Self-stabilizing Distributed Stable Marriage

Marie Laveau, George Manoussakis, Joffroy Beauquier, Thibault Bernard, Janna Burman, Johanne Cohen, Laurence Pilard

Self-stabilizing Distributed Stable Marriage

Regular submission. Eligible for the best student paper award.

Marie Laveau[1] ⋆, George Manoussakis[1], Joffroy Beauquier[1], Thibault Bernard[2,3], Janna Burman[1], Johanne Cohen[1], and Laurence Pilard[2]

[1] LRI, Université Paris-Sud, CNRS, Université Paris-Saclay, France
[2] LI-PaRAD, Université de Versailles, Université Paris-Saclay, France
[3] CReSTIC, Université de Reims Champagne Ardenne, France

Abstract. *Stable matching* (also called *stable marriage* in the literature) is a problem of matching in a bipartite graph, introduced in an economic context by Gale and Shapley. In this problem, each node has preferences for matching with its neighbors. The final matching should satisfy these preferences such that in no unmatched pair both nodes prefer to be matched together. The problem has a lot of useful applications (two sided markets, migration of virtual machines in Cloud computing, content delivery on the Internet, etc.). There even exists companies dedicated solely to administering stable matching programs. Numerous algorithms have been designed for solving this problem (and its variants), in different contexts, including distributed ones. However, to the best of our knowledge, none of the distributed solutions is *self-stabilizing* (self-stabilization is a formal framework that allows dealing with transient corruptions of memory and channels). We present a self-stabilizing stable matching solution, in the model of *composite atomicity* (*state-reading* model), under an *unfair distributed scheduler*. The algorithm is given with a formal proof of correctness and an upper bound on its time complexity in terms of *moves* and *steps*.

1 Introduction

1.1 Historical Background

Stable matching or equivalently *stable marriage* is a problem of matching in a bipartite graph, introduced in an economic context by Gale and Shapley [GS62]. It can be described by a natural example of marriage formations between a group of women and a group of men in some community (represented by two groups of nodes, each of size n, in a bipartite graph). As in the real life, each member of the community has preferences regarding other members. Assuming that the given group sizes are equal (*i.e.*, the bipartite graph is complete), the problem is to find a satisfactory marriage for each member with a member of the opposite sex. Satisfactory means that, in the final matching, there is no unmarried pair

⋆ Ph.D. student and Contact author. laveau@lri.fr, Bat 650, Rue Noetzlin, 91190, Gif-sur-Yvette, +33169156476

of a man and a woman such that they both prefer each other over their current spouses. One says then that there are no *blocking pairs* and the marriage or the matching is *stable*. In a game theory context, stable matching realizes a pure Nash equilibrium, given lists of preferences for both sides. Gale and Shapley showed that a stable matching always exists. It was shown by providing a centralized algorithm running in $O(n^2)$ time, which is proved to be asymptotically optimal (for centralized algorithms) in [GI89].

Stable matching has a lot of applications in economy and computer science. It can be viewed as a particular formulation of two sided matching markets that have been proved useful in the empirical study of many labor markets. Stable matching is used to assign graduating medical students to residency programs at hospitals in the US, Canada and Scotland, and to assign students to schools and universities in Norway and Singapore (*cf.* [Gol06]). In the domain of Cloud computing, stable matching is used for performing efficient migration of virtual machines to servers (*e.g.*, [XL11], [KL14]). Content delivery networks that distribute much of the world's content and services have to solve a large and complex stable matching problem between users and servers [MS15]. Finally, one can also notice that stable matching has applications in models without any hint of selfish agents, such as scheduling network switches [CGMP99].

Given this large potential application domain, it is not surprising that a lot of algorithms, each corresponding to a particular context and a problem variant, have been proposed and studied. For the studies on different problem variants in the centralized context, one can see for example the books by Knuth [Knu76], by Gusfield and Irving [GI89], or by Roth and Sotomayor [RS90].

The interest of the current work is a decentralized distributed setting, where the bipartite graph represents a communication network. Edges represent the communication links and nodes are computing entities (to be matched). Each node has only partial information about the problem instance, contrary to the centralized case. In particular, it is assumed to be initially aware only of its own preferences, but not of the preferences of the other nodes. In addition, to ensure confidentiality of the preferences [BM05] and avoid high message complexity, we follow previous studies and rule out a trivial solution where nodes exchange their preference lists and then run a known centralized solution at each node.

Studies on distributed stable matching appeared much later than the centralized versions. Among these studies, theoretical ones consider an idealized *synchronous* distributed communication model, where nodes progress in a lock-step manner, exchanging information and performing computations *all* together at each step (called *round*). These works focus on round complexity of the problem and its variants. Kipnis and Patt-Shamir [KPS09] were the first to study round complexity of the distributed stable matching. They proved a lower bound of $\Omega(\sqrt{(n/B \log n)})$, where B is the number of bits per message, and provided an algorithm that solves the distributed stable marriage in $O(n^2)$ rounds. Searching for better time complexity and conditions that can provide it, many studies considered specific restrictions on the preference lists. Consider for example the *weighted stable matching* in [AGL10], incomplete or bounded lists

in [OR15], [FKPS10], "almost regular" lists in [OR15] and "similarity" in preference lists in [PW16]). With the same goal in mind (of obtaining better time complexity), approximate versions of the stable marriage have been considered (*e.g.*, [KPS09], [FKPS10], [OR15]). Such versions can be solved in a polylogarithmic time and random algorithms can improve it even more. Furthermore, when assuming restrictions on preference lists, approximate stable marriage can be solved even in constant time (*cf.* [OR15] and [FKPS10]).

1.2 Overview of Results

Contrary to the previous works, we are interested in the stable marriage problem for an *asynchronous* distributed communication model. Additionally, we tackle the problem by providing a general type of a solution, called self-stabilizing [Dij74]. Such solution tolerates transient (or short-lived) failures (volatile memory corruptions) of any number of nodes. That is, it solves a problem from an arbitrary starting configuration (see a formal definition in the model section). This property is particularly interesting for Cloud and Internet based applications in general, since they frequently require (at least) some level of self-stabilization.

It is now described how we obtained such solution. First, notice that even though the original stable matching algorithm by Gale and Shapley (GSA) is essentially centralized, it can be interpreted as a distributed one [BM05] and most of the existing distributed algorithms rely on GSA. In general, the algorithm proceeds by iteratively realizing proposals, *e.g.*, by women, and acceptances, *e.g.*, by men. Intuitively speaking, the algorithm creates matches and *resolves* appearing blocking pairs, when improving iteratively the quality of the matches according to the preferences (dynamics "better match").

GSA has received a lot of attention, in particular by Knuth [Knu76]. When investigating combinatorial properties of the algorithm, Knuth discovered the possibility of cycles when executing GSA from some initial configurations with an incomplete matching. That is, GSA does not necessarily converge from any initial configuration towards correct configurations (due to the existence of cycles). In other words, it does not naturally tolerate transient failures that can put a system in an arbitrary configuration, *i.e.*, it is not self-stabilizing.

After this negative result, a step forward was taken by Roth and Vande Vate [RVV90] and by Ackermann *et al.* [AGM+11]. Both works present completely centralized strategies allowing to solve stable matching starting from any given matching. The strategy proposed by Roth and Vande Vate stores and consults a global access set of previously resolved blocking pairs and thus is inherently centralized. On the contrary, the strategy by Ackermann *et al.* [AGM+11] works in two phases. In the first one, only married women make proposals for improving their marriages. When no married woman can improve anymore, the second phase starts. In this phase, only unmarried women can make proposals (until they all are matched). At the end of this phase, a stable matching is obtained (after at most $O(n^2)$ steps). In this work, we adopt the main idea of these two phases.

Making this idea work in a distributed asynchronous and self-stabilizing way is still very challenging. First, there is a need of a sort of synchronization of phases between the nodes that cannot move all together to the next phase, like in the centralized case. Then, termination detection is needed for detecting the end of the first phase. Furthermore, Ackerman *et al.* supposed "best response" dynamics, contrary to the "better" ones in a distributed GSA. "Best response" dynamics are inherently centralized too, since creation or suppression of a match is not instantaneous (as it is in the centralized case) and the actual matches can change during the delay for realizing these actions. Hence, it is difficult to implement perfect "best response" dynamics. Finally, notice that a distributed matching has to be encoded with pointers that can be badly initialized. This is not taken into account in the algorithm of Ackerman *et al.*

In addition to these difficulties, we strive to provide a truly decentralized solution using neither leader nor global reset and detecting and correcting faults locally (similarly to the way GSA resolves blocking pairs). This rules out the known self-stabilizing automatic transformers requiring such type of primitives. On the positive side, this allows obtaining more efficient algorithms in terms of time and space. This is also the reason for not using known synchronization techniques (*e.g.*, [AKM+07], [BPV04]). Our algorithm works with only one additional phase of synchronization (in addition to the two phases in the strategy of Ackerman *et al.*), while using known synchronization techniques would result in much more additional phases.

The proposed algorithm works under an *unfair* distributed scheduler, *i.e.*, choosing at each step a subset of nodes that have actions to perform (*i.e.*, *eligible* or *enabled* nodes; see model section for a formal definition). In particular, some constantly eligible node may stay inactivated for an arbitrary period of time. In spite of all the mentioned difficulties, we design and prove such a self-stabilizing stable matching algorithm which also guarantees confidentiality of the preference lists. We present it together with its correctness proof and time complexity analysis providing an upper bound of $O(n^4)$ *moves* (activations changing the state of a node) or *steps* (activations changing the configuration of the system; see the model section).

2 Model

A distributed system is based on a set of nodes. Each node can communicate with a subset of other nodes, called its neighbors and denoted by $\mathcal{N}(v)$. Communication is assumed to be bidirectional. Hence, the topology of the system can be represented as a simple undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges, *i.e.*, communication links. We assume here that G is a complete bipartite graph $K_{n,n}$, over two subsets of nodes of equal size. We are interested in the stable matching problem, also called *stable marriage*. Following the terminology of [GS62], where the problem is introduced, we call women the n nodes of the first subset (Women) in the bipartite graph and men the n nodes of the second subset (Men). Each node has a unique identifier and

a complete list of n preferences for the nodes of the other set (each woman has a complete list of men and reciprocally). In other words, each women w is given with a *priority* for each man m, denoted $p(w, m)$, and reciprocally. The priorities go from 1 to n and the most preferred person have priority 1. The goal is to match (marry) the women and the men together such that everyone is matched and there is no unmarried pair (w, m) of a woman and a man, who both prefer each other to their current matches (partners) m' and w', *i.e.*, there is no pair (w, m) such that (w, m') and (w', m) are married, but $p(w, m) < p(w, m')$ and $p(m, w) < p(m, w')$. When there are no such pairs of people, called *blocking pairs* (BP), the set of marriages is deemed stable.

Remark 1. For technical reasons, we use in the proofs a definition of blocking pair that is more general than the definition given above, as it applies to incomplete matching. In the original definition, a blocking pair has to be a pair of already married persons. In the definition of BP used here, the man can be unmarried. Formally, a pair (w, m) of a woman w and a man m is blocking iff w is matched to m', m is matched to w' and w and m prefer each other to their actual matching, or, w is matched to m', m is unmatched and w prefer m to m'. Clearly enough, the two notions coincide if the matching is complete. The definition implies that a man prefers to be matched with any woman rather than to stay unmatched.

For designing solutions to this problem, we use the composite atomicity model of computation (*cf.* [Dij74] and [Gho14]) in which the nodes communicate using a finite number of locally shared variables. Each node can read its own variables and those of its neighbors, but can write only to its own variables. The *state* of a node is a vector of the values of its variables. A *configuration* of the system is a vector of states of all nodes. A distributed algorithm consists of one code per node. The code of a node v is a finite set of guarded rules of the following form:
Label: (* Comment *)
{Guard}
Actions
The labels are used to identify actions. The guard of a rule in the code of v is a Boolean expression involving the variables of v and of its neighbors. If the guard of some rule evaluates to true, then the rule is said to be *enabled* at v. By extension, v is said to be enabled or *eligible* if at least one of its rules is enabled. *Actions* represents a sequence of actions on v's variables. A rule can be executed (activated) only if it is enabled. In this case, its execution consists in performing the sequence of actions, using the values of the variables at the time of the guard evaluation. The asynchrony of the system is modeled by an adversary, called *scheduler*. In a configuration, the scheduler selects a non-empty subset of eligible nodes, then atomically evaluates the guards of one enabled rule per node (chosen non-deterministically), then, still atomically, executes the corresponding actions. This is called a *step* (or *transition*) and the activation of each rule in the step is called a *move*. Such a scheduler is called *distributed* in the literature (contrary to a *central* scheduler, choosing at each step only one enabled node, or to the *synchronous* scheduler that chooses all the enabled nodes). When a step is executed in the configuration C, it leads to a configuration

C' and we write $C \to C'$. We say that C' is *reached* from C, denoted by $C \xrightarrow{*} C'$, if $C \to C_1 \to C_2 \to ... \to C'$. An *execution* is a maximal sequence of configurations C_0, C_1, ... , C_k, ... such that $C_i \to C_{i+1}$ for all $i \geq 0$. The term "*maximal*" means that the execution is either infinite or ends in a *terminal configuration*, *i.e.*, a configuration in which no node is enabled. Different types of fairness, limiting the possible choices of the scheduler, appear in the literature. We do not make any such limitation, that is the schedulers we consider are *unfair*.

A distributed algorithm *solves* the stable marriage problem if each of its executions starting from a predefined initial configuration, under the unfair distributed scheduler, reaches a terminal configuration in which there is a stable marriage. A distributed algorithm solves the stable marriage problem in a *self-stabilizing* way if it solves it as above, but for any possible initial configuration. The relation between self-stabilization and transient failures is well known. Even if all the variables of all nodes have been corrupted once, (producing an arbitrary configuration possibly considered as initial), the algorithm reaches a terminal configuration in which there is a stable marriage. Hence, in some sense, it tolerates the transient failure, since it regains by itself a correct configuration, without any external intervention. Formally, let $\mathcal{A}$ be a distributed algorithm, let $\mathcal{C}$ be the set of its configurations and let $\mathcal{E}$ be the set of its executions, from any configuration in $\mathcal{C}$. Call *graph problem* a predicate $\mathcal{P}$ on configurations.

Definition 1. *$\mathcal{A}$ is self-stabilizing for $\mathcal{P}$ if and only if there exists a non-empty subset $\mathcal{L}$ of configurations of $\mathcal{C}$, such that:*

1. *(Closure) starting from any $C \in \mathcal{L}$, any reached configuration is in $\mathcal{L}$ (i.e., $\mathcal{L}$ is closed under $\to$) and any configuration in $\mathcal{L}$ satisfies $\mathcal{P}$,*
2. *(Convergence) any execution in $\mathcal{E}$ (starting from any configuration in $\mathcal{C}$), reaches a configuration in $\mathcal{L}$.*

The time complexity of a self-stabilizing distributed algorithm can be evaluated in terms of moves or steps. The *stabilization time* of a distributed algorithm, counted in moves (respectively in steps), is the maximum number of moves (resp. steps) to reach a configuration in $\mathcal{L}$, starting from an arbitrary configuration. The stabilization time in moves gives an upper bound on the stabilization time in steps.

