamma^2} \neq \texttt{c}_r^{\gamma^2}$.

According to Lemma 56, u executes one circulation round during ϵ. After ϵ, u cannot execute $\mathbb{R}_{\texttt{Win}}$ after, at least, $\texttt{tok}_r = \downdownarrows$ which occurs after γ^2. Thus, in γ^2, u still has the color it took when reaching D_r during ϵ Thus, $\texttt{c}_u^{\gamma^2} = \texttt{c}_r^{\gamma^1} \neq \texttt{c}_r^{\gamma^2}$

By definition of $\texttt{Reach}_r$, $\exists v \in \mathcal{P}(u)$ such that $v \in \texttt{Reach}_r(\gamma^1)$. According to Lemma 56, v executes a circulation round during ϵ, and thus v executes $\mathbb{R}_{\texttt{OfferUp}}$ during ϵ. Let us consider $cs_i = \gamma_i \to \gamma_{i+1}$ the last computing step of ϵ such that one parent v of u executes $\mathbb{R}_{\texttt{OfferUp}}$. At γ_i, u terminated its circulation round. Indeed, if at γ_i, u has not begun its circulation round, then $\texttt{c}_u^{\gamma_i} \neq \texttt{c}_v^{\gamma_i}$ and $\texttt{play}_u^{\gamma_i} \in \{\texttt{P},\texttt{L}\}$ which would prevent v to execute $\mathbb{R}_{\texttt{OfferUp}}$. Furthermore, since $\neg\texttt{Er}^\gamma(v)$, we have $\texttt{tok}_u^{\gamma_i} = \bot$ so u terminated its circulation round at γ_i.

Consequently, we have $\texttt{c}_u^{\gamma_i} = \texttt{c}_r^{\gamma_i}$ and since $\neg\texttt{WaitSib}^{\gamma_i}(v)$, we also have $\texttt{play}_u^{\gamma_i} \neq \texttt{F}$. Furthermore, according to Lemma 50, we know that u has no parent such that $\texttt{tok}_p^{\gamma_i} \notin \{\bot, \uparrow\}$ apart from $D_r^{\gamma_i}$, which implies that u has only one parent such that $\texttt{tok}_p^{\gamma_i} \notin \{\bot, \uparrow\}$, which is v. Consequently, since v is the last parent of u that executes a circulation round, then from γ_{i+1} to the end of ϵ, all the parents of u are such that $\texttt{tok}_p^\gamma \in \{\bot, \uparrow\}$. Thus, during that part of the execution, we have constantly $\neg\texttt{FakeReplay}(u)$, and thus for that entire part of the execution, $\texttt{play}_u \neq \texttt{F}$.

Let us now consider γ_j the last configuration that is part of the circulation round of v. According to Lemma 53 we have $\texttt{MayDropUp}^{\gamma_j}(v)$, so $\texttt{play}_u^{\gamma_j} \neq \texttt{W}$ and thus $\texttt{play}_u^{\gamma_j} \in \{\texttt{P},\texttt{L}\}$. Since v does not execute $\mathbb{R}_{\texttt{Win}}$ nor $\mathbb{R}_{\texttt{FakeWin}}$ after its first circulation round of ϵ, we deduce $\texttt{play}_u^{\gamma^2} \in \{\texttt{P},\texttt{L}\}$

Since this is true for all the nodes of $\texttt{Reach}_r(\gamma^1)$, we can conclude $\texttt{Reach}_r(\gamma^1) \subseteq \texttt{Reach}_r(\gamma^2)$

