Gale-Shapley Algorithm

Tobias Nipkow
Department of Informatics
Technical University of Munich

October 13, 2025

Abstract

This is a stepwise refinement and proof of the Gale-Shapley stable matching (or marriage) algorithm down to executable code. Both a purely functional implementation based on lists and a functional implementation based on efficient arrays (provided by the Collections entry in the AFP) are developed. The latter implementation runs in time $O(n^2)$ where n is the cardinality of the two sets to be matched.

1 Introduction

The Gale-Shapley algorithm [3, 4] for stable matchings (or marriages) matches two sets of the same cardinality n, where each element has a complete list of preferences (a linear order) of the elements of the other set.

The refinement process is carried out largely on the level of a simple imperative language. In every refinement step the whole algorithm is stated and proved. Most of the proof is abstrtacted into general lemmas that are used in multiple proofs. Except for one bigger step, each algorithm proof is obtained from the previous one by incremental changes. In the end, two executable functional algorithms are obtained: a purely functional one based on lists and a functional one based on a persistent imperative implementation of arrays (provided by the AFP entry Collections Framework [5] based on [1] (see also [2])). The latter algorithm has linear complexity, i.e. $O(n^2)$.

We prove that each of the algorithm computes a stable matching that is optimal for one of the two sets.

2 Part 1: Refinement down to lists

theory *Gale-Shapley1*
imports
HOL−Hoare.Hoare-Logic
List−Index.List-Index
HOL−Library.While-Combinator
HOL−Library.LaTeXsugar
begin

2.1 Misc

lemmas *conj12 = conjunct1 conjunct2*

syntax
-assign-list :: *idt* ⇒ *nat* ⇒ *'b* ⇒ *'com* (‹(*2*-[-] :=/ -)› [*70*, *0*, *65*] *61*)

syntax-consts
-assign-list ⇌ *list-update*

translations
xs[*n*] := *e* ⇀ *xs* := *CONST list-update xs n e*

abbreviation *upt-set* :: *nat* ⇒ *nat set* (‹{<-}›) **where**
{<*n*} ≡ {*0*..<*n*}

definition *prefers* :: *'a list* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
prefers P x y = (*index P x* < *index P y*)

abbreviation *prefa* :: *'a list* ⇒ *'a* ⇒ *'a* ⇒ *bool* (‹(- ⊢/ - < -)› [*50*,*50*,*50*] *50*)
where
P ⊢ *x* < *y* ≡ *prefers P x y*

lemma *prefers-asym*: *P* ⊢ *x* < *y* ⟹ ¬ *P* ⊢ *y* < *x*
by(*simp add*: *prefers-def*)

lemma *prefers-trans*: *P* ⊢ *x* < *y* ⟹ *P* ⊢ *y* < *z* ⟹ *P* ⊢ *x* < *z*
by (*meson order-less-trans prefers-def*)

fun *rk-of-pref* :: *nat* ⇒ *nat list* ⇒ *nat list* ⇒ *nat list* **where**
rk-of-pref r rs (*n*#*ns*) = (*rk-of-pref* (*r*+*1*) *rs ns*)[*n* := *r*] |
rk-of-pref r rs [] = *rs*

definition *ranking* :: *nat list* ⇒ *nat list* **where**
ranking P = *rk-of-pref 0* (*replicate* (*length P*) *0*) *P*

lemma *length-rk-of-pref*[*simp*]: *length*(*rk-of-pref v vs P*) = *length vs*
by(*induction P arbitrary*: *v*)(*auto*)

lemma *nth-rk-of-pref*: ⟦ *length P* ≤ *length rs*; *i* ∈ *set P*; *distinct P*; *set P* ⊆ {<*length rs*} ⟧
⟹ *rk-of-pref r rs P* ! *i* = *index P i* + *r*
by(*induction P arbitrary*: *r i*) (*auto simp add*: *nth-list-update*)

lemma *ranking-index*: ⟦ *length P* = *n*; *set P* = {<*n*} ⟧ ⟹ *ranking P* = *map* (*index P*) [*0*..<*length P*]
by(*simp add*: *list-eq-iff-nth-eq ranking-def card-distinct nth-rk-of-pref*)

lemma *ranking-iff-pref*: ⟦ *set P* = {<*length P*}; *i* < *length P*; *j* < *length P* ⟧
⟹ *ranking P* ! *i* < *ranking P* ! *j* ⟷ *P* ⊢ *i* < *j*
by(*simp add*: *ranking-index prefers-def*)

2.2 Fixing the preference lists

type-synonym *prefs* = *nat list list*

locale *Pref* =
fixes *n*
fixes *P* :: *prefs*
fixes *Q* :: *prefs*
defines *n* ≡ *length P*
assumes *length-Q*: *length Q* = *n*
assumes *P-set*: *a* < *n* ⟹ *length*(*P*!*a*) = *n* ∧ *set*(*P*!*a*) = {<*n*}
assumes *Q-set*: *b* < *n* ⟹ *length*(*Q*!*b*) = *n* ∧ *set*(*Q*!*b*) = {<*n*}
begin

abbreviation *wf* :: *nat list* ⇒ *bool* **where**
wf xs ≡ *length xs* = *n* ∧ *set xs* ⊆ {<*n*}

lemma *wf-less-n*: ⟦ *wf A*; *a* < *n* ⟧ ⟹ *A*!*a* < *n*
by (*simp add*: *subset-eq*)

corollary *wf-le-n1*: ⟦ *wf A*; *a* < *n* ⟧ ⟹ *A*!*a* ≤ *n*−*1*
using *wf-less-n* **by** *fastforce*

lemma *sumA-ub*: *wf A* ⟹ (∑ *a*<*n*. *A*!*a*) ≤ *n*∗(*n*−*1*)
using *sum-bounded-above*[*of* {..<*n*} ((!) *A*) *n*−*1*] *wf-le-n1*[*of A*] **by** (*simp*)

2.3 The (termination) variant(s)

Basic idea: either some $A!a$ is incremented or size of M is incremented, but this cannot go on forever because in the worst case all $A!a = n-1$ and $M = n$. Because $n*(n-1) + n = n\hat{}2$, this leads to the following simple variant:

definition *var0* :: *nat list* ⇒ *nat set* ⇒ *nat* **where**
[*simp*]: *var0 A M* = (*n*^*2* − ((∑ *a*<*n*. *A*!*a*) + *card M*))

lemma *var0-match*:

assumes *wf A M* $\subseteq \{<n\}$ $a < n \wedge a \notin M$
shows *var0 A* $(M \cup \{a\})$ < *var0 A M*
proof −
have *2*: *M* $\subset \{<n\}$ **using** *assms(2−3)* **by** *auto*
have *3*: *card M* < *n* **using** *psubset-card-mono*[*OF - 2*] **by** *simp*
then show *?thesis*
using *sumA-ub*[*OF assms(1)*] *assms(3) finite-subset*[*OF assms(2)*]
by (*simp add*: *power2-eq-square algebra-simps le-diff-conv2*)
qed

lemma *var0-next*:
assumes *wf A M* $\subseteq \{<n\}$ *M* $\neq \{<n\}$ $a' < n$
shows *var0* $(A[a' := A\ !\ a' + 1])$ *M* < *var0 A M*
proof −
have *0*: *card M* < *n* **using** *assms(2,3)*
by (*metis atLeast0LessThan card-lessThan card-subset-eq finite-lessThan lessThan-iff nat-less-le*
subset-eq-atLeast0-lessThan-card)
have ∗: *1* + $(\sum a<n.\ A!a)$ + *card M* ≤ *n*∗*n*
using *sumA-ub*[*OF assms(1)*] *0* **by** (*simp add*: *algebra-simps le-diff-conv2*)
have *var0* $(A[a' := A\ !\ a' + 1])$ *M* = *n*∗*n* − (*1* + (*A* ! *a'* + *sum* ((!) *A*) ({<*n*} − {*a'*})) + *card M*)
using *assms* **by**(*simp add*: *power2-eq-square nth-list-update sum.If-cases lessThan-atLeast0 flip*:*Diff-eq*)
also have ... = *n*^2 − (*1* + $(\sum a<n.\ A!a)$ + *card M*)
using *sum.insert-remove*[*of* {<*n*} *nth A a'*,*simplified*,*symmetric*] *assms(4)*
by (*simp add*:*insert-absorb lessThan-atLeast0 power2-eq-square*)
also have ... < *n*^2 − (($\sum a<n.\ A!a$) + *card M*) **unfolding** *power2-eq-square*
using ∗ **by** *linarith*
finally show *?thesis* **unfolding** *var0-def* **.**
qed

definition *var* :: *nat list* ⇒ *nat set* ⇒ *nat* **where**
[*simp*]: *var A M* = (*n*^2 − *n* + *1* − (($\sum a<n.\ A!a$) + *card M*))

lemma *sumA-ub2*:
assumes $a' < n$ $A!a' \leq n-1$ $\forall\, a < n.\ a \neq a' \longrightarrow A!a \leq n-2$
shows $(\sum a<n.\ A!a) \leq (n-1)*(n-1)$
proof −
have $(\sum a<n.\ A!a) = (\sum a \in (\{<n\}-\{a'\}) \cup \{a'\}.\ A!a)$
by (*simp add*: *assms(1) atLeast0LessThan insert-absorb*)
also have ... =$(\sum a \in \{<n\}-\{a'\}.\ A!a)$ + $A!a'$
by (*simp add*: *sum.insert-remove*)
also have ... ≤ $(\sum a \in \{<n\}-\{a'\}.\ A!a)$ + $(n-1)$ **using** *assms(2)* **by** *linarith*
also have ... ≤ $(n-1)*(n-2)$ + $(n-1)$
using *sum-bounded-above*[*of* {..<*n*}−{*a'*} ((!) *A*) *n−2*] *assms(1,3)*
by (*simp add*: *atLeast0LessThan*)
also have ... = $(n-1)*(n-1)$
by (*metis Suc-diff-Suc Suc-eq-plus1 add.commute diff-is-0-eq' linorder-not-le*

mult-Suc-right mult-cancel-left nat-1-add-1)
finally show *?thesis* **.**
qed

definition *match A a* = *P* ! *a* ! (*A* ! *a*)

lemma *match-less-n*: ⟦ *wf A*; *a* < *n* ⟧ ⟹ *match A a* < *n*
by (*metis P-set atLeastLessThan-iff match-def nth-mem subset-eq*)

lemma *match-upd-neq*: ⟦ *wf A*; *a* < *n*; *a* ≠ *a'* ⟧ ⟹ *match* (*A*[*a* := *b*]) *a'* = *match A a'*
by (*simp add*: *match-def*)

definition *stable* :: *nat list* ⇒ *nat set* ⇒ *bool* **where**
stable A M = (¬(∃ *a*∈*M*. ∃ *a'*∈*M*. *P* ! *a* ⊢ *match A a'* < *match A a* ∧ *Q* ! *match A a'* ⊢ *a* < *a'*))

The set of Bs that an A would prefer to its current match, i.e. all Bs above its current match *A*!*a*.

abbreviation *preferred* **where**
preferred A a == *nth* (*P*!*a*) ‘ {<*A*!*a*}

definition *matching* **where** [*simp*]:
matching A M = (*wf A* ∧ *inj-on* (*match A*) *M*)

If *a'* is unmatched and final then all other *a* are matched:

lemma *final-last*:
assumes *M*: *M* ⊆ {<*n*} **and** *inj*: *inj-on* (*match A*) *M* **and** *pref-match'*: *preferred A a* ⊆ *match A* ‘ *M*
and *a*: *a* < *n* ∧ *a* ∉ *M* **and** *final*: *A* ! *a* + *1* = *n*
shows *insert a M* = {<*n*}
proof −
let *?B* = *preferred A a*
have (!) (*P* ! *a*) ‘ {<*n*} = {<*n*} **by** (*metis P-set a map-nth set-map set-upt*)
hence *inj-on* ((!) (*P* ! *a*)) {<*n*} **by**(*simp add*: *eq-card-imp-inj-on*)
hence *inj-on* ((!) (*P* ! *a*)) {<*A*!*a*} **using** *final* **by**(*simp add*: *inj-on-subset*)
hence *1*: *Suc*(*card ?B*) = *n* **using** *final* **by** (*simp add*: *card-image*)
have *2*: *card ?B* ≤ *card M*
by(*rule surj-card-le*[*OF subset-eq-atLeast0-lessThan-finite*[*OF M*] *pref-match'*])
have *3*: *card M* < *n* **using** *M a*
by (*metis atLeast0LessThan card-seteq order.refl finite-atLeastLessThan le-neq-implies-less lessThan-iff subset-eq-atLeast0-lessThan-card*)
have *Suc* (*card M*) = *n* **using** *1 2 3* **by** *simp*
thus *?thesis* **using** *M a* **by** (*simp add*: *card-subset-eq finite-subset*)
qed

lemma *more-choices*:
assumes *A*: *wf A* **and** *M*: *M* ⊆ {<*n*} *M* ≠ {<*n*}
and *pref-match'*: *preferred A a* ⊆ *match A* ‘ *M*

and $a < n$ **and** *matched*: *match A a* $\in$ *match A ‘ M*
shows *A ! a + 1 < n*
proof (*rule ccontr*)
assume ¬ *A ! a + 1 < n*
hence *A ! a + 1 = n* **using** *A* ‹*a < n*› **unfolding** *matching-def*
by (*metis add.commute wf-less-n linorder-neqE-nat not-less-eq plus-1-eq-Suc*)
hence ∗: *nth* (*P ! a*) ‘ {*<n*} ⊆ *match A ‘ M*
using *pref-match′ matched less-Suc-eq match-def* **by** *fastforce*
have *nth* (*P!a*) ‘ {*<n*} = {*<n*}
using *P-set*[*OF* ‹*a < n*›] **by** (*metis map-nth set-map set-upt*)
hence {*<n*} ⊆ *match A ‘ M* **using** ∗ **by** *metis*
hence *card* {*<n*} ≤ *card M*
using *finite-subset*[*OF* ‹*M* ⊆ {*<n*}› *finite-atLeastLessThan*] **by** (*metis surj-card-le*)
then show *False* **using** *M card-seteq* **by** *blast*
qed

corollary *more-choices-matched*:
assumes *wf A M* ⊆ {*<n*} *M* ≠ {*<n*}
and *preferred A a* ⊆ *match A ‘ M* **and** *a* ∈ *M*
shows *A ! a + 1 < n*
using *more-choices*[*OF assms*(*1−4*)] ‹*a* ∈ *M*› ‹*M* ⊆ {*<n*}› *atLeastLessThan-iff*
by *blast*

lemma *atmost1-final*: **assumes** *M*: *M* ⊆ {*<n*} **and** *inj*: *inj-on* (*match A*) *M*
and ∀ *a*<*n*. *preferred A a* ⊆ *match A ‘ M*
shows ∃$_{\leq 1}$ *a*. *a < n* ∧ *a* ∉ *M* ∧ *A ! a + 1 = n*
apply *rule*
subgoal for *x y*
using *final-last*[*OF M inj*, *of x*] *final-last*[*OF M inj*, *of y*] *assms*(*3*) **by** *blast*
done

lemma *sumA-UB*:
assumes *matching A M M* ⊆ {*<n*} *M* ≠ {*<n*} ∀ *a*<*n*. *preferred A a* ⊆ *match A ‘ M*
shows (∑ *a*<*n*. *A*!*a*) ≤ (*n*−*1*)^*2*
proof −
have *wf A* **using** *assms*(*1*) **by**(*simp*)
have *M*: ∀ *a*∈*M*. *A*!*a* + *1 < n* **using** *more-choices-matched*[*OF* ‹*wf A*› *assms*(*2−3*)] *assms*(*4*)
‹*M* ⊆ {*<n*}› *atLeastLessThan-iff* **by** *blast*
note *Ainj* = *conj12*[*OF assms*(*1*)[*unfolded matching-def*]]
show *?thesis*
proof (*cases* ∃ *a′*<*n*. *a′* ∉ *M* ∧ *A*!*a′* + *1* = *n*)
case *True*
then obtain *a′* **where** *a′*: *a′*<*n a′* ∉ *M A*!*a′* + *1* = *n* **using** ‹*M* ⊆ {*<n*}› ‹*M* ≠ {*<n*}› **by** *blast*
hence ∀ *a*<*n*. *a* ≠ *a′* ⟶ *A*!*a* ≤ *n*−*2*
using *Uniq-D*[*OF atmost1-final*[*OF assms*(*2*) *Ainj*(*2*) *assms*(*4*)], *of a′*] *M wf-le-n1*[*OF Ainj*(*1*)]

by (*metis Suc-1 Suc-eq-plus1 add-diff-cancel-right' add-le-imp-le-diff diff-diff-left less-eq-Suc-le order-less-le*)
from *sumA-ub2*[*OF a'*(*1*) - *this*] *a'*(*3*) **show** *?thesis* **unfolding** *power2-eq-square* **by** *linarith*
next
case *False*
hence $\forall a'<n.\ a' \notin M \longrightarrow A\ !\ a' + 1 < n$
by (*metis Suc-eq-plus1 Suc-lessI wf-less-n*[*OF Ainj*(*1*)])
with *M* **have** $\forall a<n.\ A\ !\ a + 1 < n$ **by** *blast*
hence $(\sum a<n.\ A!a) \leq n*(n-2)$ **using** *sum-bounded-above*[*of* {..<*n*} ((!) *A*) *n*−*2*] **by** *fastforce*
also have $\ldots \leq (n-1)*(n-1)$ **by**(*simp add*: *algebra-simps*)
finally show *?thesis* **unfolding** *power2-eq-square* **.**
qed
qed

lemma *var-ub*:
assumes *matching A M M* $\subseteq$ {<*n*} *M* $\neq$ {<*n*} $\forall a<n.$ *preferred A a* $\subseteq$ *match A* ‘ *M*
shows $(\sum a<n.\ A!a) +$ *card M* $< n\hat{}2 - n + 1$
proof −
have *1*: *M* $\subset$ {<*n*} **using** *assms*(*2*,*3*) **by** *auto*
have *2*: *card M* $< n$ **using** *psubset-card-mono*[*OF - 1*] **by** *simp*
have *3*: *sum* ((!) *A*) {..<*n*} $\leq n\hat{}2 + 1 - 2*n$
using *sumA-UB*[*OF assms*(*1*−*4*)] **by** (*simp add*: *power2-eq-square algebra-simps*)
have *4*: $2*n \leq$ *Suc* $(n\hat{}2)$ **using** *le-square*[*of n*] **unfolding** *power2-eq-square*
by (*metis Suc-1 add-mono-thms-linordered-semiring*(*1*) *le-SucI mult-2 mult-le-mono1 not-less-eq-eq plus-1-eq-Suc*)
show $(\sum a<n.\ A!a) +$ *card M* $< n\hat{}2 - n + 1$ **using** *2 3 4* **by** *linarith*
qed

