Gale-Shapley Algorithm

Tobias Nipkow
Department of Informatics
Technical University of Munich

March 17, 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*
⟨*proof*⟩

lemma *prefers-trans*: *P* ⊢ *x* < *y* ⟹ *P* ⊢ *y* < *z* ⟹ *P* ⊢ *x* < *z*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *nth-rk-of-pref*: $\llbracket$ *length* $P \leq$ *length* rs; $i \in$ *set* P; *distinct* P; *set* $P \subseteq \{<$*length* $rs\}$ $\rrbracket$
$\Longrightarrow$ *rk-of-pref* r rs P ! i = *index* P i + r
⟨*proof*⟩

lemma *ranking-index*: $\llbracket$ *length* $P = n$; *set* $P = \{<n\}$ $\rrbracket \Longrightarrow$ *ranking* P = *map* (*index* P) [*0*..<*length* P]
⟨*proof*⟩

lemma *ranking-iff-pref*: $\llbracket$ *set* $P = \{<$*length* $P\}$; $i <$ *length* P; $j <$ *length* P $\rrbracket$
$\Longrightarrow$ *ranking* P ! i < *ranking* P ! $j \longleftrightarrow P \vdash i < j$
⟨*proof*⟩

2.2 Fixing the preference lists

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

locale *Pref* =
fixes n
fixes P :: *prefs*
fixes Q :: *prefs*
defines $n \equiv$ *length* P
assumes *length-Q*: *length* $Q = n$
assumes *P-set*: $a < n \Longrightarrow$ *length*($P!a$) $= n \wedge$ *set*($P!a$) $= \{<n\}$
assumes *Q-set*: $b < n \Longrightarrow$ *length*($Q!b$) $= n \wedge$ *set*($Q!b$) $= \{<n\}$
begin

abbreviation *wf* :: *nat list* ⇒ *bool* **where**
wf xs $\equiv$ *length xs* $= n \wedge$ *set xs* $\subseteq \{<n\}$

lemma *wf-less-n*: $\llbracket$ *wf* A; $a < n$ $\rrbracket \Longrightarrow A!a < n$
⟨*proof*⟩

corollary *wf-le-n1*: $\llbracket$ *wf* A; $a < n$ $\rrbracket \Longrightarrow A!a \leq n-1$
⟨*proof*⟩

lemma *sumA-ub*: *wf* $A \Longrightarrow (\sum a<n.\ A!a) \leq n*(n-1)$
⟨*proof*⟩

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 $M = (n\hat{\ }2 - ((\sum a<n.\ A!a) + card\ M))$

lemma *var0-match*:

assumes $wf\ A\ M \subseteq \{<n\}\ a < n \wedge a \notin M$
shows $var0\ A\ (M \cup \{a\}) < var0\ A\ M$
$\langle proof \rangle$

lemma *var0-next*:
assumes $wf\ A\ M \subseteq \{<n\}\ M \neq \{<n\}\ a' < n$
shows $var0\ (A[a' := A\ !\ a' + 1])\ M < var0\ A\ M$
$\langle proof \rangle$

definition $var :: nat\ list \Rightarrow nat\ set \Rightarrow nat$ **where**
$[simp]$: $var\ A\ M = (n\hat{\ }2 - n + 1 - ((\sum a{<}n.\ A!a) + card\ M))$

lemma *sumA-ub2*:
assumes $a' < n\ A!a' \leq n{-}1\ \forall\, a < n.\ a \neq a' \longrightarrow A!a \leq n{-}2$
shows $(\sum a{<}n.\ A!a) \leq (n{-}1){*}(n{-}1)$
$\langle proof \rangle$

definition $match\ A\ a = P\ !\ a\ !\ (A\ !\ a)$

lemma *match-less-n*: $[\![\ wf\ A;\ a < n\ ]\!] \Longrightarrow match\ A\ a < n$
$\langle proof \rangle$

lemma *match-upd-neq*: $[\![\ wf\ A;\ a < n;\ a \neq a'\ ]\!] \Longrightarrow match\ (A[a := b])\ a' = match\ A\ a'$
$\langle proof \rangle$

definition $stable :: nat\ list \Rightarrow nat\ set \Rightarrow bool$ **where**
$stable\ A\ M = (\neg(\exists\, a{\in}M.\ \exists\, a'{\in}M.\ P\ !\ a \vdash match\ A\ a' < match\ A\ a \wedge Q\ !\ match\ A\ a' \vdash a < a'))$

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** $[simp]$:
$matching\ A\ M = (wf\ A \wedge inj\text{-}on\ (match\ A)\ M)$