2. Let us consider one node $u \in \texttt{Exp}_r^1(\gamma^1)$. By definition of $\texttt{Exp}_r^1$, there exists $p \in \mathcal{P}(v) : p \in \texttt{Reach}_r(\gamma^1)$. According to what we established above, $p \in \texttt{Reach}_r(\gamma^2)$, so we only have to prove that $\texttt{c}_u^{\gamma^2} \neq \texttt{c}_r^{\gamma^2} \wedge \texttt{play}_u^{\gamma^2} \in \{\texttt{P}, \texttt{L}\}$ to establish that $u \in \texttt{Reach}_r(\gamma^2)$.
   By definition, we have $\texttt{c}_u^{\gamma_0} = \texttt{c}_r^{\gamma_0}$. Since r does not update c_r during ϵ, and since D_r is a branch-$\texttt{CD}^o$ along ϵ, then until u executes $\mathbb{R}_{\texttt{Win}}$, it has the same color as all the nodes of D_r^γ during ϵ. Since, on the other hand, u cannot activate any other $\texttt{CD}^o$, we deduce that u actually does not execute $\mathbb{R}_{\texttt{Win}}$ during ϵ, neither during $\gamma_k \to \gamma^2$ for the same reason. Consequently, we have $\texttt{c}_u^{\gamma^2} \neq \texttt{c}_r^{\gamma^2}$.
   Now, the proof we made for the previous case totally applies: we can select the last computing step such that one parent p of u executes $\mathbb{R}_{\texttt{OfferUp}}$, at that moment we have $\texttt{c}_u = \texttt{c}_p$, thus we can infer from $\texttt{WaitSib}(v)$ that $\texttt{play}_u \neq \texttt{F}$ and from the non-activation of other $\texttt{CD}^o$'s that this remains true until at least γ^2. FInally, we can reason about `MayDropUp` as well to establish that $\texttt{play}_u^{\gamma^2} \in \{\texttt{P}, \texttt{L}\}$ which terminates the proof.
3. Let us consider one node $u \in \texttt{Exp}_r^2(\gamma^1)$. By definition of $\texttt{Exp}_r^2$, there exists $p \in \mathcal{P}(v) : p \in \texttt{Reach}_r(\gamma^1)$. According to what we established above, $p \in \texttt{Reach}_r(\gamma^2)$, so we only have to prove that $\texttt{c}_u^{\gamma^2} = \texttt{c}_r^{\gamma^2}$ to establish that $u \in \texttt{Reach}_r(\gamma^2)$.
   By definition, we have $\texttt{c}_u^{\gamma_0} \neq \texttt{c}_r^{\gamma_0} \wedge \texttt{play}_u^{\gamma_0} \in \{\texttt{W}, \texttt{F}\}$. Thus, u cannot execute $\mathbb{R}_{\texttt{Win}}$ before it executes $\mathbb{R}_{\texttt{ReplayUp}}$. Since u does not activate any other $\texttt{CD}^o$ than D_r, u executes $\mathbb{R}_{\texttt{ReplayUp}}$ only if it has one parent $p \in D_r$ such that $\texttt{tok}_p = \uparrow$ and $\texttt{c}_p = \texttt{c}_v$. But this cannot happen in ϵ before u executes $\mathbb{R}_{\texttt{Win}}$, since according to Lemma 52, all the nodes of D_r have the same color $\texttt{c}_r^{\gamma_0}$, which is the opposite that $\texttt{c}_u$ until it executes $\mathbb{R}_{\texttt{Win}}$. We just found a loop of dependencies.
   We prove that u cannot update $\texttt{c}_u$ before at least γ^2, and thus $u \in \texttt{Exp}^1(\gamma^2)$. ■

The two following lemmas prove that the sets of reachable and expansion areas of r allow a permanent growth of `Reach`, which stops only when `Reach` includes all the nodes of $\texttt{FD}^o$.

Lemma 58

Let γ be a reset point of an execution.
$\big(\texttt{Reach}_r(\gamma) = \texttt{FD}^o(\gamma)\big) \vee \big(\texttt{Exp}_r^1(\gamma) \cup \texttt{Exp}_r^2(\gamma) \neq \emptyset\big)$

Proof: By contradiction, suppose $\texttt{Reach}_r(\gamma) \neq \texttt{FD}^o(\gamma)$, and let us consider one node $u \in \texttt{FD}^o(\gamma) \setminus \texttt{Reach}_r(\gamma)$ with minimal depth.