lemma *var-match*:
assumes *matching A M M* $\subseteq$ {<*n*} *M* $\neq$ {<*n*} $\forall a<n.$ *preferred A a* $\subseteq$ *match A* ‘ *M a* $\notin$ *M*
shows *var A* $(M \cup \{a\})$ < *var A M*
proof −
have *2*: *M* $\subset$ {<*n*} **using** *assms*(*2*,*3*) **by** *auto*
have *3*: *card M* $< n$ **using** *psubset-card-mono*[*OF - 2*] **by** *simp*
have *4*: *sum* ((!) *A*) {..<*n*} $\leq n\hat{}2 + 1 - 2*n$
using *sumA-UB*[*OF assms*(*1*−*4*)] **by** (*simp add*: *power2-eq-square algebra-simps*)
have *5*: $2*n \leq$ *Suc* $(n\hat{}2)$ **using** *le-square*[*of n*] **unfolding** *power2-eq-square*
by (*metis Suc-1 add-mono-thms-linordered-semiring*(*1*) *le-SucI mult-2 mult-le-mono1 not-less-eq-eq plus-1-eq-Suc*)
have *6*: $(\sum a<n.\ A!a) +$ *card M* $< n\hat{}2 + 1 - n$ **using** *3 4 5* **by** *linarith*
from *var-ub*[*OF assms*(*1*−*4*)] **show** *?thesis* **using** ‹*a* $\notin$ *M*› *finite-subset*[*OF assms*(*2*)] **by**(*simp*)
qed

lemma *var-next*:

assumes*matching A M M* $\subseteq$ $\{<n\}$ *M* $\neq$ $\{<n\}$ $\forall$ *a*$<$*n. preferred A a* $\subseteq$ *match A* ‘ *M*
a $<$ *n*
shows *var* (*A*[*a* := *A* ! *a* + *1*]) *M* $<$ *var A M*
proof −
have *var* (*A*[*a* := *A* ! *a* + *1*]) *M* = *n*∗*n* − *n* + *1* − (*1* + (*A* ! *a* + *sum* ((!) *A*) ($\{<n\}$ − {*a*})) + *card M*)
using *assms*(*1*,*5*) **by**(*simp add*: *power2-eq-square nth-list-update sum.If-cases lessThan-atLeast0 flip*:*Diff-eq*)
also have ... = *n*^*2* − *n* + *1* − (*1* + ($\sum$ *a*$<$*n. A*!*a*) + *card M*)
using *sum.insert-remove*[*of* $\{<n\}$ *nth A a*,*simplified*,*symmetric*] *assms*(*5*)
by (*simp add*:*insert-absorb lessThan-atLeast0 power2-eq-square*)
also have ... $<$ *n*^*2* − *n* + *1* − (($\sum$ *a*$<$*n. A*!*a*) + *card M*) **using** *var-ub*[*OF assms*(*1*−*4*)] **unfolding** *power2-eq-square*
by *linarith*
finally show *?thesis* **unfolding** *var-def* **.**
qed

2.4 Auxiliary Predicates

The following two predicates express the same property: if *a* prefers *b* over *a*'s current match, then *b* is matched with an *a*′ that *b* prefers to *a*.

definition *pref-match* **where**
pref-match A M = ($\forall$ *a*$<$*n.* $\forall$ *b*$<$*n. P*!*a* $\vdash$ *b* $<$ *match A a* $\longrightarrow$ ($\exists$ *a*′$\in$*M. b* = *match A a*′ $\wedge$ *Q* ! *b* $\vdash$ *a*′ $<$ *a*))

definition *pref-match*′ **where**
pref-match′ *A M* = ($\forall$ *a*$<$*n.* $\forall$ *b* $\in$ *preferred A a.* $\exists$ *a*′$\in$*M. b* = *match A a*′ $\wedge$ *Q* ! *b* $\vdash$ *a*′ $<$ *a*)

lemma *pref-match*′*-iff*: *wf A* $\Longrightarrow$ *pref-match*′ *A M* = *pref-match A M*
apply (*auto simp add*: *pref-match*′*-def pref-match-def imp-ex prefers-def match-def*)
apply (*smt* (*verit*) *P-set atLeast0LessThan order.strict-trans index-first lessThan-iff linorder-neqE-nat nth-index*)
by (*smt* (*verit*, *best*) *P-set atLeast0LessThan card-atLeastLessThan card-distinct diff-zero in-mono index-nth-id lessThan-iff less-trans nth-mem*)

definition *optiA* **where**
optiA A = ($\nexists$ *A*′. *matching A*′ $\{<n\}$ $\wedge$ *stable A*′ $\{<n\}$ $\wedge$
($\exists$ *a*$<$*n. P* ! *a* $\vdash$ *match A*′ *a* $<$ *match A a*))

definition *pessiB* **where**
pessiB A = ($\nexists$ *A*′. *matching A*′ $\{<n\}$ $\wedge$ *stable A*′ $\{<n\}$ $\wedge$
($\exists$ *a*$<$*n.* $\exists$ *a*′$<$*n. match A a* = *match A*′ *a*′ $\wedge$ *Q* ! *match A a* $\vdash$ *a* $<$ *a*′))

lemma *optiA-pessiB*: **assumes** *optiA A* **shows** *pessiB A*
unfolding *pessiB-def*
proof (*safe*, *goal-cases*)

case (*1 A′ a a′*)
have ¬ *P*!*a* ⊢ *match A a* < *match A′ a* **using** *1*
by (*metis atLeast0LessThan lessThan-iff stable-def*)
with *1* ‹*optiA A*› **show** *?case* **using** *P-set match-less-n optiA-def prefers-def* **unfolding** *matching-def*
by (*metis* (*no-types*) *atLeast0LessThan inj-on-contraD lessThan-iff less-not-refl linorder-neqE-nat nth-index*)
qed

lemma *optiA-inv*:
assumes *A*: *wf A* **and** *a*: *a* < *n* **and** *a′*: *a′* < *n* **and** *same-match*: *match A a′* = *match A a*
and *pref*: *Q* ! *match A a′* ⊢ *a′* < *a* **and** *optiA A*
shows *optiA* (*A*[*a* := *A* ! *a* + *1*])
proof (*unfold optiA-def matching-def*, *rule notI*, *elim exE conjE*)
note *optiA* = ‹*optiA A*›[*unfolded optiA-def matching-def*]
let *?A* = *A*[*a* := *A* ! *a* + *1*]
fix *A′ a″*
assume *a″* < *n* **and** *A′*: *length A′* = *n set A′* ⊆ {<*n*} *stable A′* {<*n*} *inj-on* (*match A′*) {<*n*}
and *pref-a″*: *P* ! *a″* ⊢ *match A′ a″* < *match ?A a″*
show *False*
proof *cases*
assume [*simp*]: *a″* = *a*
have *A*!*a* < *n* **using** *A a* **by**(*simp add*: *subset-eq*)
with *A A′ a pref-a″* **have** *P* ! *a* ⊢ *match A′ a* < *match A a* ∨ *match A′ a* = *match A a*
apply(*auto simp*: *prefers-def match-def*)
by (*smt* (*verit*) *P-set wf-less-n card-atLeastLessThan card-distinct diff-zero index-nth-id*
not-less-eq not-less-less-Suc-eq)
thus *False*
proof
assume *P* ! *a* ⊢ *match A′ a* < *match A a* **thus** *False* **using** *optiA A′* ‹*a* < *n*› **by**(*fastforce*)
next
assume *match A′ a* = *match A a*
have *a* ≠ *a′* **using** *pref a′* **by**(*auto simp*: *prefers-def*)
hence *P* ! *a′* ⊢ *match A′ a* < *match A′ a′* ∧ *Q* ! *match A′ a* ⊢ *a′* < *a* **using** *optiA pref A′ same-match* ‹*match A′ a* = *match A a*› *a a′*
by (*metis P-set atLeast0LessThan match-less-n inj-onD lessThan-iff linorder-neqE-nat nth-index prefers-def*)
thus *False* **using** *a a′* ‹*a* ≠ *a′*› *A′*(*3*) **by** (*metis stable-def atLeastLessThan-iff zero-le*)
qed
next
assume *a″* ≠ *a* **thus** *False* **using** *optiA A′ pref-a″* ‹*a″* < *n*› **by**(*metis match-def nth-list-update-neq*)
qed

qed

lemma *pref-match-stable*:
⟦ *matching A* {<*n*}; *pref-match A* {<*n*} ⟧ ⟹ *stable A* {<*n*}
unfolding *pref-match-def stable-def matching-def*
by (*metis atLeast0LessThan match-less-n inj-onD lessThan-iff prefers-asym*)

2.5 Algorithm 1

definition *invAM* **where**
[*simp*]: *invAM A M* = (*matching A M* ∧ *M* ⊆ {<*n*} ∧ *pref-match A M* ∧ *optiA A*)

lemma *invAM-match*:
⟦ *invAM A M*; *a* < *n* ∧ *a* ∉ *M*; *match A a* ∉ *match A* ' *M* ⟧ ⟹ *invAM A* (*M* ∪ {*a*})
by(*simp add*: *pref-match-def*)

lemma *invAM-swap*:
assumes *invAM A M*
assumes *a*: *a* < *n* ∧ *a* ∉ *M* **and** *a'*: *a'* ∈ *M* ∧ *match A a'* = *match A a* **and** *pref*: *Q* ! *match A a'* ⊢ *a* < *a'*
shows *invAM* (*A*[*a'* := *A*!*a'*+*1*]) (*M* − {*a'*} ∪ {*a*})
proof −
have *A*: *wf A* **and** *M* : *M* ⊆ {<*n*} **and** *inj*: *inj-on* (*match A*) *M* **and** *pref-match*: *pref-match A M*
and *optiA A* **by**(*insert* ‹*invAM A M*›) (*auto*)
have *M* ≠ {<*n*} *a'* < *n a* ≠ *a'* **using** *a' a M* **by** *auto*
have *pref-match'*: *pref-match' A M* **using** *pref-match pref-match'-iff*[*OF A*] **by** *blast*
let *?A* = *A*[*a'* := *A*!*a'*+*1*] **let** *?M* = *M* − {*a'*} ∪ {*a*}
have *neq-a'*: ∀ *x*. *x* ∈ *?M* ⟶ *a'* ≠ *x* **using** ‹*a* ≠ *a'*› **by** *blast*
have ‹*set ?A* ⊆ {<*n*}›
apply(*rule set-update-subsetI*[*OF A*[*THEN conjunct2*]])
using *more-choices*[*OF* - *M* ‹*M* ≠ {<*n*}›] *A inj pref-match' a' subsetD*[*OF M*, *of a'*]
by(*fastforce simp*: *pref-match'-def*)
hence *wf ?A* **using** *A* **by**(*simp*)
moreover have *inj-on* (*match ?A*) *?M* **using** *a a' inj*
by(*simp add*: *match-def inj-on-def*)(*metis Diff-iff insert-iff nth-list-update-neq*)
moreover have *pref-match' ?A ?M* **using** *a a' pref-match' A pref* ‹*a'* < *n*›
apply(*simp add*: *pref-match'-def match-upd-neq neq-a' Ball-def Bex-def image-iff imp-ex nth-list-update less-Suc-eq*
flip: *match-def*)
by (*metis prefers-trans*)
moreover have *optiA ?A* **using** *optiA-inv*[*OF A* ‹*a'* < *n*› - - - ‹*optiA A*›] *a a'*[*THEN conjunct2*] *pref* **by** *auto*
ultimately show *?thesis* **using** *a a' M pref-match'-iff* **by** *auto*
qed

lemma *preferred-subset-match-if-invAM*:
assumes *invAM A M*
shows $\forall\, a{<}n.$ *preferred A a* $\subseteq$ *match A ‘ M* (**is** *?P*)
proof −
have *A*: *wf A* **and** *pref-match*: *pref-match A M* **using** *assms*(*1*) **by** *auto*
note *pref-match′* = *pref-match*[*THEN pref-match′-iff*[*OF A*, *THEN iffD2*]]
thus *pref-match1*: $\forall\, a{<}n.$ *preferred A a* $\subseteq$ *match A ‘ M* **unfolding** *pref-match′-def*
by *blast*
qed

lemma *invAM-next*:
assumes *invAM A M*
assumes *a*: $a < n \wedge a \notin M$ **and** *a′*: $a' \in M \wedge$ *match A a′* = *match A a* **and** *pref*: $\neg$ *Q* ! *match A a′* ⊢ *a* < *a′*
shows *invAM* (*A*[*a* := *A*!*a* + *1*]) *M*
proof −
have *A*: *wf A* **and** *M* : *M* ⊆ {<*n*} **and** *inj*: *inj-on* (*match A*) *M* **and** *pref-match*: *pref-match A M*
and *optiA*: *optiA A* **and** *a′* < *n*
by(*insert* ‹*invAM A M*› *a′*) (*auto*)
hence *pref′*: *Q* ! *match A a′* ⊢ *a′* < *a*
using *pref a a′ Q-set* **unfolding** *prefers-def*
by (*metis match-def match-less-n index-eq-index-conv linorder-less-linear subsetD*)
have *M* ≠ {<*n*} **using** *a* **by** *fastforce*
have *neq-a*: $\forall\, x.\ x{\in}\ M \longrightarrow a \neq x$ **using** *a* **by** *blast*
have *pref-match′*: *pref-match′ A M* **using** *pref-match pref-match′-iff*[*OF A*,*of M*] **by** *blast*
hence $\forall\, a{<}n.$ *preferred A a* $\subseteq$ *match A ‘ M* **unfolding** *pref-match′-def* **by** *blast*
hence *A*!*a* + *1* < *n*
using *more-choices*[*OF* - *M* ‹*M* ≠ {<*n*}›] *A inj a a′* **unfolding** *matching-def*
by (*metis* (*no-types*, *lifting*) *imageI*)
let *?A* = *A*[*a* := *A*!*a*+*1*]
have *wf ?A* **using** *A* ‹*A*!*a* + *1* < *n*› **by**(*simp add*: *set-update-subsetI*)
moreover have *inj-on* (*match ?A*) *M* **using** *a inj*
by(*simp add*: *match-def inj-on-def*) (*metis nth-list-update-neq*)
moreover have *pref-match′ ?A M* **using** *a pref-match′ pref′ A a′ neq-a*
by(*auto simp*: *match-upd-neq pref-match′-def Ball-def Bex-def image-iff nth-list-update imp-ex less-Suc-eq*
simp flip: *match-def*)
moreover have *optiA ?A* **using** *optiA-inv*[*OF A conjunct1*[*OF a*] ‹*a′* < *n*› *conjunct2*[*OF a′*] *pref′ optiA*] **.**
ultimately show *?thesis* **using** *M* **by** (*simp add*: *pref-match′-iff*)
qed

lemma *Gale-Shapley1*: *VARS M A a a′ b*

```
[M = {} ∧ A = replicate n 0]
WHILE M ≠ {<n}
INV { invAM A M }
VAR {var A M}
DO a := (SOME a. a < n ∧ a ∉ M); b := match A a;
 IF b ∉ match A ‘ M
 THEN M := M ∪ {a}
 ELSE a' := (SOME a'. a' ∈ M ∧ match A a' = b);
      IF Q ! match A a' ⊢ a < a'
      THEN A[a'] := A!a'+1; M := M − {a'} ∪ {a}
      ELSE A[a] := A!a+1
      FI
 FI
OD
[matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
   by(auto simp: stable-def optiA-def pref-match-def P-set card-distinct match-def
index-nth-id prefers-def)
next
  case 3 thus ?case using pref-match-stable by auto
next
  case (2 v M A a)
  hence invAM: invAM A M and m: matching A M and M: M ⊆ {<n} M ≠
{<n} and optiA A
    and v: var A M = v by auto
  note Ainj = conj12[OF m[unfolded matching-def]]
  note pref-match1 = preferred-subset-match-if-invAM[OF invAM]
  define a where a = (SOME a. a < n ∧ a ∉ M)
  have a: a < n ∧ a ∉ M unfolding a-def using M
   by (metis (no-types, lifting) atLeastLessThan-iff someI-ex subsetI subset-antisym)
  show ?case (is ?P((SOME a. a < n ∧ a ∉ M))) unfolding a-def[symmetric]
  proof −
    show ?P a (is (?not-matched ⟶ ?THEN) ∧ (¬ ?not-matched ⟶ ?ELSE))
    proof (rule; rule)
      assume ?not-matched
      show ?THEN
      proof(simp only:mem-Collect-eq prod.case, rule conjI, goal-cases)
        case 1 show ?case using invAM-match[OF invAM a ‹?not-matched›] .

        case 2 show ?case
          using var-match[OF m M pref-match1] var0-match[OF Ainj(1) M(1)] a
unfolding v by blast
      qed
    next
      assume matched: ¬ ?not-matched
      define a' where a' = (SOME a'.  a' ∈ M ∧ match A a' = match A a)
     have a': a' ∈ M ∧ match A a' = match A a unfolding a'-def using matched
       by (smt (verit) image-iff someI-ex)
```

```
    hence a' < n a ≠ a' using a M atLeast0LessThan by auto
     show ?ELSE (is ?P((SOME a'. a' ∈ M ∧ match A a' = match A a)))
unfolding a'-def[symmetric]
    proof −
     show ?P a' (is (?pref ⟶ ?THEN) ∧ (¬ ?pref ⟶ ?ELSE))
     proof (rule; rule)
      assume ?pref
      show ?THEN
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
       case 1 show ?case by(rule invAM-swap[OF invAM a a' ‹?pref›])

       case 2
       have card(M − {a'} ∪ {a}) = card M
        using a a' card.remove subset-eq-atLeast0-lessThan-finite[OF M(1)] by
fastforce
       thus ?case using v var-next[OF m M pref-match1 ‹a' < n›] var0-next[OF
Ainj(1) M ‹a' < n›]
         by simp
      qed
     next
      assume ¬ ?pref
      show ?ELSE
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
       case 1 show ?case using invAM-next[OF invAM a a' ‹¬ ?pref›] .

       case 2
       show ?case using a v var-next[OF m M pref-match1, of a] var0-next[OF
Ainj(1) M, of a]
         by simp
      qed
     qed
    qed
   qed
  qed
qed
```

Proof also works for *var0* instead of *var*.

