

DEFERRED ACCEPTANCE IN LEAN FOR FINITE STRICT PREFERENCE MARKETS

ARTHUR FREITAS RAMOS, DAVID BARROS HULAK, AND RUY J. G. B. DE QUEIROZ

ABSTRACT. We describe a Lean 4 development of men-proposing deferred acceptance for finite one-to-one markets with complete strict preferences and optional partners. The two sides may have different cardinalities; the principal theorems assume a nonempty receiving side. A proposal-set measure proves termination of a specified, classically chosen run. Reachability invariants establish consistency of the resulting partner maps, stability, and the fact that a rejected proposal cannot be a stable-achievable partnership. The public optimality theorem is conditional on a man receiving a partner; an internal lemma separately excludes stable-achievable partners for men unmatched at quiescence. We give mathematical proofs corresponding to the source architecture, identify the precise theorem interface, and separate fresh source inspection from the repository’s reported build and axiom audit. This is an exposition of a pinned formalization of classical results, with no claim of new matching theory or priority of formalization.

1. INTRODUCTION

Gale and Shapley introduced deferred acceptance as a way to construct stable allocations from two-sided preferences [3]. This note explains a Lean 4 development of the one-to-one, men-proposing procedure. The contribution of the note is an explicit correspondence between a mathematical argument and the definitions, invariants, and theorem statements in the source. The underlying existence and proposing-side optimality results are classical. We make no novelty or first-formalization claim.

The development uses Lean 4 [2] and Mathlib [5]. Throughout, “men” and “women” are the source’s names for the proposing and receiving sides; they carry no further mathematical meaning. The model permits unequal side sizes and unmatched participants, while regarding every partner as preferable to being unmatched. Its scope is therefore narrower than models with ties, unacceptable partners, or many-to-one capacities.

Every source reference in this note is pinned to the repository [4] at commit

`f5f34f68a0e38441cdf5632adde5420d5fe0021e`.

This was also the repository’s main-branch head when inspected on September 30, 2026. The present note does not extend the formalization beyond that revision. Sections 2–5 explain the mathematics; Section 6 identifies the formal interface; Sections 7–8 state the evidence, limitations, and preparation provenance.

2. THE FINITE MARKET MODEL

Let M and W be finite types with decidable equality, and assume $W \neq \emptyset$. No assumption $|M| = |W|$ is imposed, and the principal theorems do not require $M \neq \emptyset$. A profile p assigns to each $m \in M$ a decidable strict preference relation $\succ_m$ on W , and to each $w \in W$ a decidable strict preference relation $\succ_w$ on M . Both are irreflexive, transitive, and total

Date: September 30, 2026.

2020 Mathematics Subject Classification. 91B68, 68V20.

Key words and phrases. Stable matching, deferred acceptance, Gale–Shapley theorem, Lean 4, Mathlib.

Author ORCID identifiers: Arthur Freitas Ramos, 0009-0003-3568-0325; David Barros Hulak, 0009-0002-8056-1774; Ruy J. G. B. de Queiroz, 0000-0003-1482-0977.

Copyright 2026 the authors. This manuscript is licensed under Creative Commons Attribution 4.0 International (CC BY 4.0).

on distinct elements. Thus, for example, $v \neq w$ implies $v \succ_m w$ or $w \succ_m v$. These are the fields of **Profile**.

For a type X , write $X_\perp$ for its optional extension, where $\perp$ denotes being unmatched and $x \in X$ denotes a present partner. The source uses **Option** **X**, **none**, and **some** **x**. Extend comparison of a real partner with an optional current partner by

$$w \succ_m^* a \iff \begin{cases} \text{true}, & a = \perp, \\ w \succ_m v, & a = v \in W, \end{cases}$$

and analogously for $m \succ_w^* b$. This is exactly the convention in **prefersM** and **prefersW**; there is no separate acceptability predicate.

Definition 2.1 (Matching and stability). A matching consists of maps $\mu_M : M \rightarrow W_\perp$ and $\mu_W : W \rightarrow M_\perp$ satisfying

$$(1) \quad \mu_M(m) = w \iff \mu_W(w) = m.$$

A pair (m, w) blocks μ if

$$w \succ_m^* \mu_M(m) \quad \text{and} \quad m \succ_w^* \mu_W(w).$$

The matching is stable if it has no blocking pair. A partner w is stable-achievable for m , written $\text{Ach}_p(m, w)$, if there exists a stable matching μ with $\mu_M(m) = w$.