This node has a parent in $\texttt{FD}^o(\gamma) \cap \texttt{Reach}_r(\gamma)$ (otherwise it would not be minimal in depth). Thus, u belongs to $\texttt{Exp}_r^1(\gamma) \cup \texttt{Exp}_r^2(\gamma)$. ■

Lemma 59

There exists a reset point γ_f such that $\texttt{Reach}_r(\gamma_f) = \texttt{FD}^o(\gamma_f)$.

Proof: Let us consider $\gamma_1, \gamma_2, \cdots$ an infinite sequence of reset points. According to Lemma 58, we know that if $\texttt{Reach}_r(\gamma_i) \neq \texttt{FD}^o(\gamma_i)$ then $\texttt{Exp}_r^1(\gamma_i) \cup \texttt{Exp}_r^2(\gamma_i) \neq \emptyset$. But then, according to Lemma 57, we deduce that $\texttt{Reach}_r(\gamma_i) \subsetneq \texttt{Reach}_r(\gamma_{i+1})$ or $\texttt{Exp}_r^1(\gamma_i) \subsetneq \texttt{Exp}_r^1(\gamma_{i+1})$.

If we now suppose that $\forall i, \texttt{Reach}_r(\gamma_i) \neq \texttt{FD}^o(\gamma_i)$, then we build an infinite sequence of growing sets of nodes, which cannot happen. By contradiction, we conclude. ■

Since we can now apply the properties of `Reach`, and especially Lemma 56 to $\texttt{FD}^o$, we can conclude that at some point, $\texttt{FD}^o = G$.

Theorem 18

$\texttt{FD}^o(\gamma_f) = G$

Proof: Let us reason by contradiction and suppose there exists one node $u \notin \texttt{FD}^o(\gamma_f)$. Let us consider one such node u with maximal height, *i.e.* such that $\forall p \in \mathcal{P}(u), p \in \texttt{FD}^o(\gamma_f)$. By definition this implies that $\forall p \in \mathcal{P}(u), \texttt{tok}_p^\gamma = \bot \vee p = r$. Thus, $u \notin \texttt{FD}^o(\gamma_f)$ is possible only by $\texttt{tok}_v^{\gamma_f} \neq \bot$. Since γ_f is a reset point, it means that $\exists a \in \mathbb{A}^* : v \in D_a$, and thus v is not activated at all.

Let us now consider one parent p of v. According to Lemma 59, we have $p \in \texttt{Reach}_r^{\gamma_f}$, and according to Lemma 56, p executes a circulation round after γ_f. But thus, according to Lemma 53 when p starts its circulation round we have $\texttt{tok}_v = \bot$ which is contradictory. ■

Corollary 3

$\texttt{Reach}_r(\gamma_f) = G$.

Theorem 19

Our algorithm is a fair self-stabilizing token circulation algorithm.

Proof: We already proved in Theorem 17 that our algorithm satisfies Condition 2 of Specification 1.

Let us consider one execution $\epsilon = \gamma_0 \to \cdots$, and more precisely the subexecution $\epsilon_f = \gamma_f \to \cdots$

Remark that having $\texttt{FD}^o(\gamma_f) = G$ implies that all nodes v but the root are such that $\texttt{tok}_v^{\gamma_f} = \bot$. Therefore, $\mathbb{A}^*(\gamma_f) = \emptyset$, and by Lemma 9, this is true in all configurations of ϵ_f. In particular, in any configuration of ϵ_f, the set of nodes with $\texttt{tok}_v \neq \bot$ corresponds to the clean branch-$\texttt{CD}^o$ D_r^γ, and by construction, we have $U_{\mathcal{TC}}(\gamma)$. This proves that our algorithm satisfies Condition 1 of Specification 1.