2.6 Algorithm 2: List of unmatched As

```
abbreviation invas where
invas as == (set as ⊆ {<n} ∧ distinct as)

lemma Gale-Shapley2: VARS A a a' as b
 [as = [0..<n] ∧ A = replicate n 0]
 WHILE as ≠ []
 INV { invAM A ({<n} − set as) ∧ invas as}
 VAR {var A ({<n} − set as)}
 DO a := hd as; b := match A a;
  IF b ∉ match A ‘ ({<n} − set as)
```

```
  THEN as := tl as
  ELSE a' := (SOME a'. a' ∈ {<n} − set as ∧ match A a' = b);
       IF Q ! match A a' ⊢ a < a'
       THEN A[a'] := A!a'+1; as := a' # tl as
       ELSE A[a] := A!a+1
       FI
  FI
 OD
 [matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
   by(auto simp: stable-def optiA-def pref-match-def P-set card-distinct match-def
index-nth-id prefers-def)
next
  case 3 thus ?case using pref-match-stable by auto
next
  case (2 v A - a' as)
  let ?M = {<n} − set as
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M
⊆ {<n}
    and as ≠ [] and as: invas as and v: var A ?M = v using 2 by auto
  note pref-match1 = preferred-subset-match-if-invAM[OF invAM]
   from ‹as ≠ []› obtain a as' where aseq: as = a # as' by (fastforce simp:
neq-Nil-conv)
  have set-as: ?M ∪ {a} = {<n} − set as' using as aseq by force
  have a: a < n ∧ a ∉ ?M using as unfolding aseq by (simp)
  show ?case
  proof (simp only: aseq list.sel, goal-cases)
    case 1 show ?case (is (?not-matched ⟶ ?THEN) ∧ (¬ ?not-matched ⟶
?ELSE))
    proof (rule; rule)
      assume ?not-matched
      then have nm: match A a ∉ match A ‘ ?M unfolding aseq .
      show ?THEN
      proof(simp only:mem-Collect-eq prod.case, rule conjI, goal-cases)
        case 1 show ?case using invAM-match[OF invAM a nm] as unfolding
set-as by (simp add: aseq)
       case 2 show ?case
         using var-match[OF m M - pref-match1, of a] a atLeast0LessThan
         unfolding set-as v by blast
      qed
    next
      assume matched: ¬ ?not-matched
      define a' where a' = (SOME a'.  a' ∈ ?M ∧ match A a' = match A a)
      have a': a' ∈ ?M ∧ match A a' = match A a unfolding a'-def aseq using
matched
        by (smt (verit) image-iff someI-ex)
      hence a' < n a ≠ a' using a M atLeast0LessThan by auto
      show ?ELSE unfolding aseq[symmetric] a'-def[symmetric]
```

```
    proof (goal-cases)
      case 1
      show ?case (is (?pref ⟶ ?THEN) ∧ (¬ ?pref ⟶ ?ELSE))
      proof (rule; rule)
        assume ?pref
        show ?THEN
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
          have ∗: {<n} − set as − {a'} ∪ {a} = {<n} − set (a' # as') using a
a' as aseq by auto
        case 1 show ?case using invAM-swap[OF invAM a a' ‹?pref›] unfolding
∗
            using a' as aseq by force
          case 2
          have card({<n} − set as) = card({<n} − set (a' # as')) using a a' as
aseq by auto
           thus ?case using v var-next[OF m M - pref-match1, of a'] ‹a' < n› a
atLeast0LessThan
            by (metis Suc-eq-plus1 lessThan-iff var-def)
        qed
      next
        assume ¬ ?pref
        show ?ELSE
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
         case 1 show ?case using invAM-next[OF ‹invAM A ?M› a a' ‹¬ ?pref›]
as by blast

          case 2
          show ?case using a v var-next[OF m M - pref-match1, of a]
            by (metis Suc-eq-plus1 atLeast0LessThan lessThan-iff)
        qed
      qed
    qed
  qed
qed
```

2.7 Algorithm 3: Record matching of Bs to As

```
abbreviation invAB :: nat list ⇒ (nat ⇒ nat option) ⇒ nat set ⇒ bool where
invAB A B M == (ran B = M ∧ (∀ b a. B b = Some a ⟶ match A a = b))

lemma invAB-swap:
assumes invAB: invAB A B M
assumes a: a < n ∧ a ∉ M and a': a' ∈ M ∧ match A a' = match A a
  and inj-on B (dom B) B(match A a) = Some a'
shows invAB (A[a' := A!a'+1]) (B(match A a := Some a)) (M − {a'} ∪ {a})
proof −
  have ∀ b x. b ≠ match A a ⟶ B b = Some x ⟶ a'≠ x using invAB a' by
blast
```

moreover have $a \neq a'$ **using** $a\ a'$ **by** *auto*
ultimately show *?thesis* **using** *assms* **by**(*simp add*: *ran-map-upd-Some match-def*)
qed

lemma *Gale-Shapley3*: *VARS A B a a' as b*
[*as* = [*0..*<*n*] ∧ *A* = *replicate n 0* ∧ *B* = (λ-. *None*)]
WHILE as ≠ []
INV { *invAM A* ({<*n*} − *set as*) ∧ *invAB A B* ({<*n*} − *set as*) ∧ *invas as*}
VAR {*var A* ({<*n*} − *set as*)}
DO a := *hd as*; *b* := *match A a*;
IF B b = *None*
THEN B := *B*(*b* := *Some a*); *as* := *tl as*
ELSE a' := *the*(*B b*);
IF Q ! *match A a'* ⊢ *a* < *a'*
THEN B := *B*(*b* := *Some a*); *A*[*a'*] := *A*!*a'*+*1*; *as* := *a' # tl as*
ELSE A[*a*] := *A*!*a*+*1*
FI
FI
OD
[*matching A* {<*n*} ∧ *stable A* {<*n*} ∧ *optiA A*]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *stable-def optiA-def pref-match-def P-set card-distinct match-def index-nth-id prefers-def*)
next
case *3* **thus** *?case* **using** *pref-match-stable* **by** *auto*
next
case (*2 v A B - a' as*)
let *?M* = {<*n*} − *set as*
have *invAM*: *invAM A ?M* **and** *m*: *matching A ?M* **and** *A*: *wf A* **and** *M*: *?M* ⊆ {<*n*}
and *as* ≠ [] **and** *as*: *invas as* **and** *invAB*: *invAB A B ?M* **and** *v*: *var A ?M* = *v*
using *2* **by** *auto*
note *pref-match1* = *preferred-subset-match-if-invAM*[*OF invAM*]
from ‹*as* ≠ []› **obtain** *a as'* **where** *aseq*: *as* = *a* # *as'* **by** (*fastforce simp*: *neq-Nil-conv*)
have *set-as*: *?M* ∪ {*a*} = {<*n*} − *set as'* **using** *as aseq* **by** *force*
have *a*: *a* < *n* ∧ *a* ∉ *?M* **using** *as* **unfolding** *aseq* **by** (*simp*)
show *?case*
proof (*simp only*: *aseq list.sel*, *goal-cases*)
case *1* **show** *?case* (**is** (*?not-matched* ⟶ *?THEN*) ∧ (¬ *?not-matched* ⟶ *?ELSE*))
proof (*rule*; *rule*)
assume *?not-matched*
then have *nm*: *match A a* ∉ *match A* ‘ *?M* **using** *invAB* **unfolding** *aseq ran-def*
apply (*clarsimp simp*: *set-eq-iff*) **using** *not-None-eq* **by** *blast*

```
    show ?THEN
    proof(simp only:mem-Collect-eq prod.case, rule conjI, goal-cases)
     have invAM': invAM A ({<n} - set as')
       using invAM-match[OF invAM a nm] unfolding set-as[symmetric] by
simp
     have invAB': invAB A (B(match A a := Some a)) ({<n} - set as')
      using invAB ‹?not-matched› set-as by (simp)
     case 1 show ?case using invAM' as invAB' unfolding set-as aseq
      by (metis distinct.simps(2) insert-subset list.simps(15))
     case 2 show ?case
      using var-match[OF m M - pref-match1, of a] a atLeast0LessThan
      unfolding set-as v by blast
    qed
   next
    assume matched: ¬ ?not-matched
    then obtain a' where a'eq: B(match A a) = Some a' by auto
     have a': a' ∈ ?M ∧ match A a' = match A a unfolding aseq using a'eq
invAB
     by (metis ranI aseq)
    hence a' < n a ≠ a' using a M atLeast0LessThan by auto
    show ?ELSE unfolding aseq[symmetric] a'eq option.sel
    proof (goal-cases)
      have inj-dom: inj-on B (dom B) by (metis (mono-tags) domD inj-onI
invAB)
     case 1
     show ?case (is (?pref ⟶ ?THEN) ∧ (¬ ?pref ⟶ ?ELSE))
     proof (rule, rule)
      assume ?pref
      show ?THEN
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
       have ∗: {<n} - set as - {a'} ∪ {a} = {<n} - set (a' # as') using a
a' as aseq by auto
       have a'neq: ∀ b x. b ≠ match A a ⟶ B b = Some x ⟶ a' ≠ x
        using invAB a' by blast
       have invAB': invAB (A[a' := A ! a' + 1]) (B(match A a := Some a))
({<n} - insert a' (set as'))
        using invAB-swap[OF invAB a a' inj-dom a'eq] ∗ by simp
       case 1 show ?case using invAM-swap[OF invAM a a' ‹?pref›] invAB'
unfolding ∗
        using a' as aseq by simp
       case 2
       have card({<n} - set as) = card({<n} - set (a' # as')) using a a' as
aseq by auto
        thus ?case using v var-next[OF m M - pref-match1, of a'] ‹a' < n› a
atLeast0LessThan
         by (metis Suc-eq-plus1 lessThan-iff var-def)
      qed
     next
      assume ¬ ?pref
```

```
        show ?ELSE
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
          case 1
          have invAB (A[a := A ! a + 1]) B ?M using invAB a
            by (metis match-def nth-list-update-neq ranI)
          thus ?case using invAM-next[OF invAM a a' ‹¬ ?pref›] as by blast
          case 2
          show ?case using a v var-next[OF m M - pref-match1, of a]
            by (metis Suc-eq-plus1 atLeast0LessThan lessThan-iff)
        qed
      qed
    qed
  qed
 qed
qed
```

2.8 Unused data refinement step

```
abbreviation invAB' :: nat list ⇒ nat list ⇒ bool list ⇒ nat set ⇒ bool where
invAB' A B M M' == (length B = n ∧ length M = n ∧ M' = nth B ' {b. b < n
∧ M!b}
 ∧ (∀ b<n. M!b ⟶ B!b < n ∧ match A (B!b) = b))

lemma Gale-Shapley4-unused: VARS A B M a a' as b
 [as = [0..<n] ∧ A = replicate n 0 ∧ B = replicate n 0 ∧ M = replicate n False]
 WHILE as ≠ []
 INV { invAM A ({<n} − set as) ∧ invAB' A B M ({<n} − set as) ∧ invas as}
 VAR {var A ({<n} − set as)}
 DO a := hd as; b := match A a;
  IF ¬ (M ! b)
  THEN M[b] := True; B[b] := a; as := tl as
  ELSE a' := B ! b;
       IF Q ! match A a' ⊢ a < a'
       THEN B[b] := a; A[a'] := A!a'+1; as := a' # tl as
       ELSE A[a] := A!a+1
       FI
  FI
 OD
 [wf A ∧ inj-on (match A) {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
   by(auto simp: stable-def optiA-def pref-match-def P-set card-distinct match-def
index-nth-id prefers-def)
next
  case 3 thus ?case using pref-match-stable by auto
next
  case (2 v A B M - a' as)
  let ?M = {<n} − set as
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M
```

```
⊆ {<n}
    and notall: as ≠ [] and as: invas as and invAB: invAB' A B M ?M and v:
var A ?M = v
   using 2 by auto
  note pref-match1 = preferred-subset-match-if-invAM[OF invAM]
 from notall obtain a as' where aseq: as = a # as' by (fastforce simp: neq-Nil-conv)
  have set-as: ?M ∪ {a} = {<n} − set as' using as aseq by force
  have a: a < n ∧ a ∉ ?M using as unfolding aseq by (simp)
  show ?case
  proof (simp only: aseq list.sel, goal-cases)
    case 1 show ?case (is (?not-matched ⟶ ?THEN) ∧ (¬ ?not-matched ⟶
?ELSE))
   proof (rule; rule)
     assume ?not-matched
     then have nm: match A a ∉ match A ‘ ?M using invAB set-as unfolding
aseq by force
     show ?THEN
     proof(simp only:mem-Collect-eq prod.case, rule conjI, goal-cases)
       have invAM': invAM A ({<n} − set as')
         using invAM-match[OF invAM a nm] unfolding set-as[symmetric] by
simp
      then have invAB': invAB' A (B[match A a := a]) (M[match A a := True])
({<n} − set as')
        using invAB ‹?not-matched› set-as match-less-n[OF A] a
        by (auto simp add: image-def nth-list-update)
       case 1 show ?case using invAM' invAB as invAB' unfolding set-as aseq
        by (metis distinct.simps(2) insert-subset list.simps(15))
       case 2 show ?case
        using var-match[OF m M - pref-match1, of a] a atLeast0LessThan
        unfolding set-as v by blast
     qed
   next
     assume matched: ¬ ?not-matched
   hence match A a ∈ match A ‘ ({<n} − insert a (set as')) using match-less-n[OF
A] a invAB
         apply(auto) by (metis (lifting) image-eqI list.simps(15) mem-Collect-eq
aseq)
     hence Suc(A!a) < n using more-choices[OF A M, of a] a pref-match1
       using aseq atLeast0LessThan by auto
     let ?a = B ! match A a
     have a': ?a ∈ ?M ∧ match A ?a = match A a
       using invAB match-less-n[OF A] matched a by blast
     hence ?a < n a ≠ ?a using a by auto
     show ?ELSE unfolding aseq option.sel
     proof (goal-cases)
       case 1
       show ?case (is (?pref ⟶ ?THEN) ∧ (¬ ?pref ⟶ ?ELSE))
       proof (rule; rule)
         assume ?pref
```

```
        show ?THEN
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
         have ∗: {<n} − set as − {?a} ∪ {a} = {<n} − set (?a # as′) using a
a′ as aseq by auto
          have a′neq: ∀ b<n. b ≠ match A a ⟶ M!b ⟶ ?a ≠ B!b
           using invAB a′ by metis
           have invAB′: invAB′ (A[?a := A ! ?a + 1]) (B[match A a := a]) M
({<n} − set (?a#as′))
           using invAB aseq ∗ ‹a ≠ ?a› a′ match-less-n[OF A, of a] a
           apply (simp add: nth-list-update)
           apply rule
            apply(auto simp add: image-def)[]
           apply (clarsimp simp add: match-def)
           apply (metis (opaque-lifting) nth-list-update-neq)
           done
          case 1 show ?case using invAM-swap[OF invAM a a′ ‹?pref›] invAB′
unfolding ∗
           using a′ as aseq by (auto)
          case 2
         have card({<n} − set as) = card({<n} − set (?a # as′)) using a a′ as
aseq by simp
          thus ?case using v var-next[OF m M - pref-match1, of ?a] ‹?a < n› a
atLeast0LessThan
           by (metis Suc-eq-plus1 lessThan-iff var-def)
        qed
       next
        assume ¬ ?pref
        show ?ELSE
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
         case 1
         have invAB′ (A[a := A ! a + 1]) B M ({<n} − set as) using invAB a
‹a ≠ ?a›
          by (metis match-def nth-list-update-neq)
          thus ?case using invAM-next[OF invAM a a′ ‹¬ ?pref›] as aseq by
fastforce
         case 2
         show ?case using a v var-next[OF m M - pref-match1, of a] aseq
          by (metis Suc-eq-plus1 atLeast0LessThan lessThan-iff)
        qed
       qed
     qed
   qed
  qed
qed
```

2.9 Algorithm 4: remove list of unmatched As

2.9.1 An initial version

The inner variant appears intuitive but complicates the derivation of an overall complexity bound because the inner variant also depends on a variable that is modified by the outer loop.

lemma *Gale-Shapley4*:
VARS A B ai a a′
[*ai = 0 ∧ A = replicate n 0 ∧ B = (λ-. None)*]
WHILE ai < n
INV { invAM A {<ai} ∧ invAB A B {<ai} ∧ ai ≤ n }
VAR {z = n − ai}
DO a := ai;
WHILE B (match A a) ≠ None
INV { invAM A ({<ai+1} − {a}) ∧ invAB A B ({<ai+1} − {a}) ∧ (a ≤ ai ∧ ai < n) ∧ z = n−ai }
VAR {var A {<ai}}
DO a′ := the(B (match A a));
IF Q ! match A a′ ⊢ a < a′
THEN B := B(match A a := Some a); A[a′] := A!a′+1; a := a′
ELSE A[a] := A!a+1
FI
OD;
B := B(match A a := Some a); ai := ai+1
OD
[*matching A {<n} ∧ stable A {<n} ∧ optiA A*]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *stable-def pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def*)[]
next
case *2*
thus *?case* **by** (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
next
case (*4 z A B ai a*)
note *inv* = *4*[*THEN conjunct1*]
note *invAM* = *inv*[*THEN conjunct1*]
note *aai* = *inv*[*THEN conjunct2*,*THEN conjunct2*]
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
case *1*
have ∗: {<*Suc ai*} = *insert a* ({<*Suc ai*} − {*a*}) **using** *aai* **by** (*simp add*: *insert-absorb*)
have ∗∗: *inj-on* (*match A*) {<*Suc ai*} = (*inj-on* (*match A*) ({<*Suc ai*} − {*a*}) ∧ *match A a* ∉ *match A* ‘ ({<*Suc ai*} − {*a*}))
by (*metis* ∗ *Diff-idemp inj-on-insert*)
have *nm*: *match A a* ∉ *match A* ‘ ({<*Suc ai*} − {*a*}) **using** *4* **unfolding**

ran-def
apply (*clarsimp simp*: *set-eq-iff*) **by** (*metis not-None-eq*)
have *invAM'*: *invAM A* {<*ai*+*1*}
using *invAM-match*[*OF invAM*, *of a*] *aai nm* **by** (*simp add*: ∗∗ *insert-absorb*)
show *?case* **using** *4 invAM'* **by** (*simp add*: *insert-absorb*)
next
case *2* **thus** *?case* **using** *4* **by** *auto*
qed
next
case *5*
thus *?case* **using** *pref-match-stable* **unfolding** *invAM-def* **by** (*metis le-neq-implies-less*)
next
case (*3 z v A B ai a a'*)
let *?M* = {<*ai*+*1*} − {*a*}
have *invAM*: *invAM A ?M* **and** *m*: *matching A ?M* **and** *A*: *wf A* **and** *M*: *?M* ⊆ {<*n*}
and *matched*: *B*(*match A a*) ≠ *None* **and** *as*: *a* ≤ *ai* ∧ *ai* < *n* **and** *invAB*: *invAB A B ?M*
and *v*: *var A ?M* = *v* **using** *3* **by** *auto*
note *invar* = *3*[*THEN conjunct1*,*THEN conjunct1*]
note *pref-match1* = *preferred-subset-match-if-invAM*[*OF invAM*]
from *matched* **obtain** *a'* **where** *a'eq*: *B*(*match A a*) = *Some a'* **by** *auto*
have *a'*: *a'* ∈ *?M* ∧ *match A a'* = *match A a* **using** *a'eq invAB* **by** (*metis ranI*)
have *a*: *a* < *n* ∧ *a* ∉ *?M* **using** *invar* **by** *auto*
have *?M* ≠ {<*n*} **and** *a'* < *n* **using** *M a a' atLeast0LessThan* **by** *auto*
have *card*: *card* {<*ai*} = *card ?M* **using** *as* **by** *simp*
show *?case* **unfolding** *a'eq option.sel*
proof (*goal-cases*)
case *1*
show *?case* (**is** (*?unstab* ⟶ *?THEN*) ∧ (¬ *?unstab* ⟶ *?ELSE*))
proof (*rule*; *rule*)
assume *?unstab*
have ∗: {<*ai* + *1*} − {*a*} − {*a'*} ∪ {*a*} = {<*ai* + *1*} − {*a'*} **using** *invar a'* **by** *auto*
show *?THEN*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
have *inj-dom*: *inj-on B* (*dom B*) **by** (*metis* (*mono-tags*) *domD inj-onI invAB*)
have *invAB'*: *invAB* (*A*[*a'* := *A* ! *a'* + *1*]) (*B*(*match A a* ↦ *a*)) ({<*ai* + *1*} − {*a'*})
using *invAB-swap*[*OF invAB a a' inj-dom a'eq*] ∗ **by** *simp*
case *1* **show** *?case*
using *invAM-swap*[*OF invAM a a'* ‹*?unstab*›] *invAB' invar a'* **unfolding** ∗ **by** (*simp*)
next
case *2*
show *?case* **using** *v var-next*[*OF m M* ‹*?M* ≠ {<*n*}› *pref-match1* ‹*a'* < *n*›] *card*
by (*metis var-def Suc-eq-plus1 psubset-eq*)

```
    qed
  next
    assume ¬ ?unstab
    show ?ELSE
    proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
      have ∗: ∀ b a'. B b = Some a' ⟶ a ≠ a' by (metis invAB ranI a)
      case 1 show ?case using invAM-next[OF invAM a a' ‹¬ ?unstab›] invar ∗
by (simp add: match-def)
    next
      case 2
      show ?case using v var-next[OF m M ‹?M ≠ {<n}› pref-match1, of a] a
card
        by (metis Suc-eq-plus1 var-def)
    qed
  qed
 qed
qed
```