3 Self-stabilizing Solution to Stable Marriage

As already noticed in Sec 1.2, the algorithm of Ackermann *et al.* ([AGM+11]) is inherently centralized. It proceeds in two phases. In the first phase, married women try to improve their marriage. When no improvement is possible, phase 2 starts. In this phase, single women try to marry their best free choice. In the first phase, women globally *reduce their regrets*, *i.e.*, change to a better priority spouse, and in the second phase, men do the same. The algorithm is correct, even when started from an incomplete matching, but is not self-stabilizing in the strict sense, because all nodes must start in phase 1 and change simultaneously

to phase 2. It could be made self-stabilizing easily because of the centralization, with the implementation of a global phase counter. Things are not so easy in a distributed asynchronous setting. The distributed self-stabilizing solution that we propose takes the idea of two phases, but use a supplementary phase for the purpose of synchronization. We number the phases 1, 1.5, 2. Phases 1 and 2 play about the same role as in Ackermann *et al.* algorithm.

Phase 1.5 is an intermediary phase solving synchronization problems between phase 1 and 2 (due to an erroneous initial configuration). During phases 1 and 2, women have the initiative to propose marriage, men can only choose among the different proposals.

The transition from phase 1 to phase 1.5 is realized first by women who have checked the lack of blocking pairs. Once all women are in phase 1.5, men can change to phase 1.5 if they did not detect blocking pairs. Otherwise, a man blocks the process (by staying in phase 1). The woman involved in the blocking pair will be activated and will change its phase to 1 (forcing all men to come back to phase 1). Only when all nodes reach phase 1.5, women can change to phase 2 and men will follow by changing to the same phase. The checking before entering phase 1.5 guaranties the lack of blocking pairs at the beginning and during phase 2.

Nodes can also change from phase 2 to phase 1 whenever a faulty configuration is detected. For example, this happens if it is detected that some pointers are badly initiated, if a man phase has a bigger value than the one of a women, or the phase values are not consecutive. This change can also be initiated by at a married woman in phase 2, who detects a possible improvement (*i.e.*, a blocking pair). All other nodes will detect the phase change and move to phase 1 too (without this, no one would change to 1.5).

We get the property that no execution cycles more than one time through phases 1, 1.5, 2. Similarly to the algorithm of Ackermann *et al.*, we show that, during the last execution of the first phase, the regrets of the married women are globally decreasing. This ensures that no blocking pair exists at the end of this phase. During the last execution of phase 2, it is the same for the regrets of men and ensured that no blocking pair can appear (even though the matching can be still incomplete). At the end, in $O(n^4)$ moves in overall, a complete stable marriage is obtained.

Now we precise the implementation of these ideas. Each nodes v has variables and constants. The variables can be read by the neighbors, but the access to constants is limited.

Variables:

- *marriage*: the *spouse* of v. The value is `Null`, if v is *single*.
- *proposal*: for a woman w, the node to whom w has *proposed*; for a man m, the woman whose proposal has been *accepted* by m. The value is `Null` if there is no proposal or acceptance.
- $phase \in \{1,\ 1.5,\ 2\}$: *$v$ is in phase α* if $v.phase\ = \alpha$.

We use the notation $var(C)$ for the value of var in the configuration C.

Constant:

- *pref*: the v's list of its n neighbors in preference order. The priority of the i^{th} element of the list is i. Then, the first element is the most preferred neighbor and its priority is 1.

Lists of preferences are kept secret. A node v only communicates to its neighbor u the priority it gives to u and the *priority* of its actual spouse. If v is single, the latter communicated priority is $n + 1$.

Functions:

- $\mathsf{p}(v,u)$: returns the priority of u in the preference list of v (this is also the regret of v, if v is married with u; otherwise the regret of v is $n + 1$).
- $\mathsf{max}(\mathcal{A})$: returns the most preferred node in a set $\mathcal{A}$ of nodes

Let $\mathcal{C}_v$ be the set of nodes which prefer v and are preferred by v to their corresponding spouses:
$\mathcal{C}_v = \{\ u\ \in\ \mathcal{N}(v)\ :\ \mathsf{p}(v,u) < \mathsf{p}(v,v.marriage) \wedge \mathsf{p}(u,v) < \mathsf{p}(u,u.marriage)\}$
The following function is used by women to determine which man to propose to.

- $\mathsf{BestMarriage}(v)$ = if $(\mathcal{C}_v\ \neq\ \emptyset)$ then `return` $\mathsf{max}(\mathcal{C}_v)$ else `return Null`

Let $\mathcal{P}_v$ be the set of women who: (a) are preferred by v to his own spouse; (b) prefer v to their own spouse; (c) have made a proposal to v; (d) are in the same phase as v; (e) are single, if their phase is 2, or with a spouse, if their phase is 1.

$$\begin{aligned}\mathcal{P}_v = \{u\ \in\ \mathcal{N}(v)\ :\ & u.proposal\ =\ v\ \wedge\ u.phase\ =\ v.phase\\ \wedge\ \mathsf{p}(v,u)\ <\ & \mathsf{p}(v,v.marriage)\ \wedge\ \mathsf{p}(u,v)\ <\ \mathsf{p}(u,u.marriage)\\ & \wedge\ [(u.marriage\ \neq\ \texttt{Null}\ \wedge\ u.phase\ =\ 1)\\ & \vee\ (u.marriage\ =\ \texttt{Null}\ \wedge\ u.phase\ =\ 2)]\}\end{aligned}$$

The following function is used only by men to determine which proposal to accept (the considered proposals have to be done by women in the same phase).

- $\mathsf{BestProposal}(v)$ = if $(\mathcal{P}_v\ \neq\ \emptyset)$ then `return` $\mathsf{max}(\mathcal{P}_v)$ else `return Null`

Predicates:

The solution we propose introduces some predicates, which are used for testing locally certain properties.

The predicate $\mathsf{Married}(v)$ below is used by a woman v for checking whether she is reciprocally *married* (`True`), or not (`False`).

- $\mathsf{Married}(v) \equiv\ (v.marriage\ \neq \texttt{Null})\ \wedge\ [(v.marriage.marriage\ =\ v)\ \vee\ (v.marriage.proposal\ =\ v)]$

The predicate $\mathsf{Response}(v)$ checks if the proposal of v has been *accepted*.

- $\mathsf{Response}(v) \equiv\ (v.proposal\ \neq \texttt{Null}) \wedge\ (v.proposal.proposal\ =\ v)$

The predicate $\mathsf{AlreadyEngaged}(v)$ is used by a man to detect if he already accepted a proposal.

– $\mathsf{AlreadyEngaged}(v) \equiv (v.proposal \neq \texttt{Null}) \wedge$
$[(v.proposal.proposal = v) \vee (v.proposal.marriage = v)]$

Since there is an asymmetry between women's proposals and men's acceptances (women ask first for a marriage and then men answer), they have different predicates to verify whether their pointers are correct and, in particular, that their marriages are reciprocal (suffix W in the predicate name refers to women and M to men). Otherwise, the predicate is `False` and pointers are said *incoherent*.

– $\mathsf{IncoherentPointersW}(v) \equiv (v.marriage \neq \texttt{Null})$
$\wedge [((v.marriage.marriage \neq v) \wedge (v.marriage.proposal \neq v)) \vee$
$(v.marriage = v.proposal)]$
– $\mathsf{IncoherentPointersM}(v) \equiv (v.marriage \neq \texttt{Null})$
$\wedge [(v.marriage.marriage \neq v) \vee (v.marriage = v.proposal)]$

Since the definition of blocking pair is asymmetrical (cf. Remark 1), there are two predicates for checking the presence of blocking pairs (which involves a married woman). If a node detects a blocking pair, we say that it is *involved* in a blocking pair. In other words, if at least one of these two predicates is `True`, there is a blocking pair.

– $\mathsf{BlockingPairW}(v) \equiv \mathsf{Married}(v) \wedge (\mathcal{C}_v \neq \emptyset)$
– $\mathsf{BlockingPairM}(v) \equiv (\exists\, u \in \mathcal{C}_v : u.marriage \neq \texttt{Null})$

The following predicate $\mathsf{AllCoherentPhase}(v)$ checks the coherence of phases, namely whether v and all its neighbors are in phase 2, or v is in phase 1 and all its neighbors in phases 1 or 1.5. It is used only by men to decide if they can accept a proposal (women verify somewhat different conditions).

– $\mathsf{AllCoherentPhase}(v) \equiv (v.phase = 2 \wedge (\forall\, u \in \mathcal{N}(v) : u.phase = 2))$
$\vee (v.phase = 1 \wedge (\forall\, u \in \mathcal{N}(v) : u.phase \in \{1, 1.5\}))$

3.1 Algorithm Description and Code

The matching $\mathcal{M}$ built by the presented algorithm is defined by pairs $(w, m) \in E$ such that $w.marriage = m$ and $m.marriage = w$. The predicate defining the stable matching problem is $[\forall w \in \text{WOMEN} : \mathsf{Married}(w) \wedge \neg\mathsf{BlockingPairW}(w) \wedge \neg\mathsf{BlockingPairM}(w.marriage)]$. We define the legitimate configurations as the terminal configurations satisfying this predicate.

The part of the algorithm executed by women (Algorithm 1) has 9 rules. We start by describing intuitively what those rules do.

1. The **Reset** rule, performs a reset of marriage and proposal pointers, if these pointers appeared to be incoherent according to the $\mathsf{IncoherentPointersW}$ predicate.
2. The rule **BadInit** is executed by a woman in phase 2. In this phase a married woman is not supposed to make a proposal. Thus, if her proposal and marriage pointers are not set to `Null` (the only reason for that is a bad initialization), **BadInit** resets the proposal pointer and sets the phase to 1 (to restart the computation of a matching).
3. The rule **Propose1** is executed by a married woman in phase 1. The rule effect is a proposal to the man who corresponds to the best marriage for her (*i.e.*, best for the woman but also for the man with respect to its actual spouse or single status).

4. The rule **Confime1** is executed by a woman in phase 1, after she has made a proposal to a man and this proposal has been accepted (the man has put the name of the woman in its variable proposal). Then the woman confirms the marriage, breaking from her previous man and matching with the new one. The couple is now considered married.
5. The rule **Propose2** is executed by women in phase 2, in order to make a proposal.
6. The rule **Confirm2** is the analogous of **Confim1** for a woman in phase 2.
7. The rule **ToPhase1.5** is a phase transition rule from phase 1 to phase 1.5. When a woman in phase 1 can not make any proposal (no blocking pair is detected or she is single), she has to move to phase 1.5 if all men are in phase 1.
8. The rule **ToPhase2** is also a phase transition rule. A woman in phase 1.5 can change to phase 2 if she does not detect any blocking pair and if all men are in phase 1.5.
9. The rule **ToPhase1** is a third phase transition rule. It is executed by a woman in order to move from phase 2 or phase 1.5 to phase 1. The change happens if the following (faulty) conditions are detected: (a) the woman is in phase 2 but some man is in phase 1 (either a blocking pair has been detected or phase synchronisation has not stabilized yet); (b) the woman is in phase 1.5 but a man is in phase 2 (the phase synchronization has not stabilized yet); (c) the woman is married and either in phase 1.5 or 2 but detects a blocking pair.

Remark 2. If a man does not answer positively to a proposal from a woman w (it has a better priority proposal), she detects it. $\mathsf{BestMarriage}(w)$ will not return any longer this man and w can change her proposal with **Propose1** or **Propose2**.

The part of the algorithm executed by men (Algorithm 2) consists of 6 rules:

1. The **Reset** rule resets the marriage pointer of a man and changes its phase to 1. We prove later that this can happen only once for a man in phase 2.
2. The **Accept** rule checks that women are in a consistent phase related to the phase of the man ($\mathsf{AllCoherentPhase}$), that the best proposal received is different from his actual marriage and that he has not accepted another proposal ($\neg\mathsf{AlreadyEngaged}$). Remark that this is a commitment, but the couple is not yet married. If the man is married with another woman, he has to break the marriage since he has a better proposal.
3. The role of the rule **Confirm** is to confirm a marriage. The rule checks that the phases are coherent and if the woman has her variable marriage set to the man, he confirms too.
4. The rule **ToPhase1.5** is a phase transition rule from phase 1 to phase 1.5. If all women are in phase 1.5 and no blocking pairs are detected, the man changes his phase to 1.5.
5. **ToPhase2** makes men change to phase 2. When all women are in phase 2 and men have checked the lack of BPs, then phase 2 can begin.

6. The **ToPhase1** rule detects a phase synchronization problem (a woman being in phase 1 or 1.5 with the man in phase 2) or a woman willing to change to phase 1 (blocking pair detected) when he is in phase 1.5.