Furthermore, let us consider one circulation round of ϵ_f which starts at γ^1, and let us denote γ^2 the reset point consecutive to γ^1. According to Lemma 56, all nodes of $\texttt{Reach}_r(\gamma^1) = G$ execute exactly one circulation round during that circulation round. Since $\forall v \in G \setminus \{r\}, \texttt{tok}_v^{\gamma^1} = \texttt{tok}_v^{\gamma^2} = \bot$, this means that there is exactly one computing step in which v executes $\mathbb{R}_{\texttt{Win}}$, and thus there exists a finite, continuous, subexecution of this round in which $R(v)$. Therefore, this circulation round is fair. ■

C.4 Space Optimality of our Algorithm

In this section, we prove that our algorithm is asymptotically optimal in terms of memory for the token circulation problem. Our algorithm has a space complexity $\mathbb{S}(\mathcal{A}) \sim \log\log n$ bits per node.

Thus, we are going to prove that there does not exist any deterministic algorithm solving $\mathcal{TC}$ in $\mathcal{S}$ using $o(\log\log n)$ bits per node. We prove this lower bound under less challenging hypotheses than those under which our algorithm works. Namely, our algorithm is self-stabilizing, it works under the unfair distributed scheduler and in any DODAG. Our lower bound is established for general distributed algorithms (without any stabilization requirements), under simpler schedulers (the synchronous scheduler and the central strongly fair scheduler), and in a very simple class of $\texttt{D}^o$: the *bi-branch* $\texttt{D}^o$.

We basically use Theorem 3 established in [3] which introduces the equivalence between executions of small-memory algorithms in identified networks, and in anonymous networks. Specification 1 of $\mathcal{TC}$ relies on two local predicates, T and R. We can consider the function $(\mathcal{A}, T, R)$, which has asymptotically the same space complexity as $\mathcal{A}$, and to apply Theorem 3 to this new function.

Let us first define the topology used to prove our lower bound. A bi-branch $\texttt{D}^o$ is a $\texttt{D}^o$ in which the root has two children, and all the other nodes have one child, and one parent.

Although we call it $\mathtt{D}^o$, it actually a tree, with bounded degree. For simplicity, we provide the set of edges of the bi-branch as a set of oriented edges, which matches the relation $\rightarrow$.

Definition 32 (*bi-branch* $\mathtt{D}^o$ B_k)

For any $k \geq 1$, we define B_k the k-th bi-branch $\mathtt{D}^o$ by $B_k = (V_k, E_k)$ where:

- $V_k = \{r, u_1, u_2, \cdots, u_k, v_1, v_2, \cdots, v_k\}$,
- $E_k = \{(u_1, r), (u_2, u_1), \cdots, (u_k, u_{k-1}), (v_1, r), (v_2, v_1), \cdots, (v_k, v_{k-1})\}$.

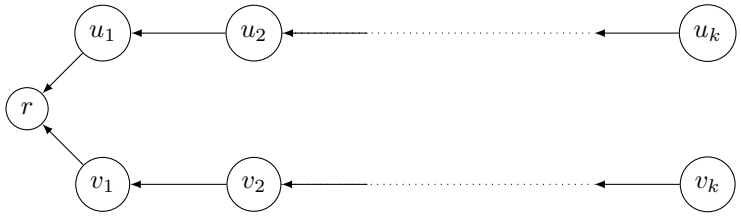

Figure 28: Bi-branch $\mathtt{D}^o$ B_k

We define the symmetric configurations on bi-branch $\mathtt{D}^o$.

Definition 33 (*Symmetric configurations on B_k*)

Let us consider $k \geq 1$. A configuration of B_k, γ, is said symmetric if
$\forall i \in [1, k], \mathtt{state}^{\gamma}_{u_i} = \mathtt{state}^{\gamma}_{v_i}$.

Theorem 20 (*Lower Bound for $\mathcal{TC}$ in $\mathcal{S}$*)

Let $c > 1$. Every deterministic algorithm solving $\mathcal{TC}$ in $\mathcal{S}$ under a central strongly fair scheduler or under the synchronous scheduler requires registers of size $\Omega(\log\log n)$ bits per node in bi-branch $\mathtt{D}^o$'s with unique identifiers in $[1, n^c]$.

Proof: We first prove the theorem under the hypothesis of the central strongly fair scheduler, and will show later how to treat the case of the synchronous scheduler.