2.9.2 A better inner variant

This is the definitive version of Algorithm 4. The inner variant is changed to support the easy derivation of the precise upper bound of the number of executed actions. This variant shows that the algorithm can at most execute $n^2 - n + 1$ basic actions (match, swap, next).

```
definition var2 :: nat list ⇒ nat where
[simp]: var2 A = (n−1)^2 − (∑ a<n. A!a)
```

Because A is not changed by the outer loop, the initial value of *var2 A*, which is $(n - 1)^2$, is an upper bound of the number of iterations of the inner loop. To this we need to add n because the outer loop executes additional n match actions at the end of the loop body. Thus at most $(n - 1)^2 + n = n^2 - n + 1$ actions are executed, exactly as in the earlier algorithms.

```
lemma var2-next:
assumes invAM (A[a := A!a + 1]) M M ≠ {<n} a < n
shows var2 (A[a := A!a + 1]) < var2 A
proof −
  let ?A = A[a := A!a + 1]
  have wf ?A using assms(1) by auto
  have 1: (∑ a<n. ?A!a) = (∑ a<n. A!a) + 1
  proof −
    have (∑ a<n. ?A!a) = 1 + (A ! a + sum ((!) A) ({<n} − {a}))
    using ‹wf ?A› ‹a < n› by(simp add: nth-list-update sum.If-cases lessThan-atLeast0
flip:Diff-eq)
    also have ... = (∑ a<n. A!a) + 1
      using ‹a < n› member-le-sum[of a {<n} nth A] by(simp add: sum-diff1-nat
lessThan-atLeast0)
    finally show ?thesis .
  qed
```

have $(\sum a{<}n.\ ?A!a) \leq (n{-}1)\hat{}2$
using *sumA-UB*[*of ?A M*] *assms*(*1*,*2*) **by** (*meson invAM-def preferred-subset-match-if-invAM*)
with *1* **show** *?thesis* **unfolding** *var2-def* **by** *auto*
qed

lemma *Gale-Shapley4-var2*:
VARS A B ai a a′
[$ai = 0 \wedge A = replicate\ n\ 0 \wedge B = (\lambda\text{-}.\ None)$]
WHILE $ai < n$
INV { $invAM\ A\ \{{<}ai\} \wedge invAB\ A\ B\ \{{<}ai\} \wedge ai \leq n$ }
VAR $\{z = n - ai\}$
DO $a := ai$;
WHILE $B\ (match\ A\ a) \neq None$
INV { $invAM\ A\ (\{{<}ai{+}1\} - \{a\}) \wedge invAB\ A\ B\ (\{{<}ai{+}1\} - \{a\}) \wedge (a \leq ai \wedge ai < n) \wedge z = n{-}ai$ }
VAR $\{var2\ A\}$
DO $a' := the(B\ (match\ A\ a))$;
IF $Q\ !\ match\ A\ a' \vdash a < a'$
THEN $B := B(match\ A\ a := Some\ a);\ A[a'] := A!a'{+}1;\ a := a'$
ELSE $A[a] := A!a{+}1$
FI
OD;
$B := B(match\ A\ a := Some\ a);\ ai := ai{+}1$
OD
[$matching\ A\ \{{<}n\} \wedge stable\ A\ \{{<}n\} \wedge optiA\ A$]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *stable-def pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def*)[]
next
case *2*
thus *?case* **by** (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
next
case (*4 z A B ai a*)
note *inv* = *4*[*THEN conjunct1*]
note *invAM* = *inv*[*THEN conjunct1*]
note *aai* = *inv*[*THEN conjunct2*,*THEN conjunct2*]
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
case *1*
have $*$: $\{{<}Suc\ ai\} = insert\ a\ (\{{<}Suc\ ai\} - \{a\})$ **using** *aai* **by** (*simp add*: *insert-absorb*)
have $**$: $inj\text{-}on\ (match\ A)\ \{{<}Suc\ ai\} = (inj\text{-}on\ (match\ A)\ (\{{<}Suc\ ai\} - \{a\}) \wedge match\ A\ a \notin match\ A\ `\ (\{{<}Suc\ ai\} - \{a\}))$
by (*metis* $*$ *Diff-idemp inj-on-insert*)
have *nm*: $match\ A\ a \notin match\ A\ `\ (\{{<}Suc\ ai\} - \{a\})$ **using** *4* **unfolding** *ran-def*
apply (*clarsimp simp*: *set-eq-iff*) **by** (*metis not-None-eq*)

```
    have invAM': invAM A {<ai+1}
     using invAM-match[OF invAM, of a] aai nm by (simp add: ** insert-absorb)
    show ?case using 4 invAM' by (simp add: insert-absorb)
  next
    case 2 thus ?case using 4 by auto
  qed
next
  case 5
 thus ?case using pref-match-stable unfolding invAM-def by (metis le-neq-implies-less)
next
  case (3 z v A B ai a a')
  let ?M = {<ai+1} - {a}
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M
⊆ {<n}
     and matched: B(match A a) ≠ None and as: a ≤ ai ∧ ai < n and invAB:
invAB A B ?M
    and v: var2 A = v using 3 by auto
  note invar = 3[THEN conjunct1,THEN conjunct1]
  from matched obtain a' where a'eq: B(match A a) = Some a' by auto
  have a': a' ∈ ?M ∧ match A a' = match A a using a'eq invAB by (metis ranI)
  have a: a < n ∧ a ∉ ?M using invar by auto
  have ?M ≠ {<n} and a' < n using M a a' atLeast0LessThan by auto
  show ?case unfolding a'eq option.sel
  proof (goal-cases)
    case 1
    show ?case (is (?unstab ⟶ ?THEN) ∧ (¬ ?unstab ⟶ ?ELSE))
    proof (rule; rule)
      assume ?unstab
      have *: {<ai + 1} - {a} - {a'} ∪ {a} = {<ai + 1} - {a'} using invar a'
by auto
      show ?THEN
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
        note invAM' = invAM-swap[OF invAM a a' ‹?unstab›]
          have inj-dom: inj-on B (dom B) by (metis (mono-tags) domD inj-onI
invAB)
        have invAB': invAB (A[a' := A ! a' + 1]) (B(match A a ↦ a)) ({<ai +
1} - {a'})
          using invAB-swap[OF invAB a a' inj-dom a'eq] * by simp
        case 1 show ?case using invAM' invAB' invar a' unfolding * by (simp)
      case 2 show ?case using v var2-next[OF invAM'] ‹a' < n› * atLeast0LessThan
by auto
      qed
    next
      assume ¬ ?unstab
      show ?ELSE
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
        note invAM' = invAM-next[OF invAM a a' ‹¬ ?unstab›]
        have *: ∀ b a'. B b = Some a' ⟶ a ≠ a' by (metis invAB ranI a)
        case 1 show ?case using invAM' invar * by (simp add: match-def)
```

```
      case 2 show ?case using v var2-next[OF invAM'] a ‹?M ≠ {<n}› by blast
     qed
    qed
  qed
qed
```

2.9.3 Algorithm 4.1: single-loop variant

A bit of a relic because it is an instance of a general program transformation for merging nested loops by an additional test inside the single loop.

```
lemma Gale-Shapley4-1: VARS A B a a' ai b
 [ai = 0 ∧ a = 0 ∧ A = replicate n 0 ∧ B = (λ-. None)]
 WHILE ai < n
 INV { invAM A ({<ai+1} - {a}) ∧ invAB A B ({<ai+1} - {a}) ∧ (a ≤ ai ∧
ai ≤ n) ∧ (ai=n ⟶ a=ai)}
 VAR {var A ({<ai+1} - {a})}
 DO b := match A a;
  IF B b = None
  THEN B := B(b := Some a); ai := ai + 1; a := ai
  ELSE a' := the(B b);
       IF Q ! match A a' ⊢ a < a'
       THEN B := B(b := Some a); A[a'] := A!a'+1; a := a'
       ELSE A[a] := A!a+1
       FI
  FI
 OD
 [matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
    by(auto simp: stable-def optiA-def pref-match-def P-set card-distinct match-def
index-nth-id prefers-def)
next
  case 3 thus ?case using pref-match-stable
    using atLeast0-lessThan-Suc by force
next
  case (2 v A B a a' ai b)
  let ?M = {<ai+1} - {a}
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M
⊆ {<n}
    and as: a ≤ ai ∧ ai < n and invAB: invAB A B ?M and v: var A ?M = v
using 2 by auto
  note invar = 2[THEN conjunct1,THEN conjunct1]
  note pref-match1 = preferred-subset-match-if-invAM[OF invAM]
  have a: a < n ∧ a ∉ ?M using as by (simp)
  show ?case (is (?not-matched ⟶ ?THEN) ∧ (¬ ?not-matched ⟶ ?ELSE))
  proof (rule, rule)
    assume ?not-matched
    then have nm: match A a ∉ match A ‘ ?M using invAB unfolding ran-def
      apply (clarsimp simp: set-eq-iff) using not-None-eq by blast
```

```
    show ?THEN
    proof(simp only:mem-Collect-eq prod.case, rule conjI, goal-cases)
      have ∗: {<ai + 1 + 1} − {ai + 1} = {<ai + 1} by auto
      have ∗∗: {<ai + 1} − {a} ∪ {a} = {<ai + 1} using as by auto
     hence invAM′: invAM A {<ai+1}using invAM-match[OF invAM a nm] by
simp
      have invAB′: invAB A (B(match A a := Some a)) {<ai+1}
        using invAB ‹?not-matched› ∗∗ by (simp)
      case 1 show ?case using invAM′ as invAB′ ∗ by presburger
      case 2 show ?case
        using var-match[OF m M - pref-match1, of a] a atLeast0LessThan ∗ ∗∗
        unfolding v by (metis lessThan-iff)
    qed
  next
    assume matched: ¬ ?not-matched
    then obtain a′ where a′eq: B(match A a) = Some a′ by auto
     have a′: a′ ∈ ?M ∧ match A a′ = match A a using a′eq invAB by (metis
ranI)
    hence a′ < n a ≠ a′ using a M atLeast0LessThan by auto
    show ?ELSE unfolding a′eq option.sel
    proof (goal-cases)
     have inj-dom: inj-on B (dom B) by (metis (mono-tags) domD inj-onI invAB)
      case 1
      show ?case (is (?pref ⟶ ?THEN) ∧ (¬ ?pref ⟶ ?ELSE))
      proof (rule; rule)
        assume ?pref
        show ?THEN
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
          have ∗: {<ai + 1} − {a} − {a′} ∪ {a} = {<ai + 1} − {a′} using a a′
as by auto
          have a′neq: ∀ b x. b ≠ match A a ⟶ B b = Some x ⟶ a′≠ x
            using invAB a′ by blast
           have invAB′: invAB (A[a′ := A ! a′ + 1]) (B(match A a := Some a))
({<ai + 1} − {a′})
            using invAB-swap[OF invAB a a′ inj-dom a′eq] ∗ by simp
          case 1 show ?case using invAM-swap[OF invAM a a′ ‹?pref›] invAB′
unfolding ∗
            using a′ as by simp
         case 2
          have card({<ai + 1} − {a′}) = card({<ai + 1} − {a}) using a a′ as
by auto
           thus ?case using v var-next[OF m M - pref-match1, of a′] ‹a′ < n› a
atLeast0LessThan
            by (metis Suc-eq-plus1 lessThan-iff var-def)
        qed
      next
        assume ¬ ?pref
        show ?ELSE
        proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
```

case *1*
have *invAB* (*A*[*a* := *A* ! *a* + *1*]) *B* *?M* **using** *invAB a*
by (*metis match-def nth-list-update-neq ranI*)
thus *?case* **using** *invAM-next*[*OF invAM a a'* ‹¬ *?pref*›] **using** *2* **by** *presburger*
case *2*
show *?case* **using** *a v var-next*[*OF m M - pref-match1*, *of a*]
by (*metis Suc-eq-plus1 atLeast0LessThan lessThan-iff*)
qed
qed
qed
qed
qed

2.10 Algorithm 5: Data refinement of B

definition $\alpha\ B\ N = (\lambda b.\ \mathit{if}\ b < n \wedge N!b\ \mathit{then}\ \mathit{Some}(B!b)\ \mathit{else}\ \mathit{None})$

lemma *α-Some*[*simp*]: $\alpha\ B\ N\ b = \mathit{Some}\ a \longleftrightarrow b < n \wedge N!b \wedge a = B!b$
by(*auto simp add*: *α-def*)

lemma *αupdate1*: ⟦ ¬ *N* ! *b*; *b* < *length B*; *b* < *length N*; *n* = *length N* ⟧
⟹ *ran*(*α* (*B*[*b* := *a*]) (*N*[*b* := *True*])) = *ran*(*α B N*) ∪ {*a*}
by(*force simp add*: *α-def ran-def nth-list-update*)

lemma *αupdate2*: ⟦ *N*!*b*; *b* < *length B*; *b* < *length N*; *length N* = *n* ⟧
⟹ *α* (*B*[*b* := *a*]) *N* = (*α B N*)(*b* := *Some a*)
by(*force simp add*: *α-def nth-list-update*)

abbreviation *invAB2* :: *nat list* ⇒ *nat list* ⇒ *bool list* ⇒ *nat set* ⇒ *bool* **where**
invAB2 A B N M == (*invAB A* (*α B N*) *M* ∧ (*length B* = *n* ∧ *length N* = *n*))

definition *invar1* **where**
[*simp*]: *invar1 A B N ai* = (*invAM A* {<*ai*} ∧ *invAB2 A B N* {<*ai*} ∧ *ai* ≤ *n*)

definition *invar2* **where**
[*simp*]: *invar2 A B N ai a* ≡ (*invAM A* ({<*ai*+*1*} − {*a*}) ∧ *invAB2 A B N* ({<*ai*+*1*} − {*a*}) ∧ *a* ≤ *ai* ∧ *ai* < *n*)