Algorithm 1 for $w \in$ WOMEN

Reset : *(* Reset pointers of marriage and proposal *)*
{ IncoherentPointersW(w) }
$w.marriage \leftarrow$ `Null`, $w.proposal \leftarrow$ `Null`
BadInit : *(* Reset the pointer of proposal *)*
{ $\neg$IncoherentPointersW(w) $\wedge$ $w.marriage \neq$ `Null`
$\wedge$ $w.proposal \neq$ `Null` $\wedge$ $w.phase = 2$ }
$w.proposal \leftarrow$ `Null`, $w.phase \leftarrow 1$
Propose1: *(* Propose in phase 1 *)*
{ $\neg$IncoherentPointersW(w) $\wedge \forall\, v \in \mathcal{N}(w) \cup \{w\} : v.phase = 1$
$\wedge$ BestMarriage(w) $\neq w.proposal \wedge$ Married(w) }
$w.proposal \leftarrow$ BestMarriage(w)
Confirm1: *(* Confirm a proposal in phase 1 *)*
{ $\neg$IncoherentPointersW(w) $\wedge \forall\, v \in \mathcal{N}(w) \cup \{w\} : v.phase = 1$
$\wedge$ Response(w) $\wedge$ Married(w) $\wedge$ BestMarriage(w) $= w.proposal$ }
$w.marriage \leftarrow w.proposal, w.proposal \leftarrow$ `Null`
Propose2: *(* Propose in phase 2*)*
{ $\neg$IncoherentPointersW(w) $\wedge \forall\, v \in \mathcal{N}(w) \cup \{w\} : v.phase = 2$
$\wedge$ BestMarriage(w) $\neq w.proposal \wedge w.marriage =$ `Null` }
$w.proposal \leftarrow$ BestMarriage(w)
Confirm2: *(* Confirm a proposal in phase 2 *)*
{ $\neg$IncoherentPointersW(w) $\wedge \forall\, v \in \mathcal{N}(w) \cup \{w\} : v.phase = 2$
$\wedge$ Response(w) $\wedge$ $w.marriage =$ `Null`
$\wedge$ BestMarriage(w) $= w.proposal$ }
$w.marriage \leftarrow w.proposal, w.proposal \leftarrow$ `Null`
ToPhase1.5: *(* To the phase 1.5 *)*
{ $\neg$IncoherentPointersW(w) $\wedge \forall\, v \in \mathcal{N}(w) \cup \{w\} : v.phase = 1$
$\wedge$ $\neg$BlockingPairW(w) }
$w.phase \leftarrow 1.5, w.proposal \leftarrow$ `Null`
ToPhase2: *(* To the phase 2 *)*
{ $\neg$IncoherentPointersW(w) $\wedge \forall\, v \in \mathcal{N}(w) \cup \{w\} : v.phase = 1.5$
$\wedge$ $\neg$BlockingPairW(w) }
$w.phase \leftarrow 2, w.proposal \leftarrow$ `Null`
ToPhase1: *(* To the phase 1 *)*
{ $\neg$IncoherentPointersW(w) $\wedge$ (
[$\exists\, m \in \mathcal{N}(w) : (m.phase = 1 \wedge w.phase = 2)$
$\vee$ $(m.phase = 2 \wedge w.phase = 1.5)$]
$\vee$
[$(w.phase \in \{2, 1.5\} \wedge$ BlockingPairW(w))]) }
$w.phase \leftarrow 1, w.proposal \leftarrow$ `Null`

Algorithm 2 for $m \in$ Men

1: **Reset**: *(* Reset pointer of marriage *)*
2: { IncoherentPointersM(m) }
3: $m.marriage \leftarrow$ `Null`
4: $m.phase \leftarrow 1$
5: **Accept**: *(* Accept a proposal except in phase 1.5 *)*
6: { $\neg$IncoherentPointersM(m) $\wedge$ AllCoherentPhase(m)
7: $\wedge$ BestProposal(m) $\neq$ `Null` $\wedge$ $\neg$AlreadyEngaged(m) }
8: $m.proposal \leftarrow$ BestProposal(m)
9: **Confirm**: *(* Confirm a marriage *)*
10: { $\neg$IncoherentPointersM(m) $\wedge$ $m.proposal.marriage = m$
11: $\wedge$ AllCoherentPhase(m) }
12: $m.marriage \leftarrow m.proposal, m.proposal \leftarrow$ `Null`
13: **ToPhase1.5**: *(* To the phase 1.5 *)*
14: { $\neg$IncoherentPointersM(m) $\wedge$ $\forall\, w \in \mathcal{N}(m) : w.phase = 1.5$
15: $\wedge$ $m.phase = 1 \wedge \neg$BlockingPairM(m) $\wedge \neg$AlreadyEngaged(m) }
16: $m.phase \leftarrow 1.5, m.proposal \leftarrow$ `Null`
17: **ToPhase2**: *(* To the phase 2 *)*
18: { $\neg$IncoherentPointersM(m) $\wedge$ $\forall\, w \in \mathcal{N}(m) : w.phase = 2$
19: $\wedge$ $m.phase = 1.5 \wedge \neg$BlockingPairM($m$) }
20: $m.phase \leftarrow 2, m.proposal \leftarrow$ `Null`
21: **ToPhase1**: *(* To the phase 1 *)*
22: { $\neg$IncoherentPointersM(m) $\wedge$ (
23: $[(\exists\, w \in \mathcal{N}(m) : w.phase \in \{1.5, 1\}) \wedge m.phase = 2]$
24: $\vee$
25: $[(\exists\, w \in \mathcal{N}(m) : w.phase = 1) \wedge m.phase = 1.5])$ }
26: $m.phase \leftarrow 1, m.proposal \leftarrow$ `Null`

4 Proof of Correctness and Time Complexity

The analysis of the algorithm appears to be complex and long due to several reasons. First, the algorithm has to overcome the unfair adversary that can prevent some enabled nodes from being activated as long as there are other enabled nodes. This may take many moves made by nodes in different states and configurations. Moreover, all these moves may not contribute to the convergence (*e.g.*, if an existing fault is not yet detected). Still, they have to be taken into account for the correctness and the time analysis. Another reason for the analysis difficulty is the distribution and asynchrony of the solution. For example, as reciprocal marriages, divorces, and blocking pair detection cannot be done instantaneously, or at least within some timing guaranties (as in synchronous lock-step models), the related results on previous centralized or synchronous solutions cannot be used in our case. Finally, due to self-stabilization, the analysis has to consider executions starting from an arbitrary configuration.

In particular, initially, the phase numbers can be arbitrary. Moreover there are specific rules applying to such or such phase number. The consequence of that is a great number of cases to treat, each case necessitating a particular treatment

and special arguments. For classifying the different cases into categories, the following definition is introduced.

Definition 2. *Let A and B be two sets of phase numbers and bp a non-negative integer. We say that a configuration C is in the set of configurations denoted by $(A, B, bp)^{\times}$ if in C: (a) $\forall m \in$ Men $: m.phase \in A$, (b) $\forall w \in$ Women $: w.phase \in B$ and (c) bp is the number of blocking pairs.*

Furthermore, a configuration C is in the set denoted by (A, B, bp) , if it is in $(A, B, bp)^{\times}$ and satisfies $\bigcup_{m \in \text{Men}} \{m.phase\} = A \wedge \bigcup_{w \in \text{Women}} \{w.phase\} = B$.

For example: $(\{a\}, \{b, c\}, X)^{\times} \equiv (\{a\}, \{b, c\}, X) \bigcup (\{a\}, \{b\}, X) \bigcup (\{a\}, \{c\}, X)$. Furthermore, we denote by $\mathcal{C}^1$ the set of configurations where $\exists\ v \in V : v.phase = 1$.

So, we prove the algorithm for every possible starting configuration type. Due to the lack of space, only the main statements and ideas of the proof are presented in the following. The complete proof appears in the appendix.

First we consider a relatively simple case - the one of a terminal configuration. We show (Proposition 1) that such a configuration is in $(\{2\}, \{2\}, 0)$ and whenever it is reached the *marriage*-values define a stable marriage. Notice that this implies the closure part of the correctness proof.

Proposition 1. *In a terminal configuration, the set of edges $\{(w, m) \in E : w.marriage = m \wedge m.marriage = w\}$ is a stable matching. This configuration is in $(\{2\}, \{2\}, 0)$.*

Then, we prove the convergence part of the proof by showing convergence to a terminal configuration. First, we show step by step, through Lemmas 7 - 12, that from any configuration in $\mathcal{C}^1$, in $O(n^4)$ moves, an execution reaches a configuration in $(\{1.5\}, \{1.5\}, 0)$, having no blocking pairs. It is proven in particular by showing that the sum of the regrets of married women is strictly decreasing. Notice that we cannot conclude this property directly from a similar result for the centralized two-phased algorithm of Ackermann *et. al*, because it assumes "best response" dynamics, which we do not realize here (in phase 1). As already explained before, since marriages, divorces and detection of blocking pairs cannot be done instantaneously under a distributed setting, it is difficult and costly to realize such dynamics.

Then, through Lemmas 13 - 22 and Proposition 3 below, it is proven that from any configuration in $(\{1.5, 2\}, \{1.5, 2\}, X \geq 0)^{\times}$, in $O(n^4)$ moves, either the execution reaches (possibly cycles to) a configuration in $\mathcal{C}^1$, or reaches a configuration in $(\{2\}, \{2\}, 0)$. By Proposition 2 stated below, there is at most one such possible execution cycle, *i.e.*, any execution converges to a configuration in $(\{2\}, \{2\}, 0)$ in $O(n^4)$ moves.

Proposition 2. *Let C be a configuration in $(\{1.5, 2\}, \{1.5, 2\}, X)^{\times}$ with $X \geq 0$ and $C' \in \mathcal{C}^1$. In any execution, $C \to C'$ appears at most once.*

Proposition 3. *Any execution takes $O(n^4)$ moves to reach a configuration in $(\{2\}, \{2\}, 0)$.*

Proposition 4 below ensures that the conditions of a configuration in $(\{2\}, \{2\}, 0)$ required by Corollary 1 are satisfied in $O(n^4)$ moves. In particular, these conditions ensure that no node changes to phase 1 anymore (see **Reset** and **BadInit** rules). This in turn allows to obtain and consider the *last* segment of execution of phase 2, *i.e*, the last segment where all configurations are in $(\{2\}, \{2\}, 0)$. Then, by Corollary 1, from such configurations, a terminal configuration is obtained in $O(n^2)$ moves (this is proven through Lemmas 23 - 30 and Proposition 5). Notice that, when phase 2 is executed the last time, it is ensured by the algorithm that no blocking pairs exist or appear. However, the existing matching may be incomplete (unstable) and new matches continue to appear until termination.

Proposition 4. *Any execution starting in $(\{2\}, \{2\}, 0)$ takes $O(n^4)$ moves to reach a configuration in $(\{2\}, \{2\}, 0)$ such that*

1. *no man is enabled for the **Reset** rule,*
2. *no woman w is enabled for the **BadInit** rule and either $w.proposal =$ `Null` or $w.proposal =$ BestMarriage(w).*

Corollary 1. *Let $\mathcal{E}$ be an execution starting from a configuration in $(\{2\}, \{2\}, 0)$ such that*

1. *no man is enabled for the **Reset** rule,*
2. *no woman w is enabled for the **BadInit** rule and either $w.proposal =$ `Null` or $w.proposal =$ BestMarriage(w).*

$\mathcal{E}$ contains $O(n^2)$ moves.

Finally, Proposition 1 is used to prove a convergence to a stable marriage from a terminal configuration (reached by Proposition 4 and Corollary 1). Altogether this implies the main theorem below.

Theorem 1. *Any execution takes $O(n^4)$ moves to reach a terminal configuration where the set of edges $\{(w, m) \in E : w.marriage = m \wedge m.marriage = w\}$ is a stable matching.*

Proof. By Proposition 3, any execution takes $O(n^4)$ moves to reach a configuration C' in $(\{2\}, \{2\}, 0)$. By Proposition 4, starting from C', a configuration C'' in $(\{2\}, \{2\}, 0)$ satisfying the conditions of Corollary 1 is reached in $O(n^4)$ moves. Then, by Corollary 1, from C'' , a terminal configuration is reached in $O(n^2)$ moves. By Proposition 1, this configuration is legitimate (satisfying a stable matching). This implies the theorem. □

References

AGL10. N. Amira, R. Giladi, and Z. Lotker. Distributed weighted stable marriage problem. In *SIROCCO 2010*, pages 29–40, 2010.

AGM+11. H. Ackermann, P. W. Goldberg, V. S. Mirrokni, H. Röglin, and B. Vöcking. Uncoordinated two-sided matching markets. *SIAM J. Comput.*, 40(1):92–106, 2011.

AKM+07. B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese. A time-optimal self-stabilizing synchronizer using A phase clock. *IEEE Trans. Dependable Sec. Comput.*, 4(3):180–190, 2007.

BM05. I. Brito and P. Meseguer. Distributed stable marriage problem. In *6th Workshop on Distributed Constraint Reasoning at IJCAI*, volume 5, pages 135–147, 2005.

BPV04. C. Boulinier, F. Petit, and V. Villain. When graph theory helps self-stabilization. In *PODC*, pages 150–159, 2004.

CGMP99. S. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, 1999.

Dij74. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, November 1974.

FKPS10. P. Floren, P. Kaski, V. Polishchuk, and J. Suomela. Almost stable matchings by truncating the gale-shapley algorithm. *Algorithmica*, 58(1):102–118, 2010.

Gho14. S. Ghosh. *Distributed Systems: An Algorithmic Approach, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

GI89. D. Gusfield and R. W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.

Gol06. P. Golle. A private stable matching algorithm. In *FC*, pages 65–80, 2006.

GS62. D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 120(5):386–391, 1962.

KL14. G. Kim and W. Lee. Stable matching with ties for cloud-assisted smart tv services. In *ICCE*, pages 558–559, Jan 2014.

Knu76. D. E. Knuth. *Mariages stables et leurs relations avec d'autres problemes combinatoires*. Les Presses de l'Universite de Montreal, 1976.

KPS09. A. Kipnis and B. Patt-Shamir. A note on distributed stable matching. In *ICDCS*, pages 466–473, June 2009.

MS15. B. M. Maggs and R. K. Sitaraman. Algorithmic nuggets in content delivery. *Computer Communication Review*, 45(3):52–66, 2015.

OR15. R. Ostrovsky and W. Rosenbaum. Fast distributed almost stable matchings. PODC, pages 101–108. ACM, 2015.

PW16. P.Khanchandani and R. Wattenhofer. Distributed stable matching with similar preference lists. In *OPODIS*, pages 12:1–12:16, 2016.

RS90. A. E. Roth and M. A. O. Sotomayor. *Two-sided Matching: A study in game-theoretic modeling and analysis*. Cambridge University Press, 1990.

RVV90. A. Roth and J. H. Vande Vate. Random paths to stability in two-sided matching. *Econometrica*, 58(6):1475–80, 1990.

XL11. H. Xu and B. Li. Seen as stable marriages. In *INFOCOM*, pages 586–590, 2011.

Appendix

Closure Proof

Lemma 1. *Let C be a terminal configuration. For any node v, the predicates* IncoherentPointersM(v) *and* IncoherentPointersW(v) *are* `False`.

Proof. If there exist some nodes for which these predicates are `True`, they are eligible for the **Reset** rule, which gives a contradiction with the fact that the configuration is terminal. □

Lemma 2. *Let C be a terminal configuration. Let $v \in$ Men.*
Then predicate AllCoherentPhase(v) *is* `True` *in C.*

Proof. Assume that AllCoherentPhase(v) is `False`, by contradiction. There exists some $u \in$ Women such that:

1. if $v.phase = 2$ then $u.phase \in \{1, 1.5\}$
2. if $v.phase = 1$ then $u.phase = 2$

Predicate IncoherentPointersW(u) is `False` by Lemma 1. If $u.phase \in \{2, 1.5\}$ then u is eligible for the **ToPhase1** rule since $v \in \mathcal{N}(u)$ and because of points 1 and 2. If $u.phase = 1$ then v is eligible for **ToPhase1**. This contradicts the fact that the configuration is terminal. □

Lemma 3. *Let C be a terminal configuration and v be a node. If there exists a node $u \in \mathcal{N}(v)$ such that $v.marriage = u$ then $u.marriage = v$.*

Proof. Assume first that $v \in$ Men and that $v.marriage = u \in$ Women. If $u.marriage \neq v$ then predicate IncoherentPointersM(v) is `True`, which is not possible in a terminal configuration, by Lemma 1.