Let us suppose that there exists an algorithm $\mathcal{A}$ solving $\mathcal{TC}$ in $\mathcal{S}$ under a central strongly fair scheduler, using $o(\log\log n)$ bits per nodes in bi-branch $\mathtt{D}^o$'s with unique identifiers in $[1, n^c]$. By definition, there exist two local predicates T and R which specifies how $\mathcal{A}$ solves $\mathcal{TC}$.

The class $\mathcal{G} = (B_i)_{i \geq 1}$ contains graphs of arbitrary large size, and is Δ-limited by $\log n$ since all the graphs it contains have degree 2. Therefore, we can apply Theorem 3. Rather than simply applying it to $\mathcal{A}$, we consider the tuple $(\mathcal{A}, T, R)$. Theorem 3 guarantees that, for any n large-enough, there exist n distinct identifiers $\mathtt{ID}_1, \ldots, \mathtt{ID}_n$ in $[1, n^c]$ which the tuple $(\mathcal{A}, T, R)$ does not distinguish.

Let us now consider one such large-enough network B_n, such that the identifiers of the nodes are $\mathtt{ID}_1, \ldots, \mathtt{ID}_n$. We consider the framework of distributed algorithms, where the initial configuration is not corrupted. Therefore, we suppose a configuration γ_0 such that the root holds the token (*i.e.* $T^{\gamma_0}(r) = \mathtt{true}$), and that all the other nodes are in one unique state. By definition, γ_0 is symmetric. Let us build step by step an infinite execution such that $\forall i \in \mathbb{N}, T^{\gamma_i}(r)$. This is clearly contradictory with the fairness of the token circulation.

We reason by induction, building step by step an execution in such that each one or two configuration is symmetric, and such that the root permanently holds the token. The base case is immediate: γ_0 respects both properties.

Let us suppose that γ_k is symmetric, and $T^{\gamma_k}(r) = \mathtt{true}$. By symmetry, we have $\forall i \in [1, k]$, for all rules R of $R_{\mathcal{A}}$, R is enabled on u_i if and only if it is enabled on v_i. In particular, u_i

is enabled if and only of v_i is enabled. Since by hypothesis $\mathcal{A}$ solves $\mathcal{TC}$, there is at least one enabled node.

If the central strongly fair scheduler activates r in γ_k, then the resulting configuration γ_{k+1} is symmetric. Predicate T only relies on the states of the nodes, since it cannot distinguish the identifiers by construction, and therefore $T^{\gamma_{k+1}}(u_i) \iff T^{\gamma_{k+1}}(v_i)$ by symmetry. By unicity of the token, we have $T^{\gamma_{k+1}}(r)$, and the induction step is established.

Suppose now that the central strongly fair scheduler activates u_i in γ_k. In this case, the scheduler can activate v_i in γ_{k+1}. Since there is no edge between u_i and v_i, the action of u_i does not have any incidence on v_i, which means that v_i takes the exact same step as u_i. As a result, γ_{k+2} is symmetric, and consequently r holds the token in γ_{k+2}. Finally, we prove that r also holds the token in γ_{k+1}. Since predicate T is local, and since only u_i is activated in γ_k, only neighbors of u_i or u_i itself may have their value on T updated between γ_k and γ_{k+1}. Let w be the node such that $T^{\gamma_{k+1}}(w) = \texttt{true}$, and suppose that $w \neq r$, *i.e.* $w = u_j$ with $j \in \{i-1, i, i+1\}$. Therefore, by symmetry and since v_i takes the exact same step as u_i, and is the only activated node in γ_{k+1}, $T^{\gamma_{k+2}}(v_j) = T^{\gamma_{k+2}}(u_j) = \texttt{true}$, which breaks the unicity of the token. As a consequence, r holds the token in γ_{k+1}, and the induction step is established.

Let us now consider the synchronous scheduler. All the enabled nodes are activated at each step, which preserves symmetry at each step, and thus the root permanently holds the token too. ■