The two directions in (1) are stored as the fields **inv1** and **inv2** of **Matching**. They imply that no agent has two partners. Since all partners are acceptable in the above sense, stability also rules out having both an unmatched man and an unmatched woman. This last observation is a mathematical consequence of the definition, not an additional theorem in the three-result public interface.

3. PROPOSAL STATES AND FINITE TERMINATION

3.1. The specified step. A proposal state is $s = (h, P)$, where $h : W \rightarrow M_\perp$ records each woman's held proposer and $P \subseteq M \times W$ records all proposals made so far. The structure **DAState** does not itself require consistency of holders; that property is proved for states reachable from the initial state $s_0 = (w \mapsto \perp, \emptyset)$.

A man is active in s when no woman holds him and there is at least one woman to whom he has not proposed. If there is an active man, the function **DA_step** selects one using classical choice. His next proposal is to his most-preferred member of

$$U_s(m) = \{w \in W : (m, w) \notin P\}.$$

The lemma **exists_prefM_maximal** proves existence of such a maximal member of every nonempty finite set by induction on the set. Strict totality and transitivity justify its characterization as preferred to every distinct remaining member. The function **nextWoman** chooses that member, and **nextWoman_maximal** supplies this specification.

Suppose the selected proposal is (m, w) . If $h(w) = \perp$, the woman holds m . If $h(w) = q$, she holds m exactly when $m \succ_w q$ and otherwise retains q . All other holders stay unchanged, and

$$(2) \quad P' = P \cup \{(m, w)\}, \quad (m, w) \notin P.$$

If there is no active man, the step returns **none** and makes no transition. A state is quiescent when its set of active men is empty.

The detailed specifications **DA_step_proposal_info** and **DA_step_held_info** isolate these facts from the implementation. The update uses an optional-holder default, **getD** **m**, so an initially unheld woman also acquires m even though $m \not\succ_w m$. All subsequent proofs use the specifications rather than assuming an informal algorithm silently has these properties.

3.2. The decreasing measure and the run. Put $N = |M \times W| = |M| |W|$ and define

$$T(s) = N - |P| \in \mathbb{N}.$$

Because P is a finite set of pairs, $|P| \leq N$. By (2), each successful step increases its cardinality by exactly one. Hence $T(s') < T(s)$, as proved by `terminationMeasure_decreases`.

The source does not define the run by unbounded recursion. Instead, `runFuel p n s` executes at most n proposal steps, stopping whenever `DA_step` returns `none`. The chosen run is

$$\text{run}(p, s) = \text{runFuel}(p, T(s) + 1, s).$$

The relation `DARreaches p s t` is generated by reflexivity, one successful step, and transitivity.

Proposition 3.1 (Fuel is sufficient). *If $T(s) \leq n$, then `runFuel(p, n, s)` is quiescent and is reachable from s . In particular, `run(p, s)` has both properties.*

Proof. Induct on n . For $n = 0$, the premise gives $T(s) = 0$. A successful step would have a smaller natural-number measure, which is impossible, so the step returns `none`. The source proves this equivalence with quiescence in `DA_step_none_iff_quiescent`. For the successor case, an absent step already gives quiescence. If the step succeeds to s' , the decrease gives $T(s') \leq n$ and the induction hypothesis applies. Reachability follows by adjoining the successful step to the induction hypothesis. The final assertion uses $T(s) \leq T(s) + 1$. $\square$

These are the proofs of `runFuel_quiescent` and `runFuel_reachable`. In a run starting from s_0 , no more than N successful proposals are possible because the proposal set starts empty and each success inserts a fresh pair. This counts abstract proposals; it is not a formalized bound on computation time, comparison cost, or memory use. Classical choice makes the run specification noncomputable in Lean despite the finiteness and decidability assumptions.

4. REACHABILITY INVARIANTS AND STABILITY

Four elementary properties organize the consistency and stability proofs. For a state $s = (h, P)$ reachable from s_0 , they say:

- (1) *Holder uniqueness:* if $h(w) = m$ and $h(v) = m$, then $w = v$.
- (2) *Held partners were proposed:* $h(w) = m$ implies $(m, w) \in P$.
- (3) *Proposal order:* if $(m, w) \in P$ and $(m, v) \notin P$, then $w \succ_m v$.
- (4) *No regret for receivers:* if $(m, w) \in P$, then $\neg(m \succ_w^* h(w))$.

Their source names are `HeldUnique`, `HeldWasProposed`, `ProposalOrder`, and `NoRegret`. All hold initially, with the last three vacuous because no proposal or holder exists.