Assume now that $v \in$ Women, that $v.marriage = u$ with $u \in$ Men and that $u.marriage \neq v$. Necessarily $u.proposition = v$ or IncoherentPointersW(v) is `True`, which is not possible by Lemma 1. We also have that the predicate IncoherentPointersM(u) is `False` by Lemma 1. This implies that AllCoherentPhase(u) is `False` or u eligible for **Confirm**. This gives a contradiction, by Lemma 2. □

Lemma 4. *Let C be a terminal configuration. No node v is in phase 1 in C.*

Proof. Assume by contradiction that there exists some node v in phase 1 in configuration C.

We check first the case in which $v \in$ Women. Let $u \in$ Men. We have that predicate IncoherentPointersM(u) is `False` for u and v by Lemma 1. Observe now that u is in phase 1 or it is eligible for the **ToPhase1** rule. Thus we can assume that all men are in phase 1 as well.

There are two different cases for woman v:

- she is married and has a blocking pair with some node $u_1 \in$ Men. Assume without loss of generality that u_1 corresponds to BestMarriage(v). Necessarily $v.proposition = u_1$ or v eligible for **Propose1**.
  There are two cases for man u_1:

1. BestProposition(u_1)= v
2. BestProposition(u_1)= v_1 with $v_1 \in$ WOMEN and $v_1 \neq v$.

We first check case 1. If $u_1.proposition = v$ then v is eligible for the **Confirme1** rule since predicate Response(v) is `True` and IncoherentPointersW(v) is `False`, by Lemma 1. This yields a contradiction.
If $u_1.proposition \neq v$ we show that u_1 is eligible for the **Accept** rule. First we have that AllCoherentPhase(u_1) is `True` by Lemma 2 and that BestProposition(v)$\neq$ *null*. If predicate AlreadyEngaged(u_1) is `False` then u_1 is eligible for **Accept**. Thus assume that AlreadyEngaged(u_1) is `True`, by contradiction. Let $v_2 = u_1.proposition$. By definition of the predicate $v_2 \neq null$. Now $u_1.marriage \neq v_2$ or the predicate IncoherentPointersM(u_1) is `True`, which is not possible by Lemma 1. There are two cases. If $v_2.marriage = u_1$ then u_1 is eligible for **Confirm**. Thus assume that $v_2.marriage \neq u_1$. Observe first that v_2 is either in phase 1 or 1.5 or predicate AllCoherentPhase(v_2) is `False` since u_1 is in phase 1, which is not possible by Lemma 2. Assume first that v_2 is in phase 1. If it is married with some node then it is eligible for the **Confirm** rule. Thus assume that it is not married. Then by definition, its predicate BlockingPairW(v_2) is `False`. In that case it is eligible for the **ToPhase1.5** rule. If it is in phase 1.5, if its predicate BlockingPairW(v_2) is `False` then it is eligible for **ToPhase2**, if it is `True`, it is eligible for **ToPhase1**, which yields the contradiction.
We now check case 2. If $u_1.proposition \neq v_1$ then, using exactly the same arguments as in the previous case, u_1 is eligible for the **Accept** rule. If $u_1.proposition = v_1$ then $v_1.phase = 1$ by definition of predicate BestProposition(u_1). Thus if Married(v_1) is `True` necessarily BestMarriage(v_1)= u_1 or v_1 eligible for **Propose1**. We also have that $v_1.marriage \neq u_1$ or BestMarriage(v_1) would not be equal to u_1. Observe now that v_1 is eligible for **Confirm1** which gives a contradiction. If Married(v_1) is `False`, then BlockingPairW(v_1) is `False` and thus, since $v_1.phase = 1$ and all MEN are in phase 1, then v_1 is eligible for the **ToPhase1.5** rule which gives a contradiction.

– she is single or married with no blocking pair : the **ToPhase1.5** rule can be applied.

Thus in a terminal configuration, women are not in phase 1. Assume that $v \in$ MEN with $v.phase = 1$. Women can either be in phase 1.5 or 2 (since women are not in phase 1). If some woman is in phase 2, she is eligible for the **ToPhase1** rule since $v.phase = 1$. Thus we can assume that all women are in phase 1.5. If predicate BlockingPairM(v) is `False` then v is eligible for the **ToPhase1.5** rule. Thus we can assume that BlockingPairM(v) is `True`. Let u be the woman which forms a blocking pair with v. Then the predicate BlockingPairW(u) is `True`. This implies that u is eligible for the **ToPhase1** rule since $u.phase = 1.5$. □

Lemma 5. *Let C be a terminal configuration. C is in* $(\{2\}, \{2\}, 0)$.

Proof. We prove this lemma by contradiction. Let C be a terminal configuration not in $(\{2\}, \{2\}, 0)$. We thus have that C is in the configuration set:

1. $(\{1.5\}, \{1.5\}, X)$ or
2. $(\{1.5, 2\}, \{1.5\}, X)$ or
3. $(\{1.5, 2\}, \{2\}, X)$ or
4. $(\{2\}, \{1.5, 2\}, X)$ or
5. $(\{1.5\}, \{1.5, 2\}, X)$ or
6. $(\{1.5, 2\}, \{1.5, 2\}, X)$ or
7. $(\{1.5\}, \{2\}, X)$ or
8. $(\{2\}, \{1.5\}, X)$ or
9. $(\{2\}, \{2\}, X)$

For point 1, all nodes are in phase 1.5. If $X = 0$ then women can apply the **ToPhase2** rule. If $X \neq 0$ then there exists some woman who can apply the **ToPhase1** rule.

For point 2, all women are in phase 1.5, men are in phase 1.5 or 2 (with at least one in each phase). Women in phase 1.5 are eligible for the **ToPhase1** rule. The same holds for points 6 and 8.

For point 3, all women are in phase 2, men are in phase 1.5 or 2. If there is a blocking pair, women are eligible for the **ToPhase1** rule. If there are no blocking pairs, men in phase 1.5 are eligible for the **ToPhase2** rule.

For point 4, all men are in phase 2 and women are in phase 1.5 or 2. If there is a blocking pair, women are eligible for the **ToPhase1** rule. Otherwise, women in phase 1.5 are eligible for the **ToPhase2** rule.

For point 5, all men are in phase 1.5 and women are in phase 1.5 or 2. If there is a blocking pair, women are eligible for the **ToPhase1** rule. Otherwise, men in phase 1.5 are eligible for the **ToPhase2** rule.

For point 7, if there is a blocking pair, women are eligible for **ToPhase**. Otherwise men are eligible for **ToPhase2**.

Finally consider configurations in $(\{2\}, \{2\}, X)$. Women which have a blocking pair are eligible for the **ToPhase1** rule, which concludes the overall proof. □

Lemma 6. *In a terminal configuration, every woman is married.*

Proof. Let C be a terminal configuration. By Lemma 5, configuration C is in $(\{2\}, \{2\}, 0)$. Assume by contradiction that there exists $v \in$ WOMAN which is not married in C. By definition, she is in phase 2. Since she is not married there is at least a man which is not married as well, since the graph is bipartite complete with the same number of men and women. Thus there are two cases:

- $v.marriage =$ `Null` or
- $\exists m \in$ MEN such that $v.marriage = m$ and $m.marriage \neq v$

In the first case, necessarily $v.proposition =$ `Null` or v eligible for **Reset**. Observe also that $\mathsf{BestMarriage}(v) \neq \emptyset$ since there is at least a man which is not married. Thus, in that case, woman v is eligible for **Propose2** which yields the contradiction.

In the second case, necessarily $m.marriage = v$ by Lemma 3, which yields the contradiction. □

Proposition 1. *In a terminal configuration, the set of edges* $\{(w,m) \in E : w.marriage = m \wedge m.marriage = w\}$ *is a stable matching. This configuration is in* $(\{2\},\{2\},0)$.

Proof. The fact that a terminal configuration is in $(\{2\},\{2\},0)$ holds by Lemma 5. Since all women are married by Lemma 6 and that such a marriage is stable by Lemma 3 then the proposition holds. □

Convergence to ({2}, {2}, 0)

Lemma 7. *Let C a configuration in* $\mathcal{C}^1$. *Any execution starting from a configuration C takes* $O(n^2)$ *moves to reach a configuration C' in* $(\{1\},\{1,1.5\},X)^\times \bigcup(\{1.5\},\{1.5\},0)$.

Proof. Let v be a node in phase 1. Firstly, consider the case of $v \in$ Men. Other men may be in any phase. Let w be in Women. w can be eligible for the **Reset** rule if her pointers are incoherent. For other rules, we consider the different sub-cases:

1. $w.phase = 2$: w is only eligible for the **ToPhase1** (one of her neighbors is in phase 1), **BadInit** and **Reset** rule. The first two rules set the phase of w to 1 after a **Reset** if necessary.
2. $w.phase = 1.5$: if the woman is involved in a BP, she is eligible for the **ToPhase1** rule otherwise she is eligible for the **Reset** rule (and not for both because if she is eligible for the **Reset**, she is not married and then has no blocking pair).
3. $w.phase = 1$: if all men are not in phase 1, she has no eligible rule except the **Reset**. Otherwise w is eligible for:
   - **Reset** rule only once. Indeed, after a **Reset** she is single and then cannot be married in this phase.
   - **ToPhase1.5** rule once if she is single or married without a blocking pair. After a man's **Reset**, w may detect a blocking pair with this man. She is then eligible for the **ToPhase1.5** rule. But since men cannot accept proposal (Women are not all in phase 1 or 1.5, otherwise the configuration is already *C'*), the blocking pair is still there and she is not eligible again for the **ToPhase1.5** rule.
   - **Propose1**: since w may propose only once to each man (and not to her spouse), w is eligible for this rule at most $n-1$ times. In fact, if w propose to a man m, the predicate BestMarriage selects the best possible spouse. But if pointers of men are incoherent, w cannot detect the blocking pair with a better spouse m_1 and propose to m. After the activation of m_1 for the **Reset**, she can propose. And this case may happen $n-1$ times.
   - **Confirm1**: it is a special case of the previous case. Indeed, since women are not all in phase 1 or 1.5, men cannot accept a proposal. But in the configuration C, the proposal pointer of a man m can be already set to w. Then, if w propose to this man, she can also be eligible for the **Confirm1** rule. Then, w has resolved a blocking pair (because in the

definition of $\mathcal{C}_v$, w check if her proposal is also more interesting for m). m will be eligible for the **Confim** rule when all women will be in phase 1 or 1.5, that is the configuration C'.

The worst case is when women are all in phase 1 except one in phase 2 and men all in phase 1: they can propose to men at most $n-1$ times. Indeed, each woman is eligible for $O(n)$ moves. That is altogether, $O(n^2)$ moves of women after that, all women are either in phase 1 or 1.5.

Now, let us consider the case of $v \in$ Women. Let us analyze the other nodes next moves. Let $m \in$ Men. In all the case, he is eligible for the **Reset** and then his phase is set to 1. Otherwise, if:

1. $m.phase = 1$, m has nothing to do. In fact, if m is eligible for a **Accept** or **Confirm** rule, that means that all men are in phase 1 because otherwise, women cannot propose or confirm a marriage. If all women are in phase 1 and there is an incoherent pointer, a woman possibly has her pointer of proposal to m and m is eligible for an **Accept** rule but the woman won't answer while all men are not in phase 1 (and then, the configuration is C'). Furthermore, if all pointers of a woman are incoherent (the two pointers are set to m for example) m cannot be eligible for these two rules because of the definition of $\mathcal{P}_v$. A man is eligible at most once for one of these rules.
2. $m.phase = 2$ or $m.phase = 1.5$, m is eligible for **ToPhase1** rule if he was not eligible for the **Reset**: one of his neighbor is in phase 1. If a woman in phase 2 proposes to m, he cannot accept (Predicate AllCoherentPhase).

Then, a man is eligible for at most two rules. That is altogether $O(n)$ moves, men are all in phase 1.

In short, in $O(n^2)$ moves, the system reaches C'. □

Let MarriedWomen(C) be the set of nodes v in Women that are married in the configuration C. Let $\mathcal{R}_w(C)$ be the sum of the regret of nodes v in MarriedWomen(C):

$$\mathcal{R}_w(C) = \sum_{v \in \text{MarriedWomen}(C)} w(v, v.marriage(C))$$

Lemma 8. *Let C be a configuration in $(\{1\},\{1,1.5\},X)^\times$ with $X > 0$. Any execution starting from C takes $O(n^2)$ moves to reach a configuration C' in $(\{1\},\{1,1.5\},Y)$ such that $\mathcal{R}_w(C) > \mathcal{R}_w(C')$.*

Proof. Let us consider all possible moves in C that do not change $\mathcal{R}_w(C)$ and count how many times each rule is eligible for each node. Note that **Confirm1** rule is the only rule that change $\mathcal{R}_w(C)$. Indeed, a woman w_0 is married if $[(w_0.marriage.marriage = w_0) \vee (w_0.marriage.proposal = w_0)]$ and **Confirm1** set her marriage pointer to another man m_0 if his proposal pointer is pointing on w_0 (Predicate Response(w_0)). Furthermore, the predicate BestMarriage(w_0) = $w_0.proposal$ checks if m_0 is the best man for w_0 in the current configuration.

Then let us consider all other possibles moves. Firstly, let m be in Men. m is eligible for only 5 rules depending on his state:

- **Reset** rule. m may be eligible once for this rule. Indeed, if m is eligible a second time, that means his pointers are incoherent. But after the first **Reset**, this is not possible: that means he was married and his woman found a better spouse. In this case, she has resolved a blocking pair and has been activated for the **Confirm1** rule.
- **ToPhase1.5** rule if all women are in phase 1.5, $\mathsf{BlockingPairM}(m) = \texttt{False}$ and he his not engaged.
- **ToPhase1** rule. Since men are eligible for the **ToPhase1.5** and there are X BP, at least one man (involved in a blocking pair $(m_1,\ w_1)$) will stay in phase 1. w_1 will be activated for the **ToPhase1** (Predicate $\mathsf{BlockingPairW}(w)$ is $\texttt{True}$). After this move, if w is in 1.5 he is eligible for the **ToPhase1** rule.
- **Accept** rule if a woman is proposing to m and that her proposal is the best proposal for m in C. Since $\mathcal{P}_v$ is defined respecting the preference of the proposing woman and m, m accept only if the marriage is beneficial for both of them. But in C_0 (such tat $C \xrightarrow{*} C_0$), an other better ranked woman may propose to m. Then, m is eligible $O(n)$ times.
- **Confirm** rule. When m confirm, he is already considered married (after the **Confirm1** rule of the woman). But he is not eligible twice, because it would mean he has a new marriage (and then a woman has been activated for a **Confirm1** rule).

Altogether, a man is eligible for at most $O(n)$ moves, that is $O(n^2)$ moves for all men.