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

lemma *final-last*:
assumes M: $M \subseteq \{<n\}$ **and** *inj*: $inj\text{-}on\ (match\ A)\ M$ **and** *pref-match'*: $preferred\ A\ a \subseteq match\ A\ `\ M$
and a: $a < n \wedge a \notin M$ **and** *final*: $A\ !\ a + 1 = n$
shows $insert\ a\ M = \{<n\}$
$\langle proof \rangle$

lemma *more-choices*:
assumes A: $wf\ A$ **and** M: $M \subseteq \{<n\}\ M \neq \{<n\}$
and *pref-match'*: $preferred\ A\ a \subseteq match\ A\ `\ M$

and $a < n$ **and** *matched*: $match\ A\ a \in match\ A\ `\ M$
shows $A\ !\ a + 1 < n$
$\langle proof \rangle$

corollary *more-choices-matched*:
assumes $wf\ A\ M \subseteq \{<n\}\ M \neq \{<n\}$
and $preferred\ A\ a \subseteq match\ A\ `\ M$ **and** $a \in M$
shows $A\ !\ a + 1 < n$
$\langle proof \rangle$

lemma *atmost1-final*: **assumes** M: $M \subseteq \{<n\}$ **and** *inj*: $inj\text{-}on\ (match\ A)\ M$
and $\forall a<n.\ preferred\ A\ a \subseteq match\ A\ `\ M$
shows $\exists_{\leq 1}\ a.\ a < n \wedge a \notin M \wedge A\ !\ a + 1 = n$
$\langle proof \rangle$

lemma *sumA-UB*:
assumes $matching\ A\ M\ M \subseteq \{<n\}\ M \neq \{<n\}\ \forall a<n.\ preferred\ A\ a \subseteq match\ A\ `\ M$
shows $(\sum a<n.\ A!a) \leq (n-1)\hat{\ }2$
$\langle proof \rangle$

lemma *var-ub*:
assumes $matching\ A\ M\ M \subseteq \{<n\}\ M \neq \{<n\}\ \forall a<n.\ preferred\ A\ a \subseteq match\ A\ `\ M$
shows $(\sum a<n.\ A!a) + card\ M < n\hat{\ }2 - n + 1$
$\langle proof \rangle$

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 $var\ A\ (M \cup \{a\}) < var\ A\ M$
$\langle proof \rangle$

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 $var\ (A[a := A\ !\ a + 1])\ M < var\ A\ M$
$\langle proof \rangle$

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**
$pref\text{-}match\ A\ M = (\forall a<n.\ \forall b<n.\ P!a \vdash b < match\ A\ a \longrightarrow (\exists a' \in M.\ b = match\ A\ a' \wedge Q\ !\ b \vdash a' < a))$

definition *pref-match'* **where**

$\textit{pref-match}'\ A\ M = (\forall\, a{<}n.\ \forall\, b \in \textit{preferred}\ A\ a.\ \exists\, a'{\in}M.\ b = \textit{match}\ A\ a' \wedge Q\ !\ b \vdash a' < a)$

lemma *pref-match′-iff*: $\textit{wf}\ A \Longrightarrow \textit{pref-match}'\ A\ M = \textit{pref-match}\ A\ M$
⟨*proof*⟩

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

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

lemma *optiA-pessiB*: **assumes** *optiA A* **shows** *pessiB A*
⟨*proof*⟩

lemma *optiA-inv*:
assumes *A*: *wf A* **and** *a*: $a < n$ **and** *a′*: $a' < n$ **and** *same-match*: $\textit{match}\ A\ a' = \textit{match}\ A\ a$
and *pref*: $Q\ !\ \textit{match}\ A\ a' \vdash a' < a$ **and** *optiA A*
shows $\textit{optiA}\ (A[a := A\ !\ a + 1])$
⟨*proof*⟩

lemma *pref-match-stable*:
$[\![\ \textit{matching}\ A\ \{{<}n\};\ \textit{pref-match}\ A\ \{{<}n\}\ ]\!] \Longrightarrow \textit{stable}\ A\ \{{<}n\}$
⟨*proof*⟩

2.5 Algorithm 1

definition *invAM* **where**
[*simp*]: $\textit{invAM}\ A\ M = (\textit{matching}\ A\ M \wedge M \subseteq \{{<}n\} \wedge \textit{pref-match}\ A\ M \wedge \textit{optiA}\ A)$

lemma *invAM-match*:
$[\![\ \textit{invAM}\ A\ M;\ \ a < n \wedge a \notin M;\ \ \textit{match}\ A\ a \notin \textit{match}\ A\ `\ M\ ]\!] \Longrightarrow \textit{invAM}\ A\ (M \cup \{a\})$
⟨*proof*⟩

lemma *invAM-swap*:
assumes *invAM A M*
assumes *a*: $a < n \wedge a \notin M$ **and** *a′*: $a' \in M \wedge \textit{match}\ A\ a' = \textit{match}\ A\ a$ **and** *pref*: $Q\ !\ \textit{match}\ A\ a' \vdash a < a'$
shows $\textit{invAM}\ (A[a' := A!a'{+}1])\ (M - \{a'\} \cup \{a\})$
⟨*proof*⟩

lemma *preferred-subset-match-if-invAM*:

assumes *invAM A M*
shows $\forall\, a<n.$ *preferred A a* $\subseteq$ *match A ‘ M* (**is** *?P*)
$\langle proof \rangle$

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'* $\vdash a < a'$
shows *invAM* ($A[a := A!a + 1]$) *M*
$\langle proof \rangle$

lemma *Gale-Shapley1*: *VARS M A a a' b*
 [$M = \{\} \wedge A =$ *replicate n 0*]
 WHILE $M \neq \{<n\}$
 INV { *invAM A M* }
 VAR {*