First, the 'old' version with the more complicated inner variant:

lemma *Gale-Shapley5*:
VARS A B N ai a a'
[*ai* = *0* ∧ *A* = *replicate n 0* ∧ *length B* = *n* ∧ *N* = *replicate n False*]
WHILE ai < *n*
INV { *invar1 A B N ai* }
VAR { *z* = *n* − *ai*}
DO a := *ai*;
WHILE N ! *match A a*
INV { *invar2 A B N ai a* ∧ *z* = *n*−*ai* }

```
  VAR {var A {<ai}}
  DO a' := B ! match A a;
     IF Q ! match A a' ⊢ a < a'
     THEN B[match A a] := a; A[a'] := A!a'+1; a := a'
     ELSE A[a] := A!a+1
     FI
  OD;
  B[match A a] := a; N[match A a] := True; ai := ai+1
 OD
 [matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
  by(auto simp: pref-match-def P-set card-distinct match-def index-nth-id prefers-def
optiA-def α-def cong: conj-cong)
next
  case 2
  thus ?case by (auto simp: atLeastLessThanSuc-atLeastAtMost simp flip: atLeast-
LessThan-eq-atLeastAtMost-diff)
next
  case (4 z A B M ai a)
  note inv = 4[THEN conjunct1, unfolded invar2-def]
  note invAM = inv[THEN conjunct1,THEN conjunct1]
  note aai = inv[THEN conjunct1, THEN conjunct2, THEN conjunct2]
  show ?case
  proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
    case 1
     have ∗: {<Suc ai} = insert a ({<Suc ai} − {a}) using aai by (simp add:
insert-absorb)
    have ∗∗: inj-on (match A) {<Suc ai} = (inj-on (match A) ({<Suc ai} − {a})
∧ match A a ∉ match A ‘ ({<Suc ai} − {a}))
      by (metis ∗ Diff-idemp inj-on-insert)
     have nm: match A a ∉ match A ‘ ({<Suc ai} − {a}) using 4 unfolding
invar2-def ran-def
       apply (clarsimp simp: set-eq-iff) by (metis)
    have invAM': invAM A {<ai+1}
     using invAM-match[OF invAM, of a] aai nm by (simp add: ∗∗ insert-absorb)
   show ?case using 4 invAM' by (simp add: αupdate1 match-less-n insert-absorb
nth-list-update)
  next
    case 2 thus ?case using 4 by auto
  qed
next
  case 5
  thus ?case using pref-match-stable unfolding invAM-def invar1-def by(metis
le-neq-implies-less)
next
  case (3 z v A B N ai a)
  let ?M = {<ai+1} − {a}
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M
```

```
⊆ {<n}
    and matched: N ! match A a and as: a ≤ ai ∧ ai < n and invAB: invAB2 A
B N ?M
    and v: var A {<ai} = v using 3 by auto
  note invar = 3[THEN conjunct1, THEN conjunct1, unfolded invar2-def]
  note pref-match1 = preferred-subset-match-if-invAM[OF invAM]
  let ?a = B ! match A a
  have a: a < n ∧ a ∉ ?M using invar by auto
  have a': ?a ∈ ?M ∧ match A ?a = match A a
    using invAB match-less-n[OF A] a matched by (metis α-Some ranI)
  have ?M ≠ {<n} and ?a < n using M a a' atLeast0LessThan by auto
  have card: card {<ai} = card ?M using as by simp
  have ∗: {<ai + 1} − {a} − {?a} ∪ {a} = {<ai + 1} − {?a} using invar a'
by auto
  show ?case
  proof (simp only: mem-Collect-eq prod.case, goal-cases)
    case 1
    show ?case
    proof ((rule;rule;rule), goal-cases)
      case unstab: 1
      have inj-dom: inj-on (α B N) (dom (α B N)) by (metis (mono-tags) domD
inj-onI invAB)
      have invAB': invAB (A[B ! match A a := A ! ?a + 1]) (α (B[match A a :=
a]) N) ({<ai + 1} − {?a})
      using invAB-swap[OF invAB[THEN conjunct1] a a' inj-dom] ∗ match-less-n[OF
A] a matched invAB
        by(simp add:αupdate2)
      show ?case using invAM-swap[OF invAM a a' unstab] invAB' invar a'
        unfolding ∗ by (simp add: insert-absorb αupdate2)

      case 2
      show ?case using v var-next[OF m M ‹?M ≠ {<n}› pref-match1 ‹?a < n›]
card
        by (metis var-def Suc-eq-plus1)
    next
      case stab: 3
      have ∗: ∀ b. b < n ∧ N!b ⟶ a ≠ B!b by (metis invAB ranI α-Some a)
      show ?case using invAM-next[OF invAM a a' stab] invar ∗ by (simp add:
match-def)

      case 4
      show ?case using v var-next[OF m M ‹?M ≠ {<n}› pref-match1, of a] a
card
        by (metis Suc-eq-plus1 var-def)
    qed
  qed
qed
```

The definitive version with variant *var2*:

lemma *Gale-Shapley5-var2*:
VARS A B N ai a a′
[ai = 0 ∧ A = replicate n 0 ∧ length B = n ∧ N = replicate n False]
WHILE ai < n
INV { invar1 A B N ai }
VAR { z = n − ai}
DO a := ai;
WHILE N ! match A a
INV { invar2 A B N ai a ∧ z = n−ai }
VAR {var2 A}
DO a′ := B ! match A a;
IF Q ! match A a′ ⊢ a < a′
THEN B[match A a] := a; A[a′] := A!a′+1; a := a′
ELSE A[a] := A!a+1
FI
OD;
B[match A a] := a; N[match A a] := True; ai := ai+1
OD
[matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def α-def cong*: *conj-cong*)
next
case *2*
thus *?case* **by** (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeast-LessThan-eq-atLeastAtMost-diff*)
next
case (*4 z A B N ai a*)
note *inv* = *4*[*THEN conjunct1*, *unfolded invar2-def*]
note *invAM* = *inv*[*THEN conjunct1*,*THEN conjunct1*]
note *aai* = *inv*[*THEN conjunct1*, *THEN conjunct2*, *THEN conjunct2*]
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
case *1*
have ∗: {<*Suc ai*} = *insert a* ({<*Suc ai*} − {*a*}) **using** *aai* **by** (*simp add*: *insert-absorb*)
have ∗∗: *inj-on* (*match A*) {<*Suc ai*} = (*inj-on* (*match A*) ({<*Suc ai*} − {*a*}) ∧ *match A a* ∉ *match A* ‘ ({<*Suc ai*} − {*a*}))
by (*metis* ∗ *Diff-idemp inj-on-insert*)
have *nm*: *match A a* ∉ *match A* ‘ ({<*Suc ai*} − {*a*}) **using** *4* **unfolding** *invar2-def ran-def*
apply (*clarsimp simp*: *set-eq-iff*) **by** (*metis*)
have *invAM′*: *invAM A* {<*ai+1*}
using *invAM-match*[*OF invAM*, *of a*] *aai nm* **by** (*simp add*: ∗∗ *insert-absorb*)
show *?case* **using** *4 invAM′* **by** (*simp add*: *αupdate1 match-less-n insert-absorb nth-list-update*)
next
case *2* **thus** *?case* **using** *4* **by** *auto*

```
  qed
next
  case 5
  thus ?case using pref-match-stable unfolding invAM-def invar1-def by(metis
le-neq-implies-less)
next
  case (3 z v A B N ai a)
  let ?M = {<ai+1} - {a}
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M
⊆ {<n}
    and matched: N ! match A a and as: a ≤ ai ∧ ai < n and invAB: invAB2 A
B N ?M
    and v: var2 A = v using 3 by auto
  note invar = 3[THEN conjunct1, THEN conjunct1, unfolded invar2-def]
  let ?a = B ! match A a
  have a: a < n ∧ a ∉ ?M using invar by auto
  have a': ?a ∈ ?M ∧ match A ?a = match A a
    using invAB match-less-n[OF A] a matched by (metis α-Some ranI)
  have ?M ≠ {<n} and ?a < n using M a a' atLeast0LessThan by auto
  have card: card {<ai} = card ?M using as by simp
  have ∗: {<ai + 1} - {a} - {?a} ∪ {a} = {<ai + 1} - {?a} using invar a'
by auto
  show ?case
  proof (simp only: mem-Collect-eq prod.case, goal-cases)
    case 1
    show ?case
    proof ((rule;rule;rule), goal-cases)
      case unstab: 1
      note invAM' = invAM-swap[OF invAM a a' unstab]
      have inj-dom: inj-on (α B N) (dom (α B N)) by (metis (mono-tags) domD
inj-onI invAB)
      have invAB': invAB (A[B ! match A a := A ! ?a + 1]) (α (B[match A a :=
a]) N) ({<ai + 1} - {?a})
      using invAB-swap[OF invAB[THEN conjunct1] a a' inj-dom] ∗ match-less-n[OF
A] a matched invAB
        by(simp add:αupdate2)
        show ?case using invAM' invAB' invar a' unfolding ∗ by (simp add:
insert-absorb αupdate2)


      case 2
        show ?case using v var2-next[OF invAM'] ∗ M ‹?a < n› a' by (metis
subset-Diff-insert)
    next
      case stab: 3
      note invAM' = invAM-next[OF invAM a a' stab]
      have ∀ b. b < n ∧ N!b ⟶ a ≠ B!b by (metis invAB ranI α-Some a)
      thus ?case using invAM' invar by (simp add: match-def)
```

```
    case 4
    show ?case using v var2-next[OF invAM'] a ‹?M ≠ {<n}› by blast
   qed
  qed
qed
```

2.10.1 Algorithm 5.1: single-loop variant

```
definition invar2' where
[simp]: invar2' A B N ai a ≡ (invAM A ({<ai+1} − {a}) ∧ invAB2 A B N ({<ai+1} − {a}) ∧ a ≤ ai ∧ ai ≤ n)

lemma pres2':
assumes invar2' A B N ai a ai < n var A ({<ai + 1} − {a}) = v
and after[simp]: b = match A a a' = B ! b A1 = A[a' := A ! a' + 1] A2 = A[a := A ! a + 1]
shows
  (¬ N ! b ⟶
    invar2' A (B[b := a]) (N[b := True]) (ai + 1) (ai + 1) ∧ var A ({<ai + 1 + 1} − {ai + 1}) < v) ∧
   (N ! b ⟶
    (Q ! match A a' ⊢ a < a' ⟶ invar2' A1 (B[b := a]) N ai a' ∧ var A1 ({<ai + 1} − {a'}) < v) ∧
    (¬ Q ! match A a' ⊢ a < a' ⟶ invar2' A2 B N ai a ∧ var A2 ({<ai + 1} − {a}) < v))
proof −
  let ?M = {<ai+1} − {a}
  have invAM: invAM A ?M and m: matching A ?M and A: wf A and M: ?M ⊆ {<n}
    and v: var A ?M = v and as: a ≤ ai ∧ ai < n and invAB: invAB2 A B N ?M
    using assms by auto
  note invar = assms(1)
  note pref-match1 = preferred-subset-match-if-invAM[OF invAM]
  have a: a < n ∧ a ∉ ?M using as by (simp)
  show ?thesis (is (¬ ?matched ⟶ ?THEN) ∧ (?matched ⟶ ?ELSE))
  proof (rule; rule)
    assume ¬ ?matched
    then have nm: match A a ∉ match A ‘ ?M using invAB unfolding ran-def
      apply (clarsimp simp: set-eq-iff) by metis
    show ?THEN
    proof(rule conjI, goal-cases)
      have ∗: {<ai + 1 + 1} − {ai + 1} = {<ai + 1} by auto
      have ∗∗: {<ai + 1} − {a} ∪ {a} = {<ai + 1} using as by auto
       hence invAM': invAM A {<ai+1} using invAM-match[OF invAM a nm]
by simp
       have invAB': invAB2 A (B[match A a := a]) (N[match A a := True]) {<ai+1}
       using invAB ‹¬ ?matched› ∗∗
```

```
      by (simp add: A a αupdate1 match-less-n nth-list-update)
    case 1 show ?case using invAM' as invAB' *
      by (simp add: Suc-le-eq plus-1-eq-Suc)
    case 2 show ?case
      using var-match[OF m M - pref-match1, of a] a atLeast0LessThan * **
      unfolding v by (metis lessThan-iff)
  qed
 next
  assume matched: ?matched
  let ?a = B ! match A a
  have a': ?a ∈ ?M ∧ match A ?a = match A a
   using invAB match-less-n[OF A] a matched after by (metis α-Some ranI)
  hence ?a < n a ≠ ?a using a M atLeast0LessThan by auto
   have inj-dom: inj-on (α B N) (dom (α B N)) by (metis (mono-tags) domD
inj-onI invAB)
  show ?ELSE (is (?pref ⟶ ?THEN) ∧ (¬ ?pref ⟶ ?ELSE))
  proof (rule; rule)
   assume ?pref
   show ?THEN
   proof (rule conjI, goal-cases)
    have *: {<ai + 1} - {a} - {?a} ∪ {a} = {<ai + 1} - {?a} using a a'
as by auto
    have a'neq: ∀ b<n. b ≠ match A a ⟶ N!b ⟶ ?a ≠ B!b
     using invAB a' by force
    have invAB': invAB (A[B ! match A a := A ! ?a + 1]) (α (B[match A a
:= a]) N) ({<ai + 1} - {?a})
     using invAB-swap[OF invAB[THEN conjunct1] a a' inj-dom] * match-less-n[OF
A] a matched invAB
      by(simp add:αupdate2)
      case 1 show ?case using invAM-swap[OF invAM a a'] ‹?pref› invAB
invAB' unfolding *
      using a' as by simp
    case 2
    have card({<ai + 1} - {?a}) = card({<ai + 1} - {a}) using a a' as by
auto
     thus ?case using v var-next[OF m M - pref-match1, of ?a] ‹?a < n› a
atLeast0LessThan
      by (metis Suc-eq-plus1 lessThan-iff var-def after)
   qed
  next
   assume ¬ ?pref
   show ?ELSE
   proof (rule conjI, goal-cases)
    case 1
    have invAB2 (A[a := A ! a + 1]) B N ?M using invAB a
     by (metis match-def nth-list-update-neq ranI)
    thus ?case using invAM-next[OF invAM a a'] ‹¬ ?pref› invar after
     by (meson invar2'-def)
    case 2
```

```
      show ?case using a v var-next[OF m M - pref-match1, of a] after
        by (metis Suc-eq-plus1 atLeast0LessThan lessThan-iff)
    qed
  qed
 qed
qed

lemma Gale-Shapley5-1: VARS A B N a a' ai b
 [ai = 0 ∧ a = 0 ∧ A = replicate n 0 ∧ length B = n ∧ N = replicate n False]
 WHILE ai < n
 INV { invar2' A B N ai a }
 VAR {var A ({<ai+1} − {a})}
 DO b := match A a;
  IF ¬ N ! b
  THEN B[b] := a; N[b] := True; ai := ai + 1; a := ai
  ELSE a' := B ! b;
       IF Q ! match A a' ⊢ a < a'
       THEN B[b] := a; A[a'] := A!a'+1; a := a'
       ELSE A[a] := A!a+1
       FI
  FI
 OD
 [matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
  by(auto simp: pref-match-def P-set card-distinct match-def index-nth-id prefers-def
optiA-def α-def cong: conj-cong)
next
  case 3 thus ?case using pref-match-stable
    using atLeast0-lessThan-Suc by force
next
  case (2 v A B N a a' ai b)
  thus ?case using pres2'
    by (simp only:mem-Collect-eq prod.case) blast
qed
```

2.11 Algorithm 6: replace Q by ranking R

```
lemma inner-to-outer:
assumes inv: invar2 A B N ai a ∧ b = match A a and not-b: ¬ N ! b
shows invar1 A (B[b := a]) (N[b := True]) (ai+1)
proof −
  note invAM = inv[unfolded invar2-def, THEN conjunct1,THEN conjunct1]
   have ∗: {<Suc ai} = insert a ({<Suc ai} − {a}) using inv by (simp add:
insert-absorb)
  have ∗∗: inj-on (match A) {<Suc ai} = (inj-on (match A) ({<Suc ai} − {a})
∧ match A a ∉ match A ‘ ({<Suc ai} − {a}))
    by (metis ∗ Diff-idemp inj-on-insert)
  have nm: match A a ∉ match A ‘ ({<Suc ai} − {a}) using inv not-b unfolding
```

invar2-def ran-def
apply (*clarsimp simp*: *set-eq-iff*) **by** (*metis*)
have *invAM'*: *invAM A* {<*ai*+*1*}
using *invAM-match*[*OF invAM*, *of a*] *inv nm* **by** (*simp add*: ∗∗ *insert-absorb*)
show *?thesis* **using** *inv not-b invAM' match-less-n* **by** (*clarsimp simp*: *αupdate1 insert-absorb nth-list-update*)
qed

lemma *inner-pres*:
assumes *R*: ∀ *b*<*n*. ∀ *a1*<*n*. ∀ *a2*<*n*. *R* ! *b* ! *a1* < *R* ! *b* ! *a2* ⟷ *Q* ! *b* ⊢ *a1* < *a2* **and**
inv: *invar2 A B N ai a* **and** *m*: *N* ! *b* **and** *v*: *var A* {<*ai*} = *v*
and *after*: *A1* = *A*[*a'* := *A* ! *a'* + *1*] *A2* = *A*[*a* := *A* ! *a* + *1*]
a' = *B*!*b r* = *R* ! *match A a' b* = *match A a*
shows (*r* ! *a* < *r* ! *a'* ⟶ *invar2 A1* (*B*[*b*:=*a*]) *N ai a'* ∧ *var A1* {<*ai*} < *v*) ∧
(¬ *r* ! *a* < *r* ! *a'* ⟶ *invar2 A2 B N ai a* ∧ *var A2* {<*ai*} < *v*)
proof −
let *?M* = {<*ai*+*1*} − {*a*}
note [*simp*] = *after*
note *inv'* = *inv*[*unfolded invar2-def*]
have *A*: *wf A* **and** *M*: *?M* ⊆ {<*n*} **and** *invAM*: *invAM A ?M* **and** *invAB*: *invAB A* (*α B N*) *?M*
and *mat*: *matching A ?M* **and** *as*: *a* ≤ *ai* ∧ *ai* < *n* **using** *inv'* **by** *auto*
note *pref-match1* = *preferred-subset-match-if-invAM*[*OF invAM*]
let *?a* = *B* ! *match A a*
have *a*: *a* < *n* ∧ *a* ∉ *?M* **using** *inv* **by** *auto*
have *a'*: *?a* ∈ *?M* ∧ *match A ?a* = *match A a*
using *invAB match-less-n*[*OF A*] *a m inv* **by** (*metis α-Some ranI* ‹*b* = -›)
have *?M* ≠ {<*n*} **and** *?a* < *n* **using** *M a a' atLeast0LessThan* **by** *auto*
have *card*: *card* {<*ai*} = *card ?M* **using** *as* **by** *simp*
show *?thesis*
proof ((*rule*;*rule*;*rule*), *goal-cases*)
have ∗: {<*ai* + *1*} − {*a*} − {*?a*} ∪ {*a*} = {<*ai* + *1*} − {*?a*} **using** *inv a'* **by** *auto*
case *1*
hence *unstab*: *Q* ! *match A a'* ⊢ *a* < *a'*
using *R a a' as Q-set P-set match-less-n*[*OF A*] *n-def length-Q R* **by** (*simp*)
have *inj-dom*: *inj-on* (*α B N*) (*dom* (*α B N*)) **by** (*metis* (*mono-tags*) *domD inj-onI invAB*)
have *invAB'*: *invAB A1* (*α* (*B*[*match A a* := *a*]) *N*) ({<*ai* + *1*} − {*?a*})
using *invAB-swap*[*OF invAB a a' inj-dom*] ∗ *match-less-n*[*OF A*] *a m*
by (*simp add*: *αupdate2 inv'*)
show *?case* **using** *invAM-swap*[*OF invAM a a'*] *unstab invAB' inv a'*
unfolding ∗ **by** (*simp*)
next
case *2*
show *?case* **using** *v var-next*[*OF mat M* ‹*?M* ≠ {<*n*}› *pref-match1* ‹*?a* < *n*›] *card assms*(*5*,*7*,*9*)
by (*metis Suc-eq-plus1 var-def*)

```
  next
    have ∗: ∀ b. b < n ∧ N!b ⟶ a ≠ B!b by (metis invAB ranI α-Some a)
    case 3
    hence unstab: ¬ Q ! match A a' ⊢ a < a'
      using R a a' as Q-set P-set match-less-n[OF A] n-def length-Q
      by (simp add: ranking-iff-pref)
    then show ?case using invAM-next[OF invAM a a'] 3 inv ∗ by (simp add:
match-def)
  next
    case 4
      show ?case using v var-next[OF mat M ‹?M ≠ {<n}› pref-match1, of a] a
card assms(6)
        by (metis Suc-eq-plus1 var-def)
  qed
qed
```