Now, let w be in WOMEN. w is eligible for 4 rules:

- **Reset** rule. If she is eligible for a **Reset** rule, that means she is not married. She cannot be involved in a blocking pair and she is at most eligible for an other move: **ToPhase1.5** rule.
- **ToPhase1.5** rule if the single woman or not involved in a blocking pair in phase 1. Only once, because otherwise, that means she is gone back to 1 (for a blocking pair, see the next point) and if she is again eligible **ToPhase1.5**, that means there are no more blocking pairs.
- **ToPhase1** rule if a woman involved in a blocking pair is in phase 1.5 (only once because of the previous point).
- **Propose1** rule if there is at least a blocking pair involving w: she proposes to the best ranked man in $\mathcal{C}_v$. Since w can be involved in at most $n-1$ BP, she can propose at most $n-1$ times. (If pointers of men may be incoherent, a woman cannot immediately know all blocking pairs. Then, she can propose to a first man before to see an other BP).

In overall, a woman is eligible for at most $O(n)$ moves, that is $O(n^2)$ moves for all women.

To summarize, nodes are eligible for at most $O(n^2)$ moves before that at least one woman is eligible for the **Confirm1** rule. Men are in phase 1 and women are either in phase 1 and 1.5. Thus, C' is reached. Note that the number of BPs is now Y: a blocking pair (m, w) has been resolved but the previous spouse of w is now single. New blocking pairs may appear after the resolution of BP. □

Lemma 9. *Let C and C' be configurations such that $C \xrightarrow{*} C'$. C and C' are in $(\{1\}, \{1, 1.5\}, X)^{\times} \bigcup (\{1, 1.5\}, \{1.5\}, X)^{\times}$. Let w be a woman. If $w.marriage(C) \neq w.marriage(C')$ then $w.marriage(C')$ has a better priority in the list of w than $w.marriage(C)$ or $w.marriage(C') = \texttt{Null}$. Thereby, $\mathcal{R}_w(C) > \mathcal{R}_w(C')$. Furthermore, w cannot be married again with $w.marriage(C)$ before she is activated with a* **ToPhase2** *rule.*

Proof. Let us consider cases. If $w.marriage(C) \neq w.marriage(C')$, there are two cases in phase 1:

1. $w.marriage(C) = m$ and $w.marriage(C') = \texttt{Null}$
2. $w.marriage(C) = m$ and $w.marriage(C') = m_1$

(Since single women in phase 1 or 1.5 cannot been eligible for a **Propose1** rule, the case $w.marriage(C) = \texttt{Null}$ and $w.marriage(C') = m$ is not possible. ($\mathsf{Married}(w)$ in C is $\texttt{False}$))

The first case happen if the man m married with the woman w receives a proposal of a woman better ranked than his spouse. Then m accepts the proposal and w becomes single. We have $\mathcal{R}_w(C) > \mathcal{R}_w(C')$ because w is now single and does not count no longer. w is now eligible for only one rule: the **ToPhase1.5** (if $w.phase = 1$) or **ToPhase2** (if $w.phase = 1.5$) when all nodes will be in phase 1.5.

The second case happen if w makes a proposal and confirmation to m_1. To be eligible to propose in phase 1, w belongs to a blocking pair (w, m_1). Then, it means that m_1 is better ranked by w than m and $\mathcal{R}_w(C) > \mathcal{R}_w(C')$. If w may be married again with m, it means there is a blocking pair (w, m). But m is worse ranked than m_1: this is not possible.

Then, if there is no more blocking pair involving w, she is only eligible for the **ToPhase1.5** or **ToPhase2** rule if all men are in phase 1.5. □

Lemma 10. *Let C be in $(\{1\}, \{1, 1.5\}, X)^{\times}$. Any execution starting from C takes $O(n^4)$ moves to reach a configuration C' in $(\{1\}, \{1, 1.5\}, 0) \bigcup (\{1\}, \{1.5\}, 0)$.*

Proof. Let us consider the special case where $X = 0$. The sub-case where $C \in (\{1\}, \{1.5\}, 0) \bigcup (\{1\}, \{1, 1.5\}, 0)$ is trivial: the configuration is already C'. Other configurations are in $(\{1\}, \{1\}, 0)$. In these configurations, women are only eligible for at most two rules: **Reset** and **ToPhase1.5** for women in phase 1. Men are only eligible for **Reset**. Then, if $X = 0$, after at most $2n$ **Reset** (men and women) and 1 **ToPhase1.5** (women), that is $O(n)$ moves, the configuration C' is reached. Otherwise, the configuration is still C with $X > 0$.

Now, assume that $X > 0$. Let us determine an upper bound on X. Since there is a blocking pair (w, m) only if w is married, w is involved in at most $n - 1$ blocking pairs. Then, if each women is involved in $n - 1$ blocking pairs, there are $O(n^2)$ blocking pairs.

By Lemma 8, one blocking pair is resolved in $O(n^2)$ moves. Since each blocking pair can be resolved at most once (Lemma 9), there is no more blocking pair after $O(n^4)$ moves. When a woman was activated to confirm (to resolve the last

blocking pair), all men were in phase 1 and at least one woman is in phase 1 and others in phase 1 or 1.5, that is in C' (if there is no woman in phase 1.5, after one **ToPhase1.5**, $(\{1\}, \{1, 1.5\}, 0)$ is reached). □

Lemma 11. *Any execution starting from a configuration C in $(\{1\}, \{1, 1.5\}, 0)$ takes $O(n)$ moves to reach a configuration $C' \in (\{1\}, \{1.5\}, 0)$.*

Proof. Consider the eligible rules in configuration C. Since there is no more blocking pairs, no woman is eligible for **Propose1** or even **Confirm1**. Furthermore, men are also not eligible for **Accept** or **Confirm**. Even if there are woman's incoherent pointers, men cannot accept because of the definition of $\mathcal{P}_v$. Indeed, it checks if the proposition is more interesting for both, the man and the woman. If it is the case, it means that there exists still a blocking pair. This yields to a contradiction. Concerning **ToPhase1**, nodes cannot be eligible, because there is no more BPs or they are already in phase 1. The **ToPhase2** and **BadInit** rules are not eligible because of nodes' phases. Men are not eligible for the **ToPhase1.5** rule owing to women in phase 1. These women are eligible for the **ToPhase1.5** rule. Finally, nodes are also eligible for the **Reset** rule. Thus, after at most $2n$ **Reset** and at most $n - 1$ **ToPhase1.5**, that is $O(n)$ moves, the system reaches a configuration C' in $(\{1\}, \{1.5\}, 0)$. □

Lemma 12. *Any execution starting from a configuration C in $(\{1\}, \{1.5\}, 0)$ takes $O(n)$ moves to reach a configuration $C' \in (\{1.5\}, \{1.5\}, 0)$.*

Proof. Let us consider first women. They have no eligible rule except **Reset** because of their phases and the phases of men.

Now, let us consider men. As women are in phase 1.5 and cannot change their phase, men are only eligible for the **ToPhase1.5** rule. Furthermore, they are eligible for **Reset**. Note that each man is eligible for the **Reset** rule before the **ToPhase1.5** thanks to the predicate IncoherentPointersM.

Then, after $2n$ **Reset** and n **ToPhase1.5**, that is $O(n)$ moves, the configuration C' is reached. □

Lemma 13. *In a configuration C in $(\{1.5\}, \{1.5\}, 0) \bigcup (\{1.5\}, \{1.5, 2\}, 0)$, women are enabled for rules $\in \{$**ToPhase2**, **BadInit**, **Reset**$\}$ and men are only enabled for the* **Reset** *rule. Furthermore if $C \to C'$, then the configuration C' is:*

- *in $(\{1.5\}, \{1.5, 2\}, 0) \bigcup (\{1.5\}, \{2\}, 0)$ if in C, only women's* **ToPhase2** *or* **Reset** *are activated.*
- *in $\mathcal{C}^1$ if in C, at least one men's* **Reset** *or women's* **BadInit** *is activated.*

Proof. Let v be an eligible node in Men. By definition of C, $v.phase = 1.5$. Then, **Accept**, **Confirm** and **ToPhase1.5** rules cannot be applied. Since there is no woman in phase 1 and v is in phase 1.5, v cannot be eligible for the **ToPhase1** rule. Furthermore, there exists at least one woman in phase 1.5, the **ToPhase2** rule is also not an eligible rule. If the pointer of v is incoherent, v is eligible only for the **Reset** rule.

Now, let v be in WOMEN. Because men's phase is 1.5, women cannot apply the **Propose1**, **Confirm1**, **Propose2**, **Confirm2** and **ToPhase1.5**. The **ToPhase1** rule is also not eligible: there is no blocking pair and the phases of men are not 1. Thus, the only possible rules for v are **ToPhase2** (if v is in phase 1.5), **Reset** (if the pointer is incoherent) and **BadInit** (if v is in phase 2).

If in the transition $C \to C'$ at least one man (the **Reset** rule) or a woman (the **BadInit** rule in phase 2) is activated, the configuration C' is in the set of configurations where at least one node is in phase 1. Otherwise, if only women are activated for **ToPhase2** or **Reset** rules, C' is in $(\{1.5\}, \{1.5, 2\}, 0) \bigcup (\{1.5\}, \{2\}, 0)$. □

Lemma 14. *In a configuration C in $(\{1.5\}, \{2\}, 0) \bigcup \{1.5, 2\}, \{2\}, 0)$, women are only enabled for the* **Reset** *and* **BadInit** *rules and men are enabled for rules $\in$ {***ToPhase2***,* **Reset***,* **Accept***,* **Confirm***}. Furthermore if $C \to C'$, then the configuration C' is in:*

- *in $(\{1.5, 2\}, \{2\}, 0) \bigcup (\{2\}, \{2\}, 0)$ if in C, only women's* **Reset** *or men's* **ToPhase2***,* **Accept** *and* **Confirm** *rules are activated.*
- *in $\mathcal{C}^1$ if in C, at least one men's* **Reset** *or women's* **BadInit** *rule is activated.*

Proof. Let v be an eligible node in WOMEN. By definition of C, there exists at least one man in phase 1.5. Then, rules **Propose1**, **Propose2**, **Confirm1** and **Confirm2** cannot be applied. Since women are already in phase 2 and there is no blocking pair, **ToPhase2**, **ToPhase1.5** and **ToPhase1** rules are also not eligible. Then, if the pointer of v is incoherent, v is eligible only for the **Reset** (the marriage is not reciprocal) and **BadInit** rules.

Now, let v be in MEN. Because $v.phase \neq 1$, v cannot activate **ToPhase1.5**. Since women are in phase 2, the **ToPhase1** rule is also not enabled. Let us consider the **Accept** and **Confirm** rule and the two possible cases:

- $v.phase = 1.5$. Because of the predicate $\mathsf{AllCoherentPhase}(v)$, this rules cannot be applied.
- $v.phase = 2$. If a woman is proposing to v, v can accept the proposal if it is the best proposal regarding their preference lists. But if he accepts and since the woman cannot answer in this configuration, there is no new marriage. If he confirms, that means they were already married (the woman had his identifier in her pointer of marriage). In any cases, that does not create a marriage and thereby also not a blocking pair. Moreover, the phase of nodes activated for these rules is still 2.

Finally, we consider C', the new configuration after the transition from C. If at least one woman has been activated for **BadInit** or one man for **Reset**, C' is in $\mathcal{C}^1$. Otherwise, nodes have been activated for **ToPhase2**, **Accept** and **Confirm** (men) or **Reset** (women) and C' is in $(\{1.5, 2\}, \{2\}, 0) \bigcup (\{2\}, \{2\}, 0)$. □

Lemma 15. *Any execution starting from a configuration C in $(\{1.5\}, \{1.5\}, 0) \bigcup (\{1.5\}, \{1.5, 2\}, 0) \bigcup (\{1.5\}, \{2\}, 0) \bigcup (\{1.5, 2\}, \{2\}, 0)$ takes $O(n)$ moves to reach a configuration C' in $\mathcal{C}^1$ or in $(\{2\}, \{2\}, 0)$.*

Proof. In C, by Lemma 13, we know that each man is eligible only for the **Reset** rule and each woman for the **Reset** and **ToPhase2** rules. And when women are in phase 2, they are eligible for the **BadInit** rule. Then, at most n women in phase 1.5 are eligible for the **ToPhase2** and **Reset** rules. Men's **Reset** and women's **BadInit** rules are also eligible, but, after their activation, the configuration is in $\mathcal{C}^1$. The configuration is now in $(\{1.5\}, \{2\}, 0)$ where all pointers of proposal and marriage are coherent: there are no proposal (**ToPhase2** sets this pointer to `Null`) and women's marriages are reciprocal (otherwise, before **ToPhase2**, she was eligible for **Reset**). Now, by Lemma 14, we know that men are eligible for rules **ToPhase2**, **Reset**, **Accept** and **Confirm** and women are eligible only for the **Reset** and **BadInit** rules. Note that since proposal and marriage pointers of women are coherent, **Accept**, **Confirm**, **BadInit** and women's **Reset** are not eligible (see the proof of Lemma 14). Moreover, we know that after a man's **Reset**, the configuration in in $\mathcal{C}^1$. Then, after at most n **ToPhase2** rules, the reached configuration is in $(\{2\}, \{2\}, 0)$, that is C'.

In overall, after at most $3n$ moves, that is $O(n)$ moves, a configuration either in $(\{2\}, \{2\}, 0)$ or in $\mathcal{C}^1$ is reached.

Lemma 16. *Let C be a configuration in $(\{1.5\}, \{1.5\}, X)$ where $X > 0$. Any execution starting from C takes $O(n)$ moves to reach a configuration C' in $(\{1\}, \{1, 1.5\}, X)^{\times}$.*

Proof. Let us consider first a node v in Women. Since v is in phase 1.5 and all men are in phase 1.5, v is not eligible for any **Propose1/2**, **Confirm1/2** or **ToPhase1.5** rules. Since there are X blocking pairs, some women are involved in these blocking pairs. Women involved in a blocking pair are eligible for **ToPhase1** and the others are eligible for **ToPhase2** (because of the predicate BlockingPairW). Once they are in phase 2, while all men are in phase 1.5, they can do nothing (the **BadInit** rule is not eligible because after the **ToPhase2** rule, women's proposal pointer is set to `Null`. v is also eligible for the **Reset** rule.

Let us consider now a node v in Men. Since v is in phase 1.5, he cannot be eligible for the **Accept**, **Confirm** and **ToPhase1.5** rules. The **ToPhase1** and **ToPhase2** rules are also not eligible: there are at last X women in phase 1.5 and others in phase 2. Then v can only be eligible for the **Reset** rule.