Holder uniqueness is preserved because an active proposer has no holder: holding him at the selected woman cannot duplicate him elsewhere. If the woman retains her old holder, uniqueness follows from the previous state. For held-partner history, a newly accepted proposer has just made the proposal, and every retained holder's earlier proposal persists.

Proposal order is preserved because the new woman is maximal among the proposer's unproposed candidates. Old proposals remain preferred to unproposed alternatives by the previous invariant. The related lemma `men_propose_decreasing` makes explicit that a new proposal is strictly less preferred than every earlier proposal by that man.

For no regret, a receiver retains the proposer or someone she prefers to him at the new step. She also never trades down from a previous holder. Transitivity therefore preserves her comparison with every previous proposer. The lemmas `DA_step_women_only_trade_up` and `women_only_trade_up` express this monotonicity over a step and over reachability. The invariant proofs are separate lemmas, subsequently lifted to `DARreaches` by induction on that relation.

4.1. From holders to a matching. For a reachable state, define $\mu_W = h$. If some woman holds m , let $\mu_M(m)$ be that woman; otherwise put $\mu_M(m) = \perp$. The choice of a held woman is legitimate because holder uniqueness ensures any two such witnesses agree. The source's `terminalMatching` includes both inverse proofs as part of the returned `Matching`. In particular,

$$(3) \quad \mu_M(m) = w \iff h(w) = m.$$

Although named `terminalMatching`, this construction accepts any reachable state, not only a quiescent one.

Theorem 4.1 (Stability at quiescence). *If s is reachable from s_0 and quiescent, its matching is stable. Consequently every profile satisfying the stated assumptions has a stable matching, namely the matching obtained from `run(p, s0)`.*

Proof. Suppose (m, w) blocks the matching. We first show $(m, w) \in P$. If m is unmatched, no woman holds him by (3). Quiescence then forces him to have proposed to every woman, since any remaining candidate would make him active. If m is matched to w_0 , then $(m, w_0) \in P$ by held-partner history. If $(m, w) \notin P$, proposal order gives $w_0 \succ_m w$, contradicting the blocking condition $w \succ_m w_0$ by transitivity and irreflexivity. Thus the desired proposal was made. No regret now contradicts the other blocking condition, $m \succ_w^* h(w)$. The final assertion follows from Proposition 3.1 and the matching construction. $\square$

This argument is `stability_of_quiescent`. It treats unmatched men explicitly, so it does not rely on equal market sizes or on a perfect matching. The stable-existence proof merely supplies a quiescent, reachable witness using `exists_quiescent`.

5. STABLE PARTNERS AND PROPOSING-SIDE OPTIMALITY

The central optimality invariant is stronger than no regret. For every stable matching ν , if a man has proposed to his partner under ν , that woman still holds him in the current state:

$$(4) \quad \text{Stable}(p, \nu) \wedge (m, w) \in P \wedge \nu_M(m) = w \implies h(w) = m.$$

This is `StablePartnerHeld`. It is initially vacuous. Its preservation proof uses holder uniqueness, held-partner history, and proposal order.

Lemma 5.1 (A stable partner cannot be lost). *Property (4) holds in every state reachable from s_0 .*

Proof. Fix a stable matching ν and assume the property holds before the next proposal (a, b) . Only the holder of b changes, so other pairs are unaffected. There are two possible ways for the property to fail.

First suppose the new proposal is to a 's stable partner, $\nu_M(a) = b$, but b retains an old holder q . Because a is free, $q \neq a$, and strict totality gives $q \succ_b a$. If q is unmatched in ν , the pair (q, b) blocks ν . If $\nu_M(q) = v$ and $b \succ_q v$, the same pair blocks ν . The remaining possibility is $v \succ_q b$, with $v \neq b$ by inverse consistency. Since q is currently held by b , he has proposed to b . Proposal order forces him also to have proposed to v . The induction hypothesis says v holds q , contradicting holder uniqueness because b also holds him. Thus a cannot be rejected by b .

Second suppose b already holds her partner m in ν after his earlier proposal, and the new proposer a would displace him. Then $a \succ_b m$. If a is unmatched in ν , (a, b) blocks ν . Otherwise let $\nu_M(a) = v$, where $v \neq b$. If $v \succ_a b$, maximality of the next proposal implies a has already proposed to v . The induction hypothesis would make v hold a , contrary to a being active. Hence strict totality gives $b \succ_a v$, again producing the blocking pair (a, b) . Displacement is impossible.

Thus the property is preserved by one step. Induction over reachability, with the three supporting invariants preserved as well, proves the lemma. $\square$

The source implements the case distinction in `DA_step_preserves_StablePartnerHeld` and then proves `StablePartnerHeld_of_reaches`. Its immediate contrapositive is `no_achievable_rejection`: a proposed pair not currently held cannot be stable-achievable. This includes both immediate rejection and later displacement; the development does not need a separate log of rejection events.