First, the 'old' version with the more complicated inner variant:

```
lemma Gale-Shapley6:
assumes R = map ranking Q
shows
VARS A B N ai a a' b r
 [ai = 0 ∧ A = replicate n 0 ∧ length B = n ∧ N = replicate n False]
 WHILE ai < n
 INV { invar1 A B N ai }
 VAR {z = n − ai}
 DO a := ai; b := match A a;
  WHILE N ! b
  INV { invar2 A B N ai a ∧ b = match A a ∧ z = n−ai }
  VAR {var A {<ai}}
  DO a' := B ! b; r := R ! match A a';
     IF r ! a < r ! a'
     THEN B[b] := a; A[a'] := A!a'+1; a := a'
     ELSE A[a] := A!a+1
     FI;
     b := match A a
  OD;
  B[b] := a; N[b] := True; ai := ai+1
 OD
 [matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
  case 1 thus ?case
  by(auto simp: stable-def pref-match-def P-set card-distinct match-def index-nth-id
prefers-def optiA-def α-def cong: conj-cong)
next
  case 2
  thus ?case by (auto simp: atLeastLessThanSuc-atLeastAtMost simp flip: atLeast-
LessThan-eq-atLeastAtMost-diff)
next
  case 3
```

```
  have R: ∀ b<n. ∀ a1<n. ∀ a2<n. R ! b ! a1 < R ! b ! a2 ⟷ Q ! b ⊢ a1 < a2
    by (simp add: Q-set ‹R = ->› length-Q ranking-iff-pref)
  show ?case
  proof (simp only: mem-Collect-eq prod.case, goal-cases)
    case 1 show ?case using inner-pres[OF R - - refl refl refl] 3 by blast
  qed
next
  case 4
  show ?case
  proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
    case 1 show ?case using 4 inner-to-outer by blast
  next
    case 2 thus ?case using 4 by auto
  qed
next
  case 5
  thus ?case using pref-match-stable unfolding invAM-def invar1-def by(metis
le-neq-implies-less)
qed

lemma inner-pres-var2:
assumes R: ∀ b<n. ∀ a1<n. ∀ a2<n. R ! b ! a1 < R ! b ! a2 ⟷ Q ! b ⊢ a1 <
a2 and
  inv: invar2 A B N ai a and m: N ! b and v: var2 A = v
  and after: A1 = A[a' := A ! a' + 1] A2 = A[a := A ! a + 1]
    a' = B!b r = R ! match A a' b = match A a
shows (r ! a < r ! a' ⟶ invar2 A1 (B[b:=a]) N ai a' ∧ var2 A1 < v) ∧
  (¬ r ! a < r ! a' ⟶ invar2 A2 B N ai a ∧ var2 A2 < v)
proof −
  let ?M = {<ai+1} − {a}
  note [simp] = after
  note inv' = inv[unfolded invar2-def]
   have A: wf A and M: ?M ⊆ {<n} and invAM: invAM A ?M and invAB:
invAB A (α B N) ?M
    and mat: matching A ?M and as: a ≤ ai ∧ ai < n using inv' by auto
  let ?a = B ! match A a
  have a: a < n ∧ a ∉ ?M using inv by auto
  have a': ?a ∈ ?M ∧ match A ?a = match A a
    using invAB match-less-n[OF A] a m inv by (metis α-Some ranI ‹b = ->)
  have ?M ≠ {<n} and ?a < n using M a a' atLeast0LessThan by auto
  have card: card {<ai} = card ?M using as by simp
  show ?thesis
  proof ((rule;rule;rule), goal-cases)
    have ∗: {<ai + 1} − {a} − {?a} ∪ {a} = {<ai + 1} − {?a} using inv a'
by auto
    note invAM' =  invAM-swap[OF invAM a a']
    case 1
    hence unstab: Q ! match A a' ⊢ a < a'
      using R a a' as Q-set P-set match-less-n[OF A] n-def length-Q R by (simp)
```

```
    have inj-dom: inj-on (α B N) (dom (α B N)) by (metis (mono-tags) domD
inj-onI invAB)
    have invAB': invAB A1 (α (B[match A a := a]) N) ({<ai + 1} - {?a})
      using invAB-swap[OF invAB a a' inj-dom] * match-less-n[OF A] a m
      by (simp add: αupdate2 inv')
    show ?case using invAM' unstab invAB' inv a' unfolding * by (simp)

    case 2
      show ?case using v var2-next[OF invAM'] assms(5,7,9) * M ‹?a < n› a'
        by (metis subset-Diff-insert unstab)
  next
    have *: ∀ b. b < n ∧ N!b ⟶ a ≠ B!b by (metis invAB ranI α-Some a)
    note invAM' = invAM-next[OF invAM a a']
    case 3
    hence unstab: ¬ Q ! match A a' ⊢ a < a'
      using R a a' as Q-set P-set match-less-n[OF A] n-def length-Q
      by (simp add: ranking-iff-pref)
    then show ?case using invAM' 3 inv * by (simp add: match-def)

    case 4
      show ?case using v var2-next[OF invAM'] a assms(6,7,9) ‹?M ≠ {<n}›
unstab by fastforce
  qed
qed
```

The definitive version with variant *var2*:

```
lemma Gale-Shapley6-var2:
assumes R = map ranking Q
shows
VARS A B N ai a a' b r
 [ai = 0 ∧ A = replicate n 0 ∧ length B = n ∧ N = replicate n False]
 WHILE ai < n
 INV { invar1 A B N ai }
 VAR {z = n - ai}
 DO a := ai; b := match A a;
  WHILE N ! b
  INV { invar2 A B N ai a ∧ b = match A a ∧ z = n-ai }
  VAR {var2 A}
  DO a' := B ! b; r := R ! match A a';
    IF r ! a < r ! a'
    THEN B[b] := a; A[a'] := A!a'+1; a := a'
    ELSE A[a] := A!a+1
    FI;
    b := match A a
  OD;
  B[b] := a; N[b] := True; ai := ai+1
 OD
 [matching A {<n} ∧ stable A {<n} ∧ optiA A]
proof (vcg-tc, goal-cases)
```

case *1* **thus** *?case*
by(*auto simp*: *stable-def pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)
next
case *2*
thus *?case* **by** (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
next
case *3*
have *R*: $\forall\, b{<}n.\ \forall\, a1{<}n.\ \forall\, a2{<}n.\ R\ !\ b\ !\ a1\ <\ R\ !\ b\ !\ a2 \longleftrightarrow Q\ !\ b \vdash a1\ <\ a2$
by (*simp add*: *Q-set* ‹*R* = -› *length-Q ranking-iff-pref*)
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *goal-cases*)
case *1* **show** *?case* **using** *inner-pres-var2*[*OF R - - refl refl refl*] *3* **by** *blast*
qed
next
case *4*
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
case *1* **show** *?case* **using** *4 inner-to-outer* **by** *blast*
next
case *2* **thus** *?case* **using** *4* **by** *auto*
qed
next
case *5*
thus *?case* **using** *pref-match-stable* **unfolding** *invAM-def invar1-def* **by**(*metis le-neq-implies-less*)
qed

A less precise version where the inner variant does not depend on variables changed in the outer loop. Thus the inner variant is an upper bound on the number of executions of the inner loop test/body. Superseded by the *var2* version.

lemma *var0-next2*:
assumes *wf* $(A[a' := A\ !\ a' + 1])$ $a' < n$
shows *var0* $(A[a' := A\ !\ a' + 1])$ $\{{<}n\}$ < *var0 A* $\{{<}n\}$
proof −
let *?A* = $A[a' := A\ !\ a' + 1]$
have *0*: *card* $\{{<}n\} = n$ **by** *simp*
have ∗: $(\sum a{<}n.\ ?A!a) + card\ \{{<}n\} \leq n\hat{}2$
using *sumA-ub*[*OF assms*(*1*)] *0* **by** (*simp add*: *power2-eq-square algebra-simps le-diff-conv2*)
have $(\sum a{<}n.\ A!a) < (\sum a{<}n.\ ?A!a)$
using *assms sum.remove*[*of* $\{{<}n\}$ a' (!) A]
by(*simp add*: *nth-list-update sum.If-cases lessThan-atLeast0 Diff-eq*)
thus *?thesis* **using** ∗ **unfolding** *var0-def* **by** *linarith*
qed

lemma *inner-pres2*:
assumes *R*: $\forall\, b{<}n.\ \forall\, a1{<}n.\ \forall\, a2{<}n.\ R\ !\ b\ !\ a1 < R\ !\ b\ !\ a2 \longleftrightarrow Q\ !\ b \vdash a1 < a2$ **and**
inv: *invar2 A B N ai a* **and** *m*: *N ! b* **and** *v*: *var0 A* $\{{<}n\} = v$
and *after*: $A1 = A[a' := A\ !\ a' + 1]$ $A2 = A[a := A\ !\ a + 1]$
$a' = B!b$ $r = R\ !$ *match A a' b* = *match A a*
shows $(r\ !\ a < r\ !\ a' \longrightarrow$ *invar2 A1* $(B[b{:=}a])$ *N ai a'* $\wedge$ *var0 A1* $\{{<}n\} < v) \wedge$
$(\neg\ r\ !\ a < r\ !\ a' \longrightarrow$ *invar2 A2 B N ai a* $\wedge$ *var0 A2* $\{{<}n\} < v)$
proof −
let *?M* $= \{{<}ai{+}1\} - \{a\}$
note [*simp*] = *after*
note *inv'* = *inv*[*unfolded invar2-def*]
have *A*: *wf A* **and** *M*: *?M* $\subseteq \{{<}n\}$ **and** *invAM*: *invAM A ?M* **and** *invAB*: *invAB A* $(\alpha\ B\ N)$ *?M*
and *mat*: *matching A ?M*
and *as*: $a \le ai \wedge ai < n$ **using** *inv'* **by** *auto*
note *pref-match1* = *preferred-subset-match-if-invAM*[*OF invAM*]
let *?a* = *B ! match A a*
have *a*: $a < n \wedge a \notin$ *?M* **using** *inv* **by** *auto*
have *a'*: *?a* $\in$ *?M* $\wedge$ *match A ?a* = *match A a*
using *invAB match-less-n*[*OF A*] *a m inv* **by** (*metis* α*-Some ranI* ‹*b* = -›)
have *?M* $\neq \{{<}n\}$ **and** *?a* $< n$ **using** *M a a' atLeast0LessThan* **by** *auto*
have *card*: *card* $\{{<}ai\}$ = *card ?M* **using** *as* **by** *simp*
show *?thesis*
proof ((*rule*;*rule*;*rule*), *goal-cases*)
have $*$: $\{{<}ai + 1\} - \{a\} - \{?a\} \cup \{a\} = \{{<}ai + 1\} - \{?a\}$ **using** *inv a'* **by** *auto*
case *1*
hence *unstab*: *Q ! match A a'* $\vdash a < a'$
using *R a a' as Q-set P-set match-less-n*[*OF A*] *n-def length-Q R* **by** (*simp*)
have *inj-dom*: *inj-on* $(\alpha\ B\ N)$ (*dom* $(\alpha\ B\ N)$) **by** (*metis* (*mono-tags*) *domD inj-onI invAB*)
have *invAB'*: *invAB A1* $(\alpha\ (B[match\ A\ a := a])\ N)$ $(\{{<}ai + 1\} - \{?a\})$
using *invAB-swap*[*OF invAB a a' inj-dom*] $*$ *match-less-n*[*OF A*] *a m*
by (*simp add*: α*update2 inv'*)
show *?case* **using** *invAM-swap*[*OF invAM a a'*] *unstab invAB' inv a'*
unfolding $*$ **by** (*simp add*: *insert-absorb* α*update2*)
next
case *2*
hence *unstab*: *Q ! match A a'* $\vdash a < a'$
using *R a a' as Q-set P-set match-less-n*[*OF A*] *n-def length-Q R* **by** (*simp*)
from *invAM-swap*[*OF invAM a a'*] *unstab* **have** *wf*: *wf* $(A[a' := A\ !\ a' + 1])$ **by** *auto*
show *?case* **using** *v var0-next2*[*OF wf*] **using** ‹*B ! match A a* $< n$› *assms*(5,7,9) **by** *blast*
next
have $*$: $\forall\, b.\ b < n \wedge N!b \longrightarrow a \neq B!b$ **by** (*metis invAB ranI* α*-Some a*)
case *3*
hence *unstab*: $\neg$ *Q ! match A a'* $\vdash a < a'$

using $R\ a\ a'$ *as Q-set P-set match-less-n*[*OF A*] *n-def length-Q*
by (*simp add*: *ranking-iff-pref*)
then show *?case* **using** *invAM-next*[*OF invAM a a'*] *3 inv* $*$ **by** (*simp add*: *match-def*)
next
case *4*
hence *unstab*: $\neg\ Q\ !\ match\ A\ a' \vdash a < a'$
using $R\ a\ a'$ *as Q-set P-set match-less-n*[*OF A*] *n-def length-Q*
by (*simp add*: *ranking-iff-pref*)
from *invAM-next*[*OF invAM a a'*] *unstab* **have** *wf*: *wf* $(A[a := A\ !\ a + 1])$ **by** *auto*
show *?case* **using** *v var0-next2*[*OF wf*] *a* **using** *assms*(*6*) **by** *presburger*
qed
qed

lemma *Gale-Shapley6'*:
assumes $R = map\ ranking\ Q$
shows
VARS A B N ai a a' b r
$[ai = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False]$
WHILE $ai < n$
INV { *invar1 A B N ai* }
VAR $\{z = n - ai\}$
DO $a := ai;\ b := match\ A\ a;$
WHILE $N\ !\ b$
INV { *invar2 A B N ai* $a \wedge b = match\ A\ a \wedge z = n - ai$ }
VAR {*var0 A* $\{<n\}$}
DO $a' := B\ !\ b;\ r := R\ !\ match\ A\ a';$
IF $r\ !\ a < r\ !\ a'$
THEN $B[b] := a;\ A[a'] := A!a'+1;\ a := a'$
ELSE $A[a] := A!a+1$
FI;
$b := match\ A\ a$
OD;
$B[b] := a;\ N[b] := True;\ ai := ai+1$
OD
$[matching\ A\ \{<n\} \wedge stable\ A\ \{<n\} \wedge optiA\ A]$
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *stable-def pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)
next
case *2*
thus *?case* **by** (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
next
case *3*
have *R*: $\forall\ b<n.\ \forall\ a1<n.\ \forall\ a2<n.\ R\ !\ b\ !\ a1 < R\ !\ b\ !\ a2 \longleftrightarrow Q\ !\ b \vdash a1 < a2$
by (*simp add*: *Q-set* ‹$R = \text{-}\rightarrow$› *length-Q ranking-iff-pref*)

show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *goal-cases*)
case *1* **show** *?case* **using** *inner-pres2*[*OF R - - refl refl refl*] *3* **by** *blast*
qed
next
case *4*
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
case *1* **show** *?case* **using** *4 inner-to-outer* **by** *blast*
next
case *2* **thus** *?case* **using** *4* **by** *auto*
qed
next
case *5*
thus *?case* **using** *pref-match-stable* **unfolding** *invAM-def invar1-def* **by**(*metis le-neq-implies-less*)
qed

2.11.1 Algorithm 6.1: single-loop variant

lemma *R-iff-P*:
assumes $R = map\ ranking\ Q\ invar2'\ A\ B\ N\ ai\ a\ ai < n\ N\ !\ match\ A\ a$
shows $(R\ !\ match\ A\ (B\ !\ match\ A\ a)\ !\ a < R\ !\ match\ A\ (B\ !\ match\ A\ a)\ !\ (B\ !\ match\ A\ a)) =$
$(Q\ !\ match\ A\ (B\ !\ match\ A\ a) \vdash a < B\ !\ match\ A\ a)$
proof −
have *R*: $\forall\, b{<}n.\ \forall\, a1{<}n.\ \forall\, a2{<}n.\ R\ !\ b\ !\ a1 < R\ !\ b\ !\ a2 \longleftrightarrow Q\ !\ b \vdash a1 < a2$
by (*simp add*: *Q-set* ‹*R* = -› *length-Q ranking-iff-pref*)
let *?M* = $\{{<}ai{+}1\} - \{a\}$
have *A*: *wf A* **and** *M*: $?M \subseteq \{{<}n\}$ **and** *as*: $a < n$ **and** *invAB*: *invAB2 A B N ?M*
using *assms*(*2*,*3*) **by** *auto*
have *a'*: $B\ !\ match\ A\ a \in ?M$
using *invAB match-less-n*[*OF A*] *as* ‹*N*!*match A a*› **by** (*metis α-Some ranI*)
hence $B\ !\ match\ A\ a < n$ **using** *M* **by** *auto*
thus *?thesis* **using** *assms R match-less-n* **by** *auto*
qed

lemma *Gale-Shapley6-1*:
assumes $R = map\ ranking\ Q$
shows *VARS A B N a a' ai b r*
[$ai = 0 \wedge a = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False$]
WHILE $ai < n$
INV { *invar2' A B N ai a* }
VAR {*var A* ($\{{<}ai{+}1\} - \{a\}$)}
DO $b := match\ A\ a$;
IF $\neg\ N\ !\ b$
THEN $B[b] := a$; $N[b] := True$; $ai := ai + 1$; $a := ai$