In short, after at most n **Reset** (women) + $(n - X)$ women's **ToPhase2** the only eligible rule is **ToPhase1** of a woman involved in a blocking pair or a man's **Reset**. Then, after $2n$ moves (if $X = 1$), that is $O(n)$ moves, a configuration C' in $\mathcal{C}^1$ is reached. □

Lemma 17. *Let C be a configuration in $(\{1.5, 2\}, \{1.5\}, X) \bigcup (\{1.5, 2\}, \{1.5, 2\}, X) \bigcup (\{2\}, \{1.5\}, X)$ with $X \geq 0$. Any execution starting from C takes $O(n)$ moves to reach a configuration C' in $\mathcal{C}^1$.*

Proof. A common feature to the configurations specified in Lemma 17 is that: $\exists\, w \in \text{Women} \wedge \exists\, m \in \text{Men} : w.phase = 1.5 \wedge m.phase = 2$

Let us consider first a node v in MEN. Independently of phases, the **Reset** rule can be eligible. Notice that if a man is activated for **Reset**, the configuration is immediately in $\mathcal{C}^1$. Since there exist at least one woman in phase 1.5 and no node in phase 1, the predicate $\mathsf{AllCoherentPhase}(v)$ is `False` and then the rules **Accept** and **Confirm** are not eligible. For the same reason, the **ToPhase2** rule is also not eligible. Since nodes can only be in phase 1.5 or 2, the **ToPhase1.5** rule cannot be activated. Concerning the **ToPhase1** rule, it can be eligible only if the node v is in phase 2 (because there exists a woman in phase 1.5) or if v is involved in a blocking pair.

In short, men in phase 1.5 are only eligible for the **Reset** rule and men in phase 2 are eligible for the **Reset** or **ToPhase1** rules. (any of this two rules is sufficient to reach a configuration in $\mathcal{C}^1$).

Now, let us consider a node v in WOMEN. Independently of phases, the **Reset** rule may be eligible. Then v cannot be eligible for **Propose1**, **Confirm1**, **ToPhase2** and **ToPhase1.5** because of men in phase 2. Since v and all men are not in phase 2 together, rules **Propose2** and **Confirm2** are not eligible. But if v is in phase 2, v may be activated for **BadInit** (and then the system reaches a configuration in $\mathcal{C}^1$). Concerning the **ToPhase1** rule, since there is at least a man in phase 2, all women in phase 1.5 are eligible. Married women involved in a blocking pair in phase 2 are also eligible for this rule. To summarize, a woman v is eligible for

- **Reset**
- **ToPhase1** (if v is in phase 1.5 or in phase 2 with a blocking pair)

So, after at most n **Reset** (women), women are only eligible for the **ToPhase1** or **BadInit** rules and men for the **Reset** and **ToPhase1** rules. Then, after the next activation, that is altogether $O(n)$ moves, the configuration is in $\mathcal{C}^1$. We can note that here the problem comes from phases and not from blocking pairs. □

Lemma 18. *Let C be a configuration in $(\{2\}, \{1.5, 2\}, X)$ with $X \geq 0$. Any execution starting from C takes $O(n)$ moves to reach a configuration C' in $\mathcal{C}^1$.*

Proof. Let us consider first a node u in MEN. Independently of phases, the **Reset** rule can be enabled. Note that if a man is activated for **Reset**, the system reaches immediately a configuration in $\mathcal{C}^1$. Since there exists at least one woman in phase 1.5 and no node in phase 1, the predicate $\mathsf{AllCoherentPhase}(u)$ is `False` and then the rules **Accept** and **Confirm** are not enabled. Since men are in phase 2, the **ToPhase2** ans **ToPhase1.5** rules are also not enabled. Concerning the **ToPhase1** rule, it may be enabled because a woman is in phase 1.5.

In short, men are only eligible for the **Reset** or **ToPhase1** rules. (These two rules lead to a configuration in $\mathcal{C}^1$).

Now, let us consider a node u in WOMEN. Independently of phases, the **Reset** rule may be enabled. Let $u.phase = 1.5$. Then u cannot be eligible for **Propose1/2**, **Confirm1/2**, **BadInit** and **ToPhase1.5** because of the phase of u. Moreover, because men in phase 2, the **ToPhase2** rule is also not enabled. Then, u is eligible for the **ToPhase1** rule.

Now, let $u.phase = 2$. Then, u cannot be eligible for **Propose1**, **Confirm1**, **ToPhase2** and **ToPhase1.5** (because of u's phase). In case of incoherence between proposal and marriage pointers, u is eligible for the **BadInit** rule. But if she is activated for this rule, the system reaches a configuration in $\mathcal{C}^1$. Concerning **Propose2**, **Confirm2** and **ToPhase1**, there are several cases:

- u is married: **Propose2** and **Confirm2** are not enabled. However, u can be activated for a **ToPhase1** rule if u is involved in a blocking pair.
- u is single: u cannot be activated for a **ToPhase1** rule, but for the **Propose2** and **Confirm2** rules. We know that men cannot apply **Accept** or **Confirm**. But if *proposal* pointers are incoherent, a woman may propose to a man u and then confirm to u the marriage because of u's incoherent *proposal* pointer. Each woman may propose and confirm only once. Otherwise it would mean that m_1, a man better ranked for u, has been discovered after u's **Propose2** or **Confirm2**. But when u made her proposal to m, m_1 wasn't interesting for u (better marriage or incoherent pointers). In any case, this means that m_1 has been activated for a **Reset** rule and then should be in phase 1. That is in contradiction with the fact that all men are in phase 2 and the system is now in a configuration in $\mathcal{C}^1$. Furthermore, this new marriage between m and u can create a new blocking pair, but we will see later in this proof that the system will reach a configuration where a node is in phase 1.

To summarize, a woman u is eligible for

- **Reset**
- **BadInit** if u is in phase 2 with incoherence between pointers but the system reaches a configuration in $\mathcal{C}^1$.
- **ToPhase1** if u is in phase 1.5 or in phase 2 with a blocking pair.
- **Propose2** and **Confirm2** if u is in phase 2.

So, after at most n **Reset** (women), and $n-1$ **Propose2** and **Confirm2** (there is at most one woman in phase 1.5) moves, nodes are only eligible for rules that set the phase to 1 (**ToPhase1**, men's **Reset** and **BadInit**)

Then, after at most $O(n)$ moves, the system reaches a configuration in $\mathcal{C}^1$.

□

Lemma 19. *Any execution starting from a configuration C in* $(\{1.5\}, \{1.5, 2\}, X)$ *takes* $O(n)$ *moves to reach a configuration C' in* $\mathcal{C}^1$ *or in* $(\{1.5\}, \{2\}, X)$.

Proof. Let us consider first a node v in Men. Since there is at least a woman in phase 1.5 and one in phase 2, v is eligible only for **Reset**. After his move, the reached configuration is in $\mathcal{C}^1$.

Now, let us consider a node v in Women. v is eligible for **Reset** or **BadInit** (if $v.phase = 2$) if she has incoherent pointers, but not for both (After the move of a node with **Reset**, the guard of **BadInit** is `False` and if **BadInit** is applied, that means **Reset** was not enabled). For other rules, there are two cases:

– If v detects a blocking pair: she is eligible for the **ToPhase1** rule for any phase.
– If v does not detect a blocking pair and is in phase 1.5, she is eligible for the **ToPhase2** rule. Otherwise, she is eligible for any rule.

Then, if all women in phase 1.5 do not detect a blocking pair, and other nodes are not activated, a configuration in $(\{1.5\}, \{2\}, X)$ is reached after at most $2n - 1$ moves (n **Reset** and $n - 1$ **ToPhase2**). If there is at least one woman in phase 1.5, involving in and detecting a blocking pair, she is not eligible for **ToPhase2** but for **ToPhase1**. With at most the same number of moves, the reached configuration is in $\mathcal{C}^1$.

Then, in $O(n)$ moves, the reached configuration is either in $\mathcal{C}^1$ or in $(\{1.5\}, \{2\}, X)$. □

Lemma 20. *Any execution starting from a configuration C in $(\{1.5\}, \{2\}, X)$ takes $O(n)$ moves to reach a configuration C' in $\mathcal{C}^1$ or in $(\{1.5, 2\}, \{2\}, X)$.*

Proof. Let us consider first a node v in Men. Since all women are in phase 2, men are eligible for several rules: **Reset** and either **ToPhase2** or **ToPhase1**. Indeed, if a man is involved in a blocking pair, this is detected by the predicate BlockingPairM and this man is not eligible for **ToPhase2** (other women are not eligible but the woman involved in the blocking pair). A man detects anyway a blocking pair. Indeed, if a woman has incoherent pointers, that means she is not married and then, she cannot be in a blocking pair. After a transition where at least one man is activated, the reached configuration is in $\mathcal{C}^1$ (if at least one **Reset** has been activated) or in $(\{1.5, 2\}, \{2\}, X)$ (nodes have been ativated for only **ToPhase2**).

Now, let us consider a node v in Women. v is eligible for the **Reset** or **BadInit** rules if she has incoherent pointers, but not for both. Indeed, if v is activated for the **Reset** rule, then her guard of **BadInit** is `False` ($v.marriage$ and $v.proposal$ have been set to `Null`). And if **BadInit** is applied, that means that **Reset** was not enabled and after **BadInit**, it is still not enabled.

There are two cases for other rules:

– v is involved in a blocking pair and has her predicate BlockingPairW is `True`: she is eligible for the **ToPhase1** rule.
– v is not involved in a blocking pair and is eligible for any rule.

Then, women are eligible for at most n **Reset** and after that, men are eligible for **ToPhase2** and **Reset** and women for **ToPhase1** or **BadInit**. Then, after $O(n)$ moves, the reached configuration is then in $\mathcal{C}^1$ (after a man's **Reset** or a woman's **ToPhase1**) or in $(\{1.5, 2\}, \{2\}, X)$ (after only men's **ToPhase2**). □

Lemma 21. *Any execution starting from a configuration C in $(\{1.5, 2\}, \{2\}, X)$ takes $O(n)$ moves to reach a configuration C' in $\mathcal{C}^1$ or in $(\{2\}, \{2\}, X)$.*

Proof. Let us consider first a node v in Men. If v is in phase 1.5, he is eligible for several rules: **Reset** and **ToPhase2** if he does not detect a blocking pair.

If $v.phase = 2$, then he is eligible for several rules: **Reset** and **ToPhase1** (if involved in a blocking pair) but also **Accept** and **Confirm**. In fact, if a woman is proposing to v, he can accept the proposal if it is the best proposal regarding the preference lists. But if he accepts and since the woman cannot answer in this configuration (see later), there is no new marriage. If he confirms, that means they were already married (the woman had his identifier in her pointer of marriage). In any cases, that does not create a marriage and thereby also not a blocking pair. Moreover, the phase of nodes activated for these rules is still 2. After the move of a node with one of this rules, the reached configuration is in $\mathcal{C}^1$ (**Reset**) or in $(\{1.5, 2\}, \{2\}, X)$ (**ToPhase2**, **Accept** and **Confirm**).

Now, let us consider a node v in WOMEN. v is eligible for the **Reset** or **BadInit** rules if she has incoherent pointers, but not for both. Indeed, if v is activated for the **Reset** rule, then her guard of **BadInit** is `False` ($v.marriage$ and $v.proposal$ are set to `Null`). And if a **BadInit** is applied, that means that the **Reset** rule was not enabled and after the **BadInit** rule, it is still not enabled. There are two cases for other rules:

- v is involved in a blocking pair and has her predicate BlockingPairW to `True`: she is eligible for **ToPhase1**.
- v is not involved in a blocking pair and is eligible for any rule.

Then, after at most n **Reset** of women, the enabled rules are men's **Reset** (at most n) and **ToPhase2** (at most $n - X$) and women's **ToPhase1** and **BadInit**. After at most $n - 1$ **ToPhase2** of men, the reached configuration is in $(\{2\}, \{2\}, X)$ except if at least one man's **Reset**, **ToPhase1** or woman's **ToPhase1** and **BadInit** are applied.

In short, after $O(n)$ moves, the reached configuration is either in $(\{2\}, \{2\}, X)$ or in $\mathcal{C}^1$. □

Lemma 22. *Let C be a configuration in $(\{2\}, \{2\}, X)$ with $X \geq 0$. Any execution starting from a configuration C takes $O(n^2)$ moves to reach a configuration C' in $\mathcal{C}^1$ or in $(\{2\}, \{2\}, 0)$.*

Proof. Let us consider a woman w not involved in a blocking pair. There are two possibilities:

- she is married without blocking pair. In this case, she has nothing to do, except one **BadInit**.
- she is single. Then, she is eligible for the **Reset**, **Propose2** and **Confirm2** rules.

After her **Reset** (if she needs one), she is eligible for **Propose2**. There is now also two cases. She proposes to m and we assume that w is the best proposal for m. Then m accepts the proposal and both confirm one after the other. In all cases, because of the definition of $\mathcal{C}_v$ and $\mathcal{P}_v$, m decreases his regret (either he was single and is now married or he was married and is now with a better ranked spouse). If m was involved in a blocking pair (w_1, m), after this new marriage, the blocking pair may be resolved. Indeed, if w has a better priority for m than

w_1, there is no more blocking pairs (w_1, m). Note that the pair (w, m) was not a blocking pair because w was single.

If each BPs in C is resolved by a single woman, the number of blocking pair decreases and it can not grow since men are only improving their marriage. Since there are $O(n^2)$ possible matches, there are $O(n^2)$ blocking pairs. If all women make their proposals to each man in a blocking pair, in at most $O(n^2)$ moves, there is no more blocking pairs and the configuration is then in $(\{2\}, \{2\}, 0)$. If before resolving all the blocking pairs, a married woman involved in one of them is activated (for **ToPhase1**), the system reaches a configuration in $\mathcal{C}^1$.

Proposition 2. *Let C be a configuration in $(\{1.5, 2\}, \{1.5, 2\}, X)^\times$ with $X \geq 0$ and $C' \in \mathcal{C}^1$. In any execution, $C \to C'$ appears at most once.*

Proof. Let us suppose that the transition $C \to C'$ is possible twice. Let us denote by C_0 and D_0 configurations in $(\{1.5, 2\}, \{1.5, 2\}, X)$ where $X \geq 0$ and by C_1 and D_1 configurations in $\mathcal{C}^1$. Let T be the first transition $C_0 \to C_1$ where a node v is moving to phase 1 and T' the second transition $D_0 \to D_1$.

Let us analyze each case: first v is in WOMEN. She has two rules that change her phase to 1 : **ToPhase1** and **BadInit**. These two rules have the same actions: $v.proposal$ is set to `Null` and $v.phase$ to 1. Note that **BadInit** is enabled only in phase 2 and **ToPhase1** in both phase 1.5 and 2.