Theorem 5.2 (Conditional optimality). *Let s be any state reachable from s_0 and let its associated matching assign w_0 to m . For every w with $\text{Ach}_p(m, w)$,*

$$(5) \quad w_0 \succ_m w \quad \text{or} \quad w_0 = w.$$

In particular, this holds for the quiescent outcome of the specified run.

Proof. If $(m, w) \in P$, stable achievability and Lemma 5.1 imply $h(w) = m$. Since $h(w_0) = m$, holder uniqueness gives $w = w_0$. If $(m, w) \notin P$, the current holder pair (m, w_0) was proposed and proposal order gives $w_0 \succ_m w$. $\square$

This is `men_optimal`; quiescence is not needed for its matched-man conclusion. The public theorem `GS.Palomar.daMenOptimal` specializes it to the run and keeps the premise that m actually receives w_0 . Calling this theorem “men-optimality” must retain that premise.

Proposition 5.3 (The unmatched case). *If s is reachable and quiescent, and its matching leaves m unmatched, then $\neg \text{Ach}_p(m, w)$ for every $w \in W$.*

Proof. Quiescence forces the unheld man m to have proposed to every woman. No woman holds him. If any such pair were stable-achievable, Lemma 5.1 would force that woman to hold him, a contradiction. $\square$

The internal theorem `no_achievable_if_unmatched` proves precisely this statement. Together with Theorem 5.2, it explains the usual proposing-side optimality interpretation when optional partners are allowed. It is distinct from the conditional statement chosen for the public comparison interface. Neither result asserts uniqueness of stable matchings or strategy-proofness of a reported-preference mechanism.

6. THE FORMAL THEOREM INTERFACE

The development separates a reusable implementation from a compact statement-and-solution interface. `GS/Main.lean` imports the basic, termination, and stability modules. Under `GS.Palomar.Implementation` it proves existence, run quiescence with stability, and conditional matched-man optimality. The definition `GS.Palomar.terminalMatchingRun` then names the actual matching constructed from `run p initialState` and its reachability proof.

`Solution.lean` imports `GS.Main` and exports three theorems. Each has the finite-type and decidable-equality assumptions above and `Nonempty W`; none has a `Nonempty M` binder. Their mathematical contents are:

- (1) `GS.Palomar.galeShapley`: there exists a stable matching.
- (2) `GS.Palomar.daStable`: `terminalMatchingRun p` is stable.
- (3) `GS.Palomar.daMenOptimal`: if that matching assigns w_0 to m , then every stable-achievable w satisfies (5).

The second exported statement does not itself mention the run’s quiescence; the implementation result from which it is derived does. This distinction matters when identifying exactly what has been submitted to a theorem-comparison interface.

The source files have the following responsibilities:

File	Mathematical role
<code>GS/Basic.lean</code>	Profile, optional matching, blocking, state, active men, maximal candidate, proposal step, and interface definitions
<code>GS/Termination.lean</code>	Fresh proposals, measure decrease, reachability, proposal order, receiver improvement, fuelled run, and quiescence
<code>GS/Stability.lean</code>	Holder invariants, matching construction, stability, stable-partner preservation, and matched and unmatched optimality lemmas
<code>GS/Main.lean</code>	Implementation-level results and the named run outcome
<code>Solution.lean</code>	Three proved public interface theorems

`Challenge.lean` is a separate specification module, not a dependency of `Solution`. It repeats the relevant definitions and has four intentional `sorry` occurrences: one in the placeholder definition of `terminalMatchingRun`, and one in each of the three theorem statements. The implemented run is supplied in `GS.Main`; it does not use the placeholder. An audit of the solution must therefore follow `Solution`'s imports rather than equating every hole anywhere in the repository with a missing implementation proof. Conversely, compiling the challenge alone is not proof of these results.

7. REPRODUCIBILITY AND VERIFICATION SCOPE

7.1. Pinned dependencies and reproducible checks. The file `lean-toolchain` selects `leanprover/lean4:v4.35.0-rc2`. The project dependency in `lakefile.toml` requests `Mathlib v4.35.0-rc2`, and `lake-manifest.json` fixes its resolved commit to

`065356127b1dc0016f66b7283ce0ce2c4055aa55`.

The manifest also pins the transitive packages. Reproduction should use the complete pinned repository and manifest, not replace them with the latest `Mathlib` release.