ELSE $a' := B\ !\ b$; $r := R\ !\ match\ A\ a'$;
IF $r\ !\ a < r\ !\ a'$
THEN $B[b] := a$; $A[a'] := A!a'+1$; $a := a'$
ELSE $A[a] := A!a+1$
FI
FI
OD
[*matching A* $\{<n\} \wedge$ *stable A* $\{<n\} \wedge$ *optiA A*]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)
next
case *3* **thus** *?case* **using** *pref-match-stable atLeast0-lessThan-Suc* **by** *force*
next
case (*2 v A B N a a' ai*)
have *R'*: $N\ !\ match\ A\ a \Longrightarrow$
$(R\ !\ match\ A\ (B\ !\ match\ A\ a)\ !\ a < R\ !\ match\ A\ (B\ !\ match\ A\ a)\ !\ (B\ !\ match\ A\ a)) =$
$(Q\ !\ match\ A\ (B\ !\ match\ A\ a) \vdash a < B\ !\ match\ A\ a)$
using *R-iff-P 2 assms* **by** *blast*
show *?case*
apply(*simp only*:*mem-Collect-eq prod.case*)
using *2 R' pres2'*[*of A B N ai a*] **by** *presburger*
qed

lemma *Gale-Shapley6-1-explicit*:
assumes $R = map\ ranking\ Q$
shows *VARS A B N a a' ai b r*
[$ai = 0 \wedge a = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False$]
WHILE $ai < n$
INV { *invar2' A B N ai a* }
VAR {*var A* ({$<ai+1$} − {a})}
DO $b := match\ A\ a$;
IF $\neg\ N\ !\ b$
THEN $B[b] := a$; $N[b] := True$; $ai := ai + 1$; $a := ai$
ELSE $a' := B\ !\ b$; $r := R\ !\ match\ A\ a'$;
IF $r\ !\ a < r\ !\ a'$
THEN $B[b] := a$; $A[a'] := A!a'+1$; $a := a'$
ELSE $A[a] := A!a+1$
FI
FI
OD
[*matching A* $\{<n\} \wedge$ *stable A* $\{<n\} \wedge$ *optiA A*]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)

next
case *3* **thus** *?case* **using** *pref-match-stable atLeast0-lessThan-Suc* **by** *force*
next
case (*2 v A B N a a' ai b*)
let *?M* = {<*ai+1*} − {*a*}
have *invAM*: *invAM A ?M* **and** *m*: *matching A ?M* **and** *A*: *wf A* **and** *M*: *?M* ⊆ {<*n*}
and *pref-match*: *pref-match A ?M*
and *v*: *var A ?M* = *v* **and** *as*: *a* ≤ *ai* ∧ *ai* < *n* **and** *invAB*: *invAB2 A B N ?M*
using *2* **by** *auto*
note *invar* = *2*[*THEN conjunct1*,*THEN conjunct1*]
note *pref-match'* = *pref-match*[*THEN pref-match'-iff*[*OF A*, *THEN iffD2*]]
hence *pref-match1*: ∀ *a*<*n*. *preferred A a* ⊆ *match A ‘ ?M* **unfolding** *pref-match'-def* **by** *blast*
have *a*: *a* < *n* ∧ *a* ∉ *?M* **using** *as* **by** (*simp*)
show *?case* (**is** (*?not-matched* ⟶ *?THEN*) ∧ (¬ *?not-matched* ⟶ *?ELSE*))
proof (*rule*; *rule*)
assume *?not-matched*
then have *nm*: *match A a* ∉ *match A ‘ ?M* **using** *invAB* **unfolding** *ran-def*
apply (*clarsimp simp*: *set-eq-iff*) **by** *metis*
show *?THEN*
proof(*simp only*:*mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
have ∗: {<*ai* + *1* + *1*} − {*ai* + *1*} = {<*ai* + *1*} **by** *auto*
have ∗∗: {<*ai* + *1*} − {*a*} ∪ {*a*} = {<*ai* + *1*} **using** *as* **by** *auto*
hence *invAM'*: *invAM A* {<*ai+1*} **using** *invAM-match*[*OF invAM a nm*] **by** *simp*
have *invAB'*: *invAB2 A* (*B*[*match A a* := *a*]) (*N*[*match A a* := *True*]) {<*ai+1*}
using *invAB* ‹*?not-matched*› ∗∗
by (*simp add*: *A a αupdate1 match-less-n nth-list-update*)
case *1* **show** *?case* **using** *invAM' as invAB'* ∗
by (*simp add*: *Suc-le-eq plus-1-eq-Suc*)
case *2* **show** *?case*
using *var-match*[*OF m M - pref-match1*, *of a*] *a atLeast0LessThan* ∗ ∗∗
unfolding *v* **by** (*metis lessThan-iff*)
qed
next
assume *matched*: ¬ *?not-matched*
let *?a* = *B* ! *match A a*
have *a'*: *?a* ∈ *?M* ∧ *match A ?a* = *match A a*
using *invAB match-less-n*[*OF A*] *a matched* **by** (*metis α-Some ranI*)
hence *?a* < *n a* ≠ *?a* **using** *a M atLeast0LessThan* **by** *auto*
have *inj-dom*: *inj-on* (*α B N*) (*dom* (*α B N*)) **by** (*metis* (*mono-tags*) *domD inj-onI invAB*)
show *?ELSE* (**is** (*?pref* ⟶ *?THEN*) ∧ (¬ *?pref* ⟶ *?ELSE*))
proof (*rule*; *rule*)
assume *?pref*
show *?THEN*

```
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
        have *: {<ai + 1} - {a} - {?a} ∪ {a} = {<ai + 1} - {?a} using a a'
as by auto
        have a'neq: ∀ b<n. b ≠ match A a ⟶ N!b ⟶ ?a ≠ B!b
          using invAB a' by force
        have invAB': invAB (A[B ! match A a := A ! ?a + 1]) (α (B[match A a
:= a]) N) ({<ai + 1} - {?a})
       using invAB-swap[OF invAB[THEN conjunct1] a a' inj-dom] * match-less-n[OF
A] a matched invAB
          by(simp add:αupdate2)
        have pref: Q ! match A ?a ⊢ a < ?a using A Q-set ‹?a < n› ‹?pref› a
assms length-Q
          by(auto simp: match-less-n ranking-iff-pref)
        case 1 show ?case
         using invAM-swap[OF invAM a a' pref] invAB invAB' a' as unfolding *
          by (simp add: match-less-n ranking-iff-pref)
        case 2
       have card({<ai + 1} - {?a}) = card({<ai + 1} - {a}) using a a' as by
auto
         thus ?case using v var-next[OF m M - pref-match1, of ?a] ‹?a < n› a
atLeast0LessThan
          by (metis Suc-eq-plus1 lessThan-iff var-def)
      qed
    next
      assume ¬ ?pref
      show ?ELSE
      proof (simp only: mem-Collect-eq prod.case, rule conjI, goal-cases)
        case 1
        have invAB2 (A[a := A ! a + 1]) B N ?M using invAB a
          by (metis match-def nth-list-update-neq ranI)
        thus ?case using invAM-next[OF invAM a a'] ‹¬ ?pref› ‹B ! match A a
< n› Q-set 2 assms
          by (simp add: invar2'-def length-Q match-less-n ranking-iff-pref)
        case 2
        show ?case using a v var-next[OF m M - pref-match1, of a]
          by (metis Suc-eq-plus1 atLeast0LessThan lessThan-iff)
      qed
    qed
  qed
qed

end
```

2.12 Functional implementation

definition
gs-inner P R N =
while (λ(A,B,a,b). N!b)
(λ(A,B,a,b).

```
    let a' = B ! b;
        r = R ! (P ! a' ! (A ! a')) in
    let (A, B, a) =
      if r ! a < r ! a'
      then (A[a' := A!a' + 1], B[b := a], a')
      else (A[a := A!a + 1], B, a)
    in (A, B, a, P ! a ! (A ! a)))

definition
gs n P R =
  while (λ(A,B,N,ai). ai < n)
   (λ(A,B,N,ai).
     let (A,B,a,b) = gs-inner P R N (A, B, ai, P ! ai ! (A ! ai))
     in (A, B[b:=a], N[b:=True], ai+1))
  (replicate n 0, replicate n 0, replicate n False,0)

definition
gs1 n P R =
  while (λ(A,B,N,ai,a). ai < n)
   (λ(A,B,N,ai,a).
     let b = P ! a ! (A ! a) in
     if ¬ N ! b
     then (A, B[b := a], N[b := True], ai+1, ai+1)
     else let a' = B ! b; r = R ! (P ! a' ! (A ! a')) in
       if r ! a < r ! a'
       then (A[a' := A!a'+1], B[b := a], N, ai, a')
       else (A[a := A!a+1], B, N, ai, a))
  (replicate n 0, replicate n 0, replicate n False, 0, 0)

context Pref
begin

lemma gs-inner:
assumes R = map ranking Q
assumes invar2 A B N ai a b = match A a
shows gs-inner P R N (A, B, a, b) = (A',B',a',b') ⟶ invar1 A' (B'[b' := a'])
(N[b' := True]) (ai+1)
unfolding gs-inner-def
proof(rule while-rule2[where P = λ(A,B,a,b). invar2 A B N ai a ∧ b = match
A a
 and r = measure (%(A, B, a, b). Pref.var P A {<ai})], goal-cases)
  case 1
  show ?case using assms unfolding var-def by simp
next
  case inv: (2 s)
  obtain A B a b where s: s =  (A, B, a, b)
    using prod-cases4 by blast
  have R: ∀ b<n. ∀ a1<n. ∀ a2<n. R ! b ! a1 < R ! b ! a2 ⟷ Q ! b ⊢ a1 < a2
    by (simp add: Q-set ‹R = ->  length-Q ranking-iff-pref)
```

```
  show ?case
  proof(rule, goal-cases)
    case 1 show ?case
    using inv apply(simp only: s prod.case Let-def split: if-split)
    using inner-pres[OF R - - refl refl refl refl refl, of A B N ai a b]
    unfolding invar2-def match-def by presburger
    case 2 show ?case
    using inv apply(simp only: s prod.case Let-def in-measure split: if-split)
    using inner-pres[OF R - - refl refl refl refl refl, of A B N ai a b]
    unfolding invar2-def match-def by presburger
  qed
next
  case 3
  show ?case
  proof (rule, goal-cases)
    case 1 show ?case by(rule inner-to-outer[OF 3[unfolded 1 prod.case]])
  qed
next
  case 4
  show ?case by simp
qed

lemma gs: assumes R = map ranking Q
shows gs n P R = (A,BNai) ⟶ matching A {<n} ∧ stable A {<n} ∧ optiA A
unfolding gs-def
proof(rule while-rule2[where P = λ(A,B,N,ai). invar1 A B N ai
  and r = measure(λ(A,B,N,ai). n − ai)], goal-cases)
  case 1 show ?case
  by(auto simp: stable-def pref-match-def P-set card-distinct match-def index-nth-id
prefers-def optiA-def α-def cong: conj-cong)
next
  case (2 s)
  obtain A B N ai where s: s = (A, B, N, ai)
    using prod-cases4 by blast
  have 1: invar2 A B N ai ai using 2 s
   by (auto simp: atLeastLessThanSuc-atLeastAtMost simp flip: atLeastLessThan-eq-atLeastAtMost-diff)
  show ?case using 2 s gs-inner[OF ‹R = - › 1] by (auto simp: match-def simp
del: invar1-def split: prod.split)
next
  case 3
  thus ?case using pref-match-stable by auto
next
  case 4
  show ?case by simp
qed


lemma gs1: assumes R = map ranking Q
shows gs1 n P R = (A,BNaia) ⟶ matching A {<n} ∧ stable A {<n} ∧ optiA A
```

unfolding *gs1-def*
proof(*rule while-rule2*[**where** $P = \lambda(A,B,N,ai,a).$ *invar2'* A B N *ai a*
and $r =$ *measure* $(\%(A, B, N, ai, a).$ *Pref.var P A* $(\{<ai+1\} - \{a\}))]$, *goal-cases*)
case *1* **show** *?case*
by(*auto simp*: *stable-def pref-match-def P-set card-distinct match-def index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)
next
case (*2 s*)
obtain A B N *ai a* **where** *s*: $s = (A, B, N, ai, a)$
using *prod-cases5* **by** *blast*
hence *1*: *invar2'* A B N *ai a* $ai < n$ **using** *2* **by** (*simp-all*)
have R': N ! *match* A $a \Longrightarrow$
$(R$! *match* A $(B$! *match* A $a)$! $a < R$! *match* A $(B$! *match* A $a)$! $(B$! *match* A $a)) =$
$(Q$! *match* A $(B$! *match* A $a) \vdash a < B$! *match* A $a)$
using *R-iff-P*[*OF assms 1*] **by** *linarith*
show *?case*
using *1* R' *pres2'*[*OF 1*]
apply(*simp only*: *s mem-Collect-eq prod.case Let-def in-measure match-def split*: *if-split*)
by *blast*
next
case (*3 s*)
obtain B N *ai a* **where** *BNaia* = (B, N, ai, a)
using *prod-cases4* **by** *blast*
with *3* **show** *?case*
using *pref-match-stable atLeast0-lessThan-Suc* **by** *force*
next
case *4*
show *?case* **by** *simp*
qed

end

2.13 Executable functional Code

definition
Gale-Shapley P Q = (*if Pref P Q then Some* (*fst* (*gs* (*length P*) *P* (*map ranking Q*))) *else None*)

theorem *gs*: $[\![$ *Pref P Q*; $n =$ *length P* $]\!] \Longrightarrow$
$\exists A.$ *Gale-Shapley P Q* = *Some*$(A) \wedge$ *Pref.matching P A* $\{<n\} \wedge$
Pref.stable P Q A $\{<n\} \wedge$ *Pref.optiA P Q A*
unfolding *Gale-Shapley-def* **using** *Pref.gs*
by (*metis fst-conv surj-pair*)

definition
Gale-Shapley1 P Q = (*if Pref P Q then Some* (*fst* (*gs1* (*length P*) *P* (*map ranking*

Q))) *else None*)

theorem *gs1*: ⟦ *Pref P Q*; *n* = *length P* ⟧ ⟹
∃ *A*. *Gale-Shapley1 P Q* = *Some*(*A*) ∧ *Pref.matching P A* {<*n*} ∧
Pref.stable P Q A {<*n*} ∧ *Pref.optiA P Q A*
unfolding *Gale-Shapley1-def* **using** *Pref.gs1*
by (*metis fst-conv surj-pair*)

declare *Pref-def* [*code*]

Two examples from Gusfield and Irving:

lemma *Gale-Shapley*
[[*3*,*0*,*1*,*2*], [*1*,*2*,*0*,*3*], [*1*,*3*,*2*,*0*], [*2*,*0*,*3*,*1*]]
[[*3*,*0*,*2*,*1*], [*0*,*2*,*1*,*3*], [*0*,*1*,*2*,*3*], [*3*,*0*,*2*,*1*]]
= *Some*[*0*,*1*,*0*,*1*]
by *eval*

lemma *Gale-Shapley1*
[[*4*,*6*,*0*,*1*,*5*,*7*,*3*,*2*], [*1*,*2*,*6*,*4*,*3*,*0*,*7*,*5*], [*7*,*4*,*0*,*3*,*5*,*1*,*2*,*6*], [*2*,*1*,*6*,*3*,*0*,*5*,*7*,*4*],
[*6*,*1*,*4*,*0*,*2*,*5*,*7*,*3*], [*0*,*5*,*6*,*4*,*7*,*3*,*1*,*2*], [*1*,*4*,*6*,*5*,*2*,*3*,*7*,*0*], [*2*,*7*,*3*,*4*,*6*,*1*,*5*,*0*]]
[[*4*,*2*,*6*,*5*,*0*,*1*,*7*,*3*], [*7*,*5*,*2*,*4*,*6*,*1*,*0*,*3*], [*0*,*4*,*5*,*1*,*3*,*7*,*6*,*2*], [*7*,*6*,*2*,*1*,*3*,*0*,*4*,*5*],
[*5*,*3*,*6*,*2*,*7*,*0*,*1*,*4*], [*1*,*7*,*4*,*2*,*3*,*5*,*6*,*0*], [*6*,*4*,*1*,*0*,*7*,*5*,*3*,*2*], [*6*,*3*,*0*,*4*,*1*,*2*,*5*,*7*]]
= *Some* [*0*, *1*, *0*, *5*, *0*, *0*, *0*, *2*]
by *eval*

end

3 Part 2: Refinement from lists to arrays

theory *Gale-Shapley2*
imports *Gale-Shapley1 Collections.Diff-Array*
begin

Refinement from lists to arrays, resulting in a linear (in the input size, which is *n*^*2*) time algorithm.

abbreviation *array* ≡ *new-array*
notation *array-get* (**infixl** ‹!!› *100*)
notation *array-set* (‹-[- ::= -]› [*1000*,*0*,*0*] *900*)
abbreviation *list* ≡ *list-of-array*

lemma *list-array*: *list* (*array x n*) = *replicate n x*
by (*simp add*: *new-array-def*)

lemma *array-get*: *a* !! *i* = *list a* ! *i*
by (*cases a*) *simp*

context *Pref*
begin

3.1 Algorithm 7: Arrays

definition *match-array A a = P ! a ! (A !! a)*

lemma *match-array*: *match-array A a = match (list A) a*
by(*cases A*) (*simp add*: *match-array-def match-def*)

lemmas *array-abs = match-array array-list-of-set array-get*

lemma *Gale-Shapley7*:
assumes *R = map ranking Q*
shows
VARS A B N ai a a′ b r
[ai = 0 ∧ A = array 0 n ∧ B = array 0 n ∧ N = array False n]
WHILE ai < n
INV { invar1 (list A) (list B) (list N) ai }
VAR {z = n − ai}
DO a := ai; b := match-array A a;
WHILE N !! b
INV { invar2 (list A) (list B) (list N) ai a ∧ b = match-array A a ∧ z = n−ai
}
VAR {var (list A) {<ai}}
DO a′ := B !! b; r := R ! match-array A a′;
IF r ! a < r ! a′
THEN B := B[b ::= a]; A := A[a′ ::= A !! a′ + 1]; a := a′
ELSE A := A[a ::= A !! a + 1]
FI;
b := match-array A a
OD;
B := B[b ::= a]; N := N[b ::= True]; ai := ai+1
OD
[matching (list A) {<n} ∧ stable (list A) {<n} ∧ optiA (list A)]
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *pref-match-def P-set card-distinct match-def list-array index-nth-id prefers-def optiA-def α-def cong*: *conj-cong*)
next
case *2*
thus *?case* **by** (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
next
case *3*
have *R*: *∀ b<n. ∀ a1<n. ∀ a2<n. R ! b ! a1 < R ! b ! a2 ⟷ Q ! b ⊢ a1 < a2*
by (*simp add*: *Q-set ‹R = ›› length-Q ranking-iff-pref*)
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *goal-cases*)
case *1* **show** *?case* **using** *inner-pres*[*OF R - - refl refl refl*] *3*
unfolding *array-abs* **by** *blast*
qed
next

case *4*
show *?case*
proof (*simp only*: *mem-Collect-eq prod.case*, *rule conjI*, *goal-cases*)
case *1* **show** *?case* **using** *4 inner-to-outer* **unfolding** *array-abs* **by** *blast*
next
case *2* **thus** *?case* **using** *4* **by** *auto*
qed
next
case *5*
thus *?case* **using** *pref-match-stable* **unfolding** *invAM-def invar1-def* **by**(*metis le-neq-implies-less*)
qed

3.2 Algorithm 7.1: single-loop variant

lemma *Gale-Shapley7-1*:
assumes $R = map\ ranking\ Q$
shows *VARS A B N a a′ ai b r*
$[ai = 0 \wedge a = 0 \wedge A = array\ 0\ n \wedge B = array\ 0\ n \wedge N = array\ False\ n]$
WHILE $ai < n$
INV { *invar2′* (*list A*) (*list B*) (*list N*) *ai a* }
VAR {*var* (*list A*) ({<*ai+1*} − {*a*})}
DO $b := match\text{-}array\ A\ a$;
IF $\neg\ N\ !!\ b$
THEN $B := B[b ::= a]$; $N := N[b ::= True]$; $ai := ai + 1$; $a := ai$
ELSE $a' := B\ !!\ b$; $r := R\ !\ match\text{-}array\ A\ a'$;
IF $r\ !\ a < r\ !\ a'$
THEN $B := B[b ::= a]$; $A := A[a' ::= A!!a' + 1]$; $a := a'$
ELSE $A := A[a ::= A!!a + 1]$
FI
FI
OD
$[matching\ (list\ A)\ \{<n\} \wedge stable\ (list\ A)\ \{<n\} \wedge optiA\ (list\ A)]$
proof (*vcg-tc*, *goal-cases*)
case *1* **thus** *?case*
by(*auto simp*: *pref-match-def P-set card-distinct match-def list-array index-nth-id prefers-def optiA-def α-def cong*: *conj-cong*)
next
case *3* **thus** *?case* **using** *pref-match-stable atLeast0-lessThan-Suc*[*of n*] **by** *force*
next
case (*2 v A B N a a′ ai*)
have *R′*: $N\ !!\ match\text{-}array\ A\ a \Longrightarrow$
$(R\ !\ match\text{-}array\ A\ (B\ !!\ match\text{-}array\ A\ a)\ !\ a < R\ !\ match\text{-}array\ A\ (B\ !!\ match\text{-}array\ A\ a)\ !\ (B\ !!\ match\text{-}array\ A\ a)) =$
$(Q\ !\ match\text{-}array\ A\ (B\ !!\ match\text{-}array\ A\ a) \vdash a < B\ !!\ match\text{-}array\ A\ a)$
using *R-iff-P 2 assms* **by** (*metis array-abs*)
show *?case*
apply(*simp only*:*mem-Collect-eq prod.case*)
using *2 R′ pres2′*[*of list A list B list N ai a*] **by** (*metis array-abs*)

qed

end

3.3 Executable functional Code

definition *gs-inner* **where**
gs-inner P R N =
while (λ(*A*,*B*,*a*,*b*). *N* !! *b*)
(λ(*A*,*B*,*a*,*b*).
let $a' = B$!! *b*;
$r = R$!! (*P* !! a' !! (*A* !! a')) *in*
let (*A*, *B*, *a*) =
if *r* !! *a* < *r* !! a'
then (*A*[a' ::= *A* !! a' + *1*], *B*[*b* ::= *a*], a')
else (*A*[*a* ::= *A* !! *a* + *1*], *B*, *a*)
in (*A*, *B*, *a*, *P* !! *a* !! (*A* !! *a*)))

definition *gs* :: *nat* ⇒ *nat array array* ⇒ *nat array array*
⇒ *nat array* × *nat array* × *bool array* × *nat* **where**
gs n P R =
while (λ(*A*,*B*,*N*,*ai*). *ai* < *n*)
(λ(*A*,*B*,*N*,*ai*).
let (*A*,*B*,*a*,*b*) = *gs-inner P R N* (*A*, *B*, *ai*, *P* !! *ai* !! (*A* !! *ai*))
in (*A*, *B*[*b* ::= *a*], *N*[*b*::=*True*], *ai*+*1*))
(*array 0 n*, *array 0 n*, *array False n*, *0*)

definition *gs1* :: *nat* ⇒ *nat array array* ⇒ *nat array array*
⇒ *nat array* × *nat array* × *bool array* × *nat* × *nat* **where**
gs1 n P R =
while (λ(*A*,*B*,*N*,*ai*,*a*). *ai* < *n*)
(λ(*A*,*B*,*N*,*ai*,*a*).
let *b* = *P* !! *a* !! (*A* !! *a*)
in *if* ¬ *N* !! *b*
then (*A*, *B*[*b* ::= *a*], *N*[*b* ::= *True*], *ai*+*1*, *ai*+*1*)
else *let* $a' = B$!! *b*;
$r = R$!! (*P* !! a' !! (*A* !! a'))
in *if* *r* !! *a* < *r* !! a'
then (*A*[a' ::= *A*!!a' + *1*], *B*[*b* ::= *a*], *N*, *ai*, a')
else (*A*[*a* ::= *A*!!*a* + *1*], *B*, *N*, *ai*, *a*))
(*array 0 n*, *array 0 n*, *array False n*, *0*, *0*)

definition *pref-array* = *array-of-list o map array-of-list*

lemma *list-list-pref-array*: *i* < *length Pa* ⟹ *list* (*list* (*pref-array Pa*) ! *i*) = *Pa* !
i
by(*simp add*: *pref-array-def*)

fun *rk-of-pref* :: *nat* ⇒ *nat array* ⇒ *nat list* ⇒ *nat array* **where**
rk-of-pref r rs (*n*#*ns*) = (*rk-of-pref* (*r*+*1*) *rs ns*)[*n* ::= *r*] |
rk-of-pref r rs [] = *rs*

definition *rank-array1* :: *nat list* ⇒ *nat array* **where**
rank-array1 P = *rk-of-pref 0* (*array 0* (*length P*)) *P*

definition *rank-array* :: *nat list list* ⇒ *nat array array* **where**
rank-array = *array-of-list o map rank-array1*

lemma *length-rk-of-pref*[*simp*]: *array-length*(*rk-of-pref v vs P*) = *array-length vs*
by(*induction P arbitrary*: *v*)(*auto*)

lemma *nth-rk-of-pref*:
⟦ *length P* ≤ *array-length rs*; *i* ∈ *set P*; *distinct P*; *set P* ⊆ {<*array-length rs*} ⟧
⟹ *rk-of-pref r rs P* !! *i* = *index P i* + *r*
by(*induction P arbitrary*: *r i*) (*auto simp add*: *array-get-array-set-other*)

lemma *rank-array1-iff-pref*: ⟦ *set P* = {<*length P*}; *i* < *length P*; *j* < *length P* ⟧
⟹ *rank-array1 P* !! *i* < *rank-array1 P* !! *j* ⟷ *P* ⊢ *i* < *j*
by(*simp add*: *rank-array1-def prefers-def nth-rk-of-pref card-distinct*)

definition *Gale-Shapley* **where**
Gale-Shapley P Q =
(*if Pref P Q*
then Some (*fst* (*gs* (*length P*) (*pref-array P*) (*rank-array Q*)))
else None)

definition *Gale-Shapley1* **where**
Gale-Shapley1 P Q =
(*if Pref P Q*
then Some (*fst* (*gs1* (*length P*) (*pref-array P*) (*rank-array Q*)))
else None)

context *Pref*
begin

lemma *gs-inner*:
assumes *R* = *rank-array Q*
assumes *invar2* (*list A*) (*list B*) (*list N*) *ai a b* = *match-array A a*
shows *gs-inner* (*pref-array P*) *R N* (*A*, *B*, *a*, *b*) = (*A*′,*B*′,*a*′,*b*′)
⟶ *invar1* (*list A*′) ((*list B*′)[*b*′ := *a*′]) ((*list N*)[*b*′ := *True*]) (*ai*+*1*)
unfolding *gs-inner-def*
proof(*rule while-rule2*[**where**
P = λ(*A*,*B*,*a*,*b*). *invar2* (*list A*) (*list B*) (*list N*) *ai a* ∧ *b* = *match-array A a*
and *r* = *measure* (λ(*A*, *B*, *a*, *b*). *Pref.var0 P* (*list A*) {<*n*})], *goal-cases*)

```
  case 1
  show ?case using assms unfolding var-def by simp
next
  case inv: (2 s)
  obtain A B a b where s: s = (A, B, a, b)
    using prod-cases4 by blast
  show ?case
  proof(rule, goal-cases)
    case 1
    have *: a < n using s inv(1)[unfolded invar2-def] by (auto)
    hence 2: list A ! a < n using s inv(1)[unfolded invar2-def]
      apply simp using * wf-less-n by presburger
    hence match (list A) a < n
      by (metis * P-set atLeast0LessThan lessThan-iff match-def nth-mem)
   from this have **: list B ! match (list A) a < n using s inv[unfolded invar2-def]
      apply (simp add: array-abs ran-def) using atLeast0LessThan by blast
    have R: ∀ b<n. ∀ a1<n. ∀ a2<n. map list (list R) ! b ! a1 < map list (list R)
! b ! a2 ⟷ Q ! b ⊢ a1 < a2
      using rank-array1-iff-pref by(simp add: ‹R = -› length-Q array-get Q-set
rank-array-def)
    have ***: match (list A) (list B ! b) < length (list R) using s inv(1)[unfolded
invar2-def]
        using ** by(simp add: ‹R = -› rank-array-def match-array match-less-n
length-Q)
    show ?case
    using inv apply(simp only: s prod.case Let-def split: if-split)
    using inner-pres[OF R - - refl refl refl refl refl, of list A list B list N ai a b]
    unfolding invar2-def array-abs
        list-list-pref-array[OF **[unfolded n-def]] list-list-pref-array[OF *[unfolded
n-def]] nth-map[OF ***]
    unfolding match-def by presburger
    case 2 show ?case
    using inv apply(simp only: s prod.case Let-def in-measure split: if-split)
    using inner-pres2[OF R - - refl refl refl refl refl, of list A list B list N ai a b]
    unfolding invar2-def array-abs
        list-list-pref-array[OF **[unfolded n-def]] list-list-pref-array[OF *[unfolded
n-def]] nth-map[OF ***]
    unfolding match-def by presburger
  qed
next
  case 3
  show ?case
  proof (rule, goal-cases)
      case 1 show ?case by(rule inner-to-outer[OF 3[unfolded 1 prod.case ar-
ray-abs]])
  qed
next
  case 4
  show ?case by simp
```

qed

lemma *gs*: **assumes** R = *rank-array* Q
shows *gs* n (*pref-array* P) R = (A,B,N,ai) ⟶ *matching* (*list* A) {<n} ∧ *stable* (*list* A) {<n} ∧ *optiA* (*list* A)
unfolding *gs-def*
proof(*rule while-rule2*[**where** P = λ(A,B,N,ai). *invar1* (*list* A) (*list* B) (*list* N) *ai*
  and r = *measure*(λ(A,B,N,ai). n − ai)], *goal-cases*)
  case *1* **show** *?case*
  by(*auto simp*: *pref-match-def P-set card-distinct match-def list-array index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)
next
  case (*2 s*)
  obtain A B N *ai* **where** *s*: s = (A, B, N, *ai*)
    using *prod-cases4* **by** *blast*
  have *1*: *invar2* (*list* A) (*list* B) (*list* N) *ai ai* **using** *2 s*
   by (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
  hence *ai* < n **by**(*simp*)
  show *?case* **using** *2 s gs-inner*[*OF* ‹R = - › *1*]
    by (*auto simp*: *array-abs match-def list-list-pref-array*[*OF* ‹*ai* < n›[*unfolded n-def*]]
        simp del: *invar1-def split*: *prod.split*)
next
  case *3*
  thus *?case* **using** *pref-match-stable* **by** *auto*
next
  case *4*
  show *?case* **by** *simp*
qed

lemma *R-iff-P*:
assumes R = *rank-array* Q *invar2'* A B N *ai* a *ai* < n N ! *match* A a
  b = *match* A a a' = B ! b
shows (*list* (*list* R ! *match* A a') ! a < *list* (*list* R ! *match* A a') ! a')
    = (Q ! *match* A a' ⊢ a < a')
proof −
  have R: ∀ b<n. ∀ *a1*<n. ∀ *a2*<n. R !! b !! *a1* < R !! b !! *a2* ⟷ Q ! b ⊢ *a1* < *a2*
   by (*simp add*: *Q-set* ‹R = -› *length-Q array-of-list-def rank-array-def rank-array1-iff-pref*)
  let *?M* = {<*ai*+*1*} − {a}
  have A: *wf* A **and** M: *?M* ⊆ {<n} **and** *as*: a < n **and** *invAB*: *invAB2* A B N *?M*
    using *assms*(*2*,*3*) **by** *auto*
  have a': B ! *match* A a ∈ *?M*
   using *invAB match-less-n*[*OF* A] *as* ‹N!*match* A a› **by** (*metis* α*-Some ranI*)
  hence B ! *match* A a < n **using** M **by** *auto*
  thus *?thesis* **using** *assms match-less-n* R **by** *simp* (*metis array-get as*)

qed

lemma *gs1*: **assumes** R = *rank-array* Q
shows *gs1* n (*pref-array* P) $R = (A,B,N,ai,a) \longrightarrow$ *matching* (*list* A) $\{<n\} \wedge$ *stable* (*list* A) $\{<n\} \wedge$ *optiA* (*list* A)
unfolding *gs1-def*
proof(*rule while-rule2*[**where** $P = \lambda(A,B,N,ai,a)$. *invar2'* (*list* A) (*list* B) (*list* N) *ai a*
and r = *measure*($\lambda(A,B,N,ai,a)$. *Pref.var P* (*list A*) ($\{<ai+1\} - \{a\}$))], *goal-cases*)
case *1* **show** *?case*
by(*auto simp*: *pref-match-def P-set card-distinct match-def list-array index-nth-id prefers-def optiA-def* α*-def cong*: *conj-cong*)
next
case (*2 s*)
obtain *A B N ai a* **where** *s*: $s = (A, B, N, ai, a)$
using *prod-cases5* **by** *blast*
have *1*: *invar2'* (*list A*) (*list B*) (*list N*) *ai a* **using** *2*(*1*) *s*
by (*auto simp*: *atLeastLessThanSuc-atLeastAtMost simp flip*: *atLeastLessThan-eq-atLeastAtMost-diff*)
have $ai < n$ **using** *2*(*2*) *s* **by**(*simp*)
hence $a < n$ **using** *1* **by** *simp*
hence *match* (*list A*) $a < n$ **using** *1 match-less-n* **by** *auto*
hence $*$: *list N* ! *match* (*list A*) $a \Longrightarrow$ *list B* ! *match* (*list A*) $a < n$
using *s 1*[*unfolded invar2'-def*] **apply** (*simp add*: *array-abs ran-def*)
using *atLeast0LessThan* **by** *blast*
have R': *list N* ! *match* (*list A*) $a \Longrightarrow$
(*list* (*list R* ! *match* (*list A*) (*list B* ! *match* (*list A*) *a*)) ! *a*
$<$ *list* (*list R* ! *match* (*list A*) (*list B* ! *match* (*list A*) *a*)) ! (*list B* ! *match* (*list A*) *a*)) =
(*Q* ! *match* (*list A*) (*list B* ! *match* (*list A*) *a*) $\vdash$ *a* $<$ *list B* ! *match* (*list A*) *a*)
using *R-iff-P* ‹R = -› *1* ‹$ai < n$› **by** *blast*
show *?case*
using *s* **apply** (*simp add*: *Let-def*)
unfolding *list-list-pref-array*[*OF* ‹$a < n$›[*unfolded n-def*]] *array-abs*
using *list-list-pref-array*[*OF* $*$[*unfolded n-def*]]
pres2'[*OF 1* ‹$ai < n$› *refl refl refl refl refl*] R'
apply(*intro conjI impI*) **by** (*auto simp*: *match-def*)

next
case *3* **thus** *?case* **using** *pref-match-stable atLeast0-lessThan-Suc*[*of n*] **by** *force*
next
case *4* **show** *?case* **by** *simp*
qed

end

theorem *gs*: $[\![$ *Pref P Q*; n = *length P* $]\!] \Longrightarrow$
$\exists A$. *Gale-Shapley P Q* = *Some A*
$\wedge$ *Pref.matching P* (*list A*) $\{<n\} \wedge$ *Pref.stable P Q* (*list A*) $\{<n\} \wedge$ *Pref.optiA*

P Q (*list A*)
unfolding *Gale-Shapley-def* **using** *Pref.gs*
by (*metis fst-conv surj-pair*)

theorem *gs1*: ⟦ *Pref P Q*; *n* = *length P* ⟧ ⟹
∃ *A*. *Gale-Shapley1 P Q* = *Some A*
∧ *Pref.matching P* (*list A*) {<*n*} ∧ *Pref.stable P Q* (*list A*) {<*n*} ∧ *Pref.optiA*
P Q (*list A*)
unfolding *Gale-Shapley1-def* **using** *Pref.gs1*
by (*metis fst-conv surj-pair*)

Two examples from Gusfield and Irving:

lemma *list-of-array* (*the* (*Gale-Shapley*
[[*3*,*0*,*1*,*2*], [*1*,*2*,*0*,*3*], [*1*,*3*,*2*,*0*], [*2*,*0*,*3*,*1*]] [[*3*,*0*,*2*,*1*], [*0*,*2*,*1*,*3*], [*0*,*1*,*2*,*3*], [*3*,*0*,*2*,*1*]]))
= [*0*,*1*,*0*,*1*]
by *eval*

lemma *list-of-array* (*the* (*Gale-Shapley*
[[*4*,*6*,*0*,*1*,*5*,*7*,*3*,*2*], [*1*,*2*,*6*,*4*,*3*,*0*,*7*,*5*], [*7*,*4*,*0*,*3*,*5*,*1*,*2*,*6*], [*2*,*1*,*6*,*3*,*0*,*5*,*7*,*4*],
[*6*,*1*,*4*,*0*,*2*,*5*,*7*,*3*], [*0*,*5*,*6*,*4*,*7*,*3*,*1*,*2*], [*1*,*4*,*6*,*5*,*2*,*3*,*7*,*0*], [*2*,*7*,*3*,*4*,*6*,*1*,*5*,*0*]]
[[*4*,*2*,*6*,*5*,*0*,*1*,*7*,*3*], [*7*,*5*,*2*,*4*,*6*,*1*,*0*,*3*], [*0*,*4*,*5*,*1*,*3*,*7*,*6*,*2*], [*7*,*6*,*2*,*1*,*3*,*0*,*4*,*5*],
[*5*,*3*,*6*,*2*,*7*,*0*,*1*,*4*], [*1*,*7*,*4*,*2*,*3*,*5*,*6*,*0*], [*6*,*4*,*1*,*0*,*7*,*5*,*3*,*2*], [*6*,*3*,*0*,*4*,*1*,*2*,*5*,*7*]]))
= [*0*, *1*, *0*, *5*, *0*, *0*, *0*, *2*]
by *eval*

end

References

[1] H. G. Baker. Shallow binding makes functional arrays fast. *SIGPLAN Not.*, 26(8):145–147, aug 1991.

[2] S. Conchon and J. Filliâtre. A persistent union-find data structure. In C. V. Russo and D. Dreyer, editors, *Proceedings of the ACM Workshop on ML, 2007, Freiburg, Germany, October 5, 2007*, pages 37–46. ACM, 2007.

[3] D. Gale and L. S. Shapley. College admissions and the stability of marriage. *Am. Math. Mon.*, 69(1):9–15, 1962.

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

[5] P. Lammich. Collections framework. *Archive of Formal Proofs*, Nov. 2009. https://isa-afp.org/entries/Collections.html, Formal proof development.