First, let us assume that in T, v is activated for **BadInit**. Then, in C_1, $v.proposal =$ `Null`. In this phase, she can propose and confirm marriage. But, since she is in phase 1.5 or 2 in D_0, there is a transition T_0 from C_2 to C_3 between C_1 and D_0, where she is activated for **ToPhase1.5**. By Lemma 10 there is no more BP with v and by Lemma 11 and 12, the reached configuration is in $(\{1.5\}, \{1.5\}, 0)$. The **ToPhase1** has set $v.proposal$ to `Null` and $v.phase = 1.5$, she is not eligible for both rules **ToPhase1** and **BadInit**. Then, v is enabled for **ToPhase2** and $v.proposal$ is set to `Null`: v is not eligible for **BadInit** in phase 2. . By Lemma 15, the reached configuration is in $(\{2\}, \{2\}, 0)$, there is also no blocking pair and she is not enabled for **ToPhase1**. Now, let us assume that in T, v is activated for **ToPhase1**. Thus, $v.proposal =$ `Null` in C_1. After that, the execution is the same as explain above.

To summarize, by contradiction, no woman can be enabled twice for a transition to phase 1 from configuration in $(\{1.5, 2\}, \{1.5, 2\}, X \geq 0)^\times$.

Because if a woman is in phase 1, men are eligible for **ToPhasse1** from phase 1.5 or 2 and because women cannot move twice to phase 1, men cannot be eligible the second time (T') for **ToPhase1**. Furthermore, men are not eligible for the **Reset** in T'. Indeed, $v.phase = 1$ in C_1 and $v.phase \in \{2, 1.5\}$ in D_0. Then, v has been eligible for at most one **ToPhase1.5**. Then, if he was eligible for **Reset**, it was before **ToPhase1.5**. Then, by Lemma 10 there is no more BP with v and by Lemma 11 and 12, the reached configuration is in $(\{1.5\}, \{1.5\}, 0)$. Finally, by Lemma 15, the reached configuration is in $(\{2\}, \{2\}, 0)$, there is also no blocking pair and she is not enabled for **ToPhase1**. In all this configurations, men are not eligible for the **ToPhase1** (women cannot move to phase 1) and **Reset** (No new marriage/proposal)

To summarize, men and women are not eligible for rules that change their phase to 1 and perform T'. In any execution, $C \to C$' appears at most once. □

Proposition 3. *Any execution takes $O(n^4)$ moves to reach a configuration C in $(\{2\},\{2\},0)$.*

Proof. For each set of configurations $\mathcal{C}' = (\{1.5,2\},\{1.5,2\},X)^{\times}$ with $X >= 0$ listed below, we show how any execution starting from a configuration in $\mathcal{C}'$ reaches a configuration either in $\mathcal{C}^1$ or in $(\{2\},\{2\},0)$. For doing that, we indicate the lemmas justifying the reachability from one set of configurations to another. Note that each such sub-execution takes $O(n^2)$ moves.

1. From $(\{1.5\},\{1.5\},X)$ to:
   - $(\{1\},\{1,1.5\},X)^{\times}$, for $X > 0$: Lemma 16.
   - $\mathcal{C}^1$ or $(\{2\},\{2\},0)$, for $X = 0$: Lemma 15.
2. From $(\{1.5\},\{1.5,2\},X)$ to $\mathcal{C}^1$ or $(\{1.5\},\{2\},X)$:
   - for $X > 0$: Lemma 19,
   - for $X = 0$: Lemma 15.
3. From $(\{1.5\},\{2\},X)$ to $\mathcal{C}^1$ or $(\{1.5,2\},\{2\},X)$:
   - for $X > 0$: Lemma 20,
   - for $X = 0$: Lemma 15.
4. From $(\{1.5,2\},\{2\},X)$ to: $\mathcal{C}^1$ or $(\{2\},\{2\},X)$:
   - for $X > 0$: Lemma 21,
   - $X = 0$: Lemma 15.
5. From $(\{1.5,2\},\{1.5\},X)$ to $\mathcal{C}^1$, for $X \geq 0$: Lemma 17.
6. From $(\{1.5,2\},\{1.5,2\},X)$ to $\mathcal{C}^1$, for $X \geq 0$: Lemma 17.
7. From $(\{2\},\{1.5\},X)$ to $\mathcal{C}^1$, for $X \geq 0$: Lemma 17.
8. From $(\{2\},\{1.5,2\},X)$ to $\mathcal{C}^1$, for $X \geq 0$: Lemma 18.
9. From $(\{2\},\{2\},X)$ to $\mathcal{C}^1$ or $(\{2\},\{2\},0)$, for $X \geq 0$: Lemma 22.

Now, we consider a configuration C' in $\mathcal{C}^1$. By Lemma 7, any execution starting from C' takes $O(n^2)$ moves to reach a configuration C_1 in $(\{1\},\{1,1.5\},X)^{\times} \bigcup (\{1.5\},\{1.5\},0)$. If C_1 is in $(\{1.5\},\{1.5\},0)$, the case is listed above (item 1). If C_1 is in $(\{1\},\{1,1.5\},X)^{\times}$, by Lemma 10, any execution starting in C_1 takes $O(n^4)$ to reach a configuration C_2 in $(\{1\},\{1,1.5\},0) \bigcup (\{1\},\{1.5\},0)$. Then, by Lemmas 11 and 12, any execution from C_2 takes $O(n)$ moves to reach a configuration in $(\{1.5\},\{1.5\},0)$. From there, by Lemma 15, any execution takes $O(n)$ moves to reach a configuration either in $\mathcal{C}^1$ or in $(\{2\},\{2\},0)$. Thus, starting from C', any execution reaches a configuration either in $\mathcal{C}^1$ or in $(\{2\},\{2\},0)$.

By Proposition 2, any execution starting from a configuration $\mathcal{C}'$ contains at most one transition to $\mathcal{C}^1$.

In summary, we have listed above all possible types of configurations and showed that, in each case, a configuration in $(\{2\},\{2\},0)$ is reached in $O(n^4)$ moves. □

Convergence to a Stable Marriage

Lemma 23. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Assume that a transition $D_0 \to D_1$ in $\mathcal{E}$ results from the activation of (a rule of) a woman w.*

1. *The activated rule is not in* $\{$**ToPhase1**, **BadInit**$\}$*;*
2. *If $w.marriage =$* `Null` *in D_0, then the activated rule is either* **Propose2** *or* **Confirm2***;*
3. *If $w.marriage \neq$* `Null` *and* Married(w) *is* `False` *in D_0, then the activated rule is rule* **Reset***.*

Moreover, a woman w_1 is enabled for any rule in D_0 if Married(w_1) *is* `True` *in D_0.*

Proof. Since w is in phase 2 in D_0 (assumption of this lemma), w is enabled for any rule in $\{$**ToPhase1.5**, **ToPhase2**, **Confirm1**, **Propose1**$\}$. Moreover, since w remains in phase 2 in D_1, w can not execute rules **ToPhase1** and **BadInit**. If it is the case, then w will be in phase 1 in D_1, and this fact contradicts the assumption of this lemma.

Assume that w executes a rule in $D_0 \to D_1$. We consider two cases.

First, if $w.marriage =$ `Null` in D_0, then w is eligible for rules **Propose2** and **Confirm2** in D_0. And w executes one of theses rules.

Second, if $w.marriage = m$ in D_0, then $w.proposal =$ `Null` in D_0 (otherwise w executes **BadInit**).

if $m.proposal = w$ in D_0, then Married(w) is *`True`* in D_0. Moreover, w is not eligible for any rule. So, w can not execute any rule. So $m.proposal \neq w$ in D_0 if w executes a rule in $D_0 \to D_1$. Thus IncoherentPointersW(w) $= false$ in D_0, and Married(w) is false. And, w executes rule **Reset** in $D_0 \to D_1$. □

Lemma 24. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such for each configuration in $\mathcal{E}$, all nodes are in phase 2. Assume that a transition $D_0 \to D_1$ in $\mathcal{E}$ results from the activation of (a rule of) a man m.*

1. *The activated rule is either* **Accept** *or* **Confirm***.*
2. *If* AlreadyEngaged(m) *in D_0, then the activated rule is* **Confirm***.*

Proof. Assume that m executes a rule in $D_0 \to D_1$. By definition of $\mathcal{E}$, m does not execute **Tophase1**, **Tophase1.5**, **Tophase2** and **Reset** during $D_0 \to D_1$.

Assume that AlreadyEngaged(m) in D_0. In D_0, m is enabled for **Confirm** in D_0 (due to its guard). Moreover, since not execute **Accept** in $D_0 \to D_1$ □

Lemma 25. *Let m be in* Men*. Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive activations by m of the same rule.*

1. *If this rule is* **Confirm***, then m exactly executes one* **Accept** *rule between D_1 and F_0.*

2. *If this rule is* **Accept**, *then* m *exactly executes one* **Confirm** *rule between* D_1 *and* F_0.

Proof. First, let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive **Confirm** executed by m.

We prove that m executes an **Accept** during these two transitions at least once. We have: $m.proposal = \texttt{Null}$ in D_1 and $m.proposal \neq \texttt{Null}$ in F_0 according to the **Confirm** rule. So, m has to execute a rule to modify its *proposal*-variable between D_1 and F_0. Since $\mathcal{E}$ is a sub-execution such that for each configuration in $\mathcal{E}$, all nodes are in phase 2, m can execute only **Accept** or **Confirm**. Among the two rules **Confirm** or **Accept**, there is one rule doing that: **Accept**. Thus, m executes such that a rule at least once between D_1 and F_0.

Second, Let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive rules **Accept** executed by m. We prove that m executes a **Confirm** during these two transitions at least once.

We now prove the second point. According to the **Accept** rule, $m.proposal = \texttt{Null}$ in D_1 and $m.proposal \neq \texttt{Null}$ in F_0. So, m has to execute a rule between D_1 and F_0 to modify its *proposal*-variable. Since $\mathcal{E}$ is a sub-execution such for each configuration in $\mathcal{E}$, all nodes are in phase 2, m can execute only **Accept** or **Confirm** rule. Among these two rule, there is one rule doing that: **Accept**. Thus, m executes such a rule at least once between D_1 and F_0.

By combining the previous two facts, the lemma holds. □

Lemma 26. *Let* $\mathcal{E}$ *be a sub-execution starting from* C_1 *and ending at* C_2 *such for each configuration in* $\mathcal{E}$, *all nodes are in phase* 2. *Let* m *be in* MEN. *Let* $D_0 \to D_1$ *be a transition in which* m *executes rule* **Accept**. *Let* C *be a configuration after* D_1 *in which* $\mathsf{Married}(m) = \texttt{True}$. *In every configuration* D *after* C, *we have* $\mathsf{Married}(m)$.

Proof. Let $D_0 \to D_1$ be a transition in which m executes rule **Accept**.

Let w_1 be a woman such that $m.marriage(C) = w_1$ Let D be a configuration after C. Assume that $m.marriage$ remains constant between C and D. Since $\mathsf{Married(m)}$ in C, then $\mathsf{Married}(w_1)$= $True$ in C. So from Lemma 23, f_1 can not execute any rule. Thus $\mathsf{Married(m)} = \texttt{True}$ and the lemma holds.

Assume that $m.marriage$ does not remain constant between C and D.

Let $D_4 \to D_5$ be the first transition after C such that $m.marriage(D_4) = w_1$ and $m.marriage(D_5) = w_2$ with $w_2 \neq w_1$. To change its marriage value, h must execute rule **Confirm** in $D_4 \to D_5$. By definition of rule **Confirm**, $m.proposal(D_4) = w_2$, and $w_2.marriage(D_4) = m$. Since $w_2.marriage(D_4) = m$, it implies that $\mathsf{Married}(w_2)$, and Lemma 23 implies that w_2 does not execute any rule. Thus, in D_5, $\mathsf{Married(m)}$.

Now, we will prove the second point of this lemma. Let w_1 be a woman such that $m.marriage(C) = w_1$ Let D be a configuration after C.

Assume that $m.marriage$ remains constant between C and D. Since $\mathsf{Married(m)}$ in C, then $\mathsf{Married}(w_1)$= $True$ in C. So from Lemma 23, f_1 can not execute any rule. Thus $\mathsf{Married(m)} = \texttt{True}$ and the lemma holds.

Assume that $m.marriage$ does not remain constant between C and D.

Let $D_4 \to D_5$ be the first transition after C such that $m.marriage(D_4) = w_1$ and $m.marriage(D_5) = w_2$ with $w_2 \neq w_1$. To change its marriage value, m must execute rule **Confirm** in $D_4 \to D_5$. By definition of rule **Confirm**, $m.proposal(D_4) = w_2$, and $w_2.marriage(D_4) = m$. Since $w_2.marriage(D_4) = m$, it implies that $\mathsf{Married}(w_2)$, and Lemma 23 implies that w_2 does not execute any rule. Thus, in D_5, $\mathsf{Married}(\mathsf{m})$.

If $D_5 < D$, then we iterate the same argument where D_5 becomes C. □

Lemma 27. *Let m be in* Men. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive activations by m of the same rule.*

1. *If the rule is* **Confirm**, $p(m, m.marriage(D_1)) > p(m, m.marriage(F_1))$.
2. *If the rule is* **Accept**, $p(m, m.proposal(D_1)) > p(m, m.proposal(F_1))$.

Proof. Now, we will prove the first point. First, let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive rules **Confirm** executed by m.

From Lemma 25, there only exists one transition $A \to B$ between D_1 and F_0 in which m executes rule **Accept**. From the definition of rule **Accept**, we have $p(m, m.marriage(A)) > p(m, m.proposal(B))$.

Since m does not execute any rule between D_1 and A, his local variables remain constant, and $p(m, m.marriage(D_1)) > p(m, m.proposal(B))$.

Moreover, since m does not execute any rule between B and F_0, his local variables remain constant, and $p(m, m.marriage(D_1)) > p(m, m.proposal(F_0))$.

Thus, since from the definition of rule **Accept**, we have $p(m, m.marriage(F_0)) > p(m, m.proposal(F_1))$, we can conclude that we have

$$p(m, m.marriage(D_1)) > p(m, m.marriage(F_1)).$$

We will prove the second point. Let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive rule **Accept** executed by m. From Lemma 25, there only exists one transition $A \to B$ between D_1 and F_0 in which m executes rule **Confirm**. From the definition of rule **Confirm**, we have $m.marriage(B) = m.proposal(A)$.

From the definition of rule **Accept**, we have

- $p(m, m.marriage(D_0)) > p(m, m.proposal(D_1))$
- $p(m, m.marriage(F_0)) > p(m, m.proposal(F_1))$.

So, we can conclude that $p(m, m.proposal(D_1)) > p(m, m.proposal(F_1))$. □

Lemma 28. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Let m be in* Men. *Assume that C_1 is in $(\{2\}, \{2\}, 0)$ such that*

1. *no man is enabled for a* **Reset** *rule;*
2. *no woman w is enabled for a* **BadInit** *rule and if $w.proposal \neq$ `Null`, then $w.proposal$ $=$ $\mathsf{BestMarriage}(w)$.*

Then, in every configuration D after C, $p(m, m.marriage(C)) \geq p(m, m.marriage(D))$. Moreover, $p(m, m.marriage(C)) > p(m, m.marriage(D))$ if $m.marriage(C) \neq m.marriage(D)$.