The repository provides `scripts/verify-palomar.sh`. It checks the expected files and comparison names; checks that `Challenge` has exactly four intended holes; scans `Solution` and `GS/` for `sorry`; selects the pinned toolchain; and runs `lake build Challenge Solution`. It also compiles the challenge in a fresh directory with a `Lean/Mathlib` import-source allowlist, checks the environment kinds of the named definitions and theorems, and runs `#print axioms` on the three exported solution theorems. Its axiom test rejects anything outside

`{propext, Classical.choice, Quot.sound}`.

These are propositional extensionality, classical choice, and quotient soundness. They are standard foundations available to classical Lean developments; an axiom audit still needs to identify their actual use rather than calling a classical proof axiom-free [1].

7.2. Evidence available for this note. Fresh inspection on September 30, 2026 verified the main-branch commit and retrieved the pinned source files. Their Git blob hashes were checked against the repository tree. The implementation files in `GS/` and `Solution.lean` contain no `sorry`, `admit`, or local `axiom` declaration in that inspection. The exact assumptions and the proof dependencies described in this note were read from the source.

The pinned `README.md` and `formalization.yaml` report a successful local build and that all three target theorems depend only on `propext`, `Classical.choice`, and `Quot.sound`. Those are repository-reported results, not a fresh kernel run performed while preparing this manuscript. No fresh Lean compilation or axiom audit is claimed here. A source scan is useful evidence about visible placeholders, but cannot replace checking elaborated proof terms and their transitive axiom dependencies with the pinned Lean toolchain.

The mathematical proofs above are explanatory reconstructions of the source architecture. They make the model readable without Lean syntax; they are not a replacement verification system. Likewise, a theorem checker establishes a proposition in the implemented definitions, not that those definitions accurately describe every real admissions market.

7.3. Limits of the formalization. The revision covers finite one-to-one matching with complete strict preferences. It does not cover ties, incomplete lists, explicit unacceptable partners, capacities greater than one, couples, or infinite markets. The main theorems require a nonempty receiving side. Optional partner maps permit unequal side sizes but do not silently add an acceptability condition to stability.

The choice of an active man is classically specified, and the candidate and partner-selection functions are marked `noncomputable`. The development verifies that specified run; it supplies no executable implementation, benchmark, or machine-cost analysis. The step-count argument in Section 3 should not be presented as a verified running-time bound. The three public results also do not include order independence for all possible proposal schedules, receiver-optimality, uniqueness, the lattice of stable matchings, or incentive guarantees.

There is no claim here of independent human review of every proof term, nor of an external registry’s endorsement. The package’s challenge holes and the implemented proofs are deliberately distinguished. The repository’s BSD-3-Clause code license remains separate from the CC BY 4.0 license of this manuscript.

8. PREPARATION PROVENANCE

This exposition was prepared with GPT-6.1 assistance, and its manuscript text is primarily AI-generated. The submitting contributor reports some human understanding of the content. This disclosure does not assert that all listed authors have personally reviewed every formal proof. Source inspection and document checks performed for this note are scoped as stated in Section 7.

The pinned repository metadata separately records GPT-6-Luna via Codex for proof engineering and identifies Arthur Freitas Ramos as the source project’s author and responsible maintainer. That earlier proof-engineering provenance is distinct from the authorship and GPT-6.1-assisted preparation of this exposition. No new Lean proof or repository modification is part of the manuscript preparation.

REFERENCES

1. Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich, *Theorem proving in Lean 4*, Online documentation, chapter “Axioms and Computation”, https://lean-lang.org/theorem_proving_in_lean4/Axioms-and-Computation/. Accessed September 30, 2026.
2. Leonardo de Moura and Sebastian Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction—CADE 28, Lecture Notes in Computer Science, vol. 12699, Springer, 2021, https://doi.org/10.1007/978-3-030-79876-5_37, pp. 625–635.
3. David Gale and Lloyd S. Shapley, *College admissions and the stability of marriage*, The American Mathematical Monthly **69** (1962), no. 1, 9–15, <https://doi.org/10.1080/00029890.1962.11989827>.
4. Arthur Freitas Ramos, *Gale–Shapley deferred acceptance in Lean 4*, Source repository, 2026, Commit f5f34f68a0e38441cdf5632adde5420d5fe0021e, September 26, 2026; inspected September 30, 2026. <https://github.com/Arthur742Ramos/gale-shapley-lean/tree/f5f34f68a0e38441cdf5632adde5420d5fe0021e>.
5. The mathlib Community, *The Lean mathematical library*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, ACM, 2020, <https://doi.org/10.1145/3372885.3373824>, pp. 367–381.