Proof. Let $D_0 \to D_1$ be the first transition after C_1 executed by m. Since the local value does not change between C_1 and D_0, for all configurations $C_1 \leq D \leq D_0$, we have $p(m, m.marriage(C)) = p(m, m.marriage(D))$.

From now, we assume that m executes at least one rule before C.

First, assume m executes rule **Accept**.

If no rule is executed by m between D_1 and C, then $p(m, m.marriage(C)) \geq p(m, m.marriage(D))$ and $m.marriage(C) \neq m.marriage(D)$. Otherwise, m executes a rule **Confirm** between D_1 and C. Let $D_2 \to D_3$ be the first transition **Confirm** executed by m between D_1 and C. By definition, of rule **Confirm**, we have,

$$p(m, m.marriage(D_2)) > p(m, m.marriage(D_3)) = p(m, m.proposal(D_2))$$

From now, we can build a sequence of transitions $(A_i \to B_i)_{(2 \leq i)}$ after D_0 in which m executes rule **Accept**. From Lemma 27, we have $p(m, m.marriage(B_i)) > p(m, m.proposal(B_i)) > p(m, h.proposal(B_{i+i}))$.

So for two configurations D and C such that $D_1 < D < C$, we have

$$p(m, m.marriage(D_1)) \geq p(m, m.marriage(D) \geq p(m, m.marriage(C))$$

Moreover if $m.marriage(C) \neq m.marriage(D)$, it implies that m executes rule **Confirm** between C and D and $p(m, m.marriage(D) \geq p(m, m.marriage(C))$. Second, for the case where m executes rule **Confirm**, we apply the same result in the first case, using Lemma 27. □

Lemma 29. *Let w be in* Women. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Assume that C_1 is in $(\{2\}, \{2\}, 0)$ such that*

1. *no man is enabled for a* **Reset** *rule;*
2. *no woman w is enabled for a* **BadInit** *rule and if $w.proposal \neq$ `Null`, then $w.proposal$ = BestMarriage(w).*

Let $D_0 \to D_1$ and $F_0 \to F_1$ be two transitions corresponding to two consecutive activations of **Propose2** *by w. Then, we have*

1. *$p(w, w.proposal(D_1)) < p(w, w.proposal(F_1))$*
2. *$w.proposal(D_1) \neq w.proposal(F_1)$*
3. *Let C be a configuration such that $D_1 < C < F_1$. If* Married(w) *is* `True` *in C, then* BlockingPairW(w) *is* `False` *in C.*

Proof. In D_0 and F_0, we have $w.marriage =$`Null` and $w.proposal =$ `Null`.

We have $m_2 =$BestMarriage(w) in F_1 from definition of rule **Propose2**. From Lemma 28, we have $p(m_2, m_2.marriage(D_0)) \geq p(m_2, m_2.marriage(F_0))$. So, it implies that m_2 belongs to $\mathcal{C}_w$ in D_0.

Since $m_1 = \mathsf{BestMarriage}(w)$ in D_1, we have $p(w, m_1) \leq p(w, m_2)$. To prove that $p(w, m_1) < p(w, m_2)$, we prove that $m_1 \neq m_2$.

Moreover, $p(m_1, m_1.marriage) > p(m_1, w)$ in D_0.

Let $D_2 \to D_3$ be the last transition before F_0 in which $w.proposal(D_3) \neq m_1$. Observe that it can be the transition $F_0 \to F_1$.

By assumption, variable $w.proposal$ remains constant between D_1 and D_2. Since w does not execute rule **BadInit** (Lemma 23), variable $w.marriage$ equals `Null` between D_1 and D_2. This implies that

1. *Married(w)* is false between D_1 and D_2;
2. w can execute rules **Propose2** or **Confirm2** using Lemma 23,

We consider these two cases.

First, assume that w executes rule **Propose2** in $D_2 \to D_3$. We are in the case where $D_2 \to D_3$ is the same transition $F_0 \to F_1$. Between D_1 and D_2, the local variable of w remains contained. It implies that In D_2, we have $m_1 \neq \mathsf{BestMarriage}(w)$, and $m_1 \neq m_2$.

Second, assume that w executes rule **Confirm2** in $D_2 \to D_3$. So, in D_2, we have $m_1.proposal = w$, $w.proposal = m_1$, and $\mathsf{AlreadyEngaged}(m_1)$. Thus, in D_2, m_1 is enabled for rules **Accept** and **Confirm**. Thus in D_3, $\mathsf{Married}(w)$. Moreover, Lemma 28 implies that

$$\text{for } \forall m \in \textsc{Men},\ p(m, m.marriage(D_0)) \geq p(m_2, m_2.marriage(D_2)).$$

Thus, this means that *BlockingPairW(w)* is `False` in D_3.

Since F_0, $w.marriage = \texttt{Null}$, then w should execute a rule between D_3 and F_0. Let $D_4 \to D_5$ be the first transition between in D_3 and F_0 in which w should execute a rule. Since in D_4, $w.marriage = m_1$, then $\mathsf{Married}(w)$ equal false (Lemma 23). Thus $m_1.marriage \neq w$ in D_4. Thus, there exists an transition in which m_1 executes **Confirm** between D_3 and D_4. Let $H_1 \to H_2$ be the first transition after D_3 in which m_1 executes **Confirm**. So, it implies that in H_1, $m_1.marriage = w$ and $m_1.marriage \neq w$. Since $m_1.marriage$ remains constant between D_3 and H_1, $\mathsf{Married}(w)$ is `True` between D_3 and H_1. Using the same argument for configuration D_3, *BlockingPairW(w)* is `False` between D_3 and H_1.

So using Lemma 28, we have

$p(m_1, w) > p(m_1, m_1.marriage(D_4) \geq p(m_1, m_1.marriage(F_0))$. Thus m_1 is not in $\mathcal{C}_w$ in F_0, and $m_1 \neq m_2$.

And this concludes the proof. □

Lemma 30. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Let w be in* Women. *Let $C_0 \to C_1$, $C_2 \to C_3$, $C_4 \to C_5$ be three transitions corresponding to three consecutive rules executed by w. Then w executes rule* **Propose2** *once between C_0 and C_5.*

Proof. Using Lemma 23, w can only execute **Propose2**, **Reset**, and **Confirm2**.

Assume that in $C_0 \to C_1$, w executes rule **Confirm2**. Then, in C_1, there exists a man m such that $w.marriage(C_1) = m$.

Since w does not execute any rule, then in C_2, we have $w.marriage(C_2) = m$. Using Lemma 23, since w executes a rule in $C_2 \to C_3$, w executes rule **Reset** in $C_2 \to C_3$. Moreover, since in C_3, $w.marriage(C_3) = \texttt{Null}$, Lemma 23 implies that w executes rule **Propose2** in transition $C_4 \to C_5$. And the lemma holds

We apply the same argument when w executes rule **Confirm2** in $C_0 \to C_1$. □

Proposition 4. *Let C be a configuration in $(\{2\}, \{2\}, 0)$, where*

1. *a man is enabled for* **Reset** *or*
2. *a woman w is enabled for* **BadInit** *or if $w.proposal \neq \texttt{Null}$ then $w.proposal \neq$ BestMarriage(w).*

Any execution starting from C takes $O(n^4)$ moves to reach a configuration C' in $(\{2\}, \{2\}, 0)$, where neither of these conditions is satisfied.

Proof. Let m be a man eligible for **Reset**. As nodes do not change their phase because there is no BP, by Corollary 5, after $O(n^2)$ moves, there is no other remaining moves than the m's **Reset**. m is eligible for the **Reset**, it means that $[(m.marriage.marriage \neq m) \vee (m.marriage = m.proposal)]$.
In the first case, if $(m.marriage.marriage \neq m)$ in C, men's pointers do not change without **Reset**. In fact, a woman w cannot set $w.marriage$ to m. Indeed, a woman cannot propose to m because of the definition of $\mathcal{C}_v$ (since $m.marriage = w$, $p(m, w) < p(m, m.marriage)$ is not *True*). If the proposition pointer of w is already set to m, she can only confirm if $m.proposal$ is set to w. We are also in the second case of the condition of **Reset**. Thus, $(m.marriage.marriage \neq m)$ is always *True* until m is activated for **Reset**.
Second case, if $(m.marriage = m.proposal)$ is always *True* until the **Reset** is done because of the predicate $\neg$IncoherentPointersM(m) included in each rule (**Reset** is the only eligible rule if its guard is *True*). Then, m stays eligible for the **Reset** rule. The configuration after this move is in a configuration C_1 with $(\exists\, v \in V : v.phase = 1)$.

Now, let us consider w. First, we consider the case where she is enabled for **BadInit**. Since she is in phase 2, $w.marriage \neq \texttt{Null} \wedge w.proposal \neq \texttt{Null}$. But the predicate IncoherentPointersW is `False` (otherwise, $w.marriage$ would be reset to `Null` and the **BadInit** rule not enabled). Thus, we also have $v.marriage \neq v.proposal$. Therefore, $w.marriage = w_1$ and $w.proposal = w_2$. Note first that w is married, otherwise she would be eligible for **Reset**. As such, **Propose2** and **Confirm2** are not enabled. Furthermore, there is no blocking pair, thus **ToPhase1** is not enabled. The state of the node can only change with the **BadInit** rule. After this move, the configuration is in $\mathcal{C}^1$.

Now, consider the case where $w.proposal \neq$ BestMarriage(w) and let us see all the sub-cases. We make the assumption that $w.marriage = \texttt{Null}$ if $w.proposal \neq \texttt{Null}$. Otherwise, we are in the previous case.

1. If BestMarriage(w) = `Null` then $w.proposal = m_1$

- If w is not married, m_1 can accept the proposal, but w cannot confirm (because $\mathsf{BestMarriage}(w) \neq w.proposal$) and is only eligible for the **Propose2** rule for the same reason. The reached configuration is thus in $(\{2\},\{2\},0)$ where the two conditions of this proposition statement are no satisfied.
- If w is married, m_1 can accept the proposal if this marriage is more interesting for him. Since $w.marriage \neq \texttt{Null}$, w cannot confirm the marriage. Moreover, he is unable to propose to another woman ($w.marriage \neq \texttt{Null}$). Thus, he is only eligible for one rule : **BadInit**. The reached configuration is in the set of configurations with $(\exists\, v \in V : v.phase = 1)$.

2. If $\mathsf{BestMarriage}(w) = m_1$ then either:
   - $w.proposal = \texttt{Null}$. It is the normal case: if w is married, she is enabled for the **ToPhase1** rule. Otherwise, she is enabled for **Propose2**.
   - $w.proposal = m_2$. Since $\mathsf{BestMarriage}(\mathrm{w}) \neq w.proposal$, w cannot confirm if m_2 accepts the proposal. Since $w.marriage = \texttt{Null}$, w is eligible for the **Propose2** rule.

To summarize, either the reached configuration C is in the set of configurations where $\exists\, v \in V : v.phase = 1$ or in $(\{2\},\{2\},0)$ where the two conditions of this proposition statement are no satisfied. If C_1 is $\mathcal{C}^1$, by Lemmas 7, 10,11, 12 and 15, the system reaches a configuration C' in $(\{2\},\{2\},0)$ in $O(n^4)$ moves. All nodes have been activated for transition rule **ToPhase2** and they have reset their *proposal* pointer. Because **Reset** is mutually exclusive with other rules (thanks to the predicate $\mathsf{IncoherentPointersM}$), if necessary, *marriage*-pointers have been reset before the activation for **ToPhase2**. Then C' has no woman enabled for **BadInit** or man for **Reset**, and if $w.proposal \neq \texttt{Null}$ then $w.proposal = \mathsf{BestMarriage}(w)$. □

Proposition 5. *Let $\mathcal{E}$ be a sub-execution starting from C_1 and ending at C_2 such that for each configuration in $\mathcal{E}$, all nodes are in phase 2. Assume that C_1 is in $(\{2\},\{2\},0)$ such that*

1. *no man is enabled for a* **Reset** *rule;*
2. *no woman w is enabled for a* **BadInit** *rule and if $w.proposal \neq \texttt{Null}$, then $w.proposal = \mathsf{BestMarriage}(w)$.*

$\mathcal{E}$ contains at most $O(n^2)$ moves.

Proof. Let w be a woman. Let $A_i \to B_i$ be a transition in which w executes rule **Propose2**. Let α be the number of times where w executes rule **Propose2** in $\mathcal{E}$. From Lemma 29, we have $w(w, w.proposal(A_i)) < w(w, w.proposal(A_{i+1}))$, for $1 \leq i < \alpha$. So, since the function $w()$ is upper bounded by $n+1$, then w executes rule **Propose2** in $\mathcal{E}$ at most $n+2$ times. From Lemma 30, if w executes three consecutive rules, them one of them corresponds to an execution of rule **Propose2**. Thus, w executes at most $O(n)$ moves.

Let m be a man. Using Lemma 27 and applying the same result as previously, w executes rule **Accept** (resp. **Confirm**) at most $O(n)$ times. In total, $\mathcal{E}$ contains at most $O(n^2)$ moves. □

Corollary 1. *Let $\mathcal{E}$ be an execution starting from a configuration in $(\{2\},\{2\},0)$ such that*

1. *no man is enabled for the* **Reset** *rule,*
2. *no woman w is enabled for the* **BadInit** *rule and either $w.proposal = \texttt{Null}$ or $w.proposal = \mathsf{BestMarriage}(w)$.*

$\mathcal{E}$ contains $O(n^2)$ moves.

Proof. First, we will prove by contradiction that all configurations after C_1 are in $(\{2\},\{2\},0)$. We assume that there exists a sub-execution $\mathcal{E}'$ starting from C_1 and ending by transition $C_2 \to C_3$ such

1. all nodes are in phase 2 for each configuration between C_1 and C_2 in $\mathcal{E}'$,
2. there exists a node in phase 1 in C_2 in $\mathcal{E}'$,

First, assume there is a woman w is in phase 1. Thus it implies that w executes a rule in order to change its phase. Since all nodes are in phase 2 in C_2, w executes rule **ToPhase1**. So, it implies that w is married and $\mathsf{BlockingPairW}(w)$ is *True*. This contradicts the fact that if $\mathsf{Married}(w)$ is *True* in C then $\mathsf{BlockingPairW}(w)$ is false in C (see Lemma 29)

Thus there exists a man m such that m is in phase 1. Thus it implies that m executes a rule in order to change its phase. Since all nodes are in phase 2 in C_2, m executes rule **Reset**. This contradicts the assumption of this corollary.

To sum up, all configurations after C_1 are in $(\{2\},\{2\},0)$ and the corollary holds by applying Proposition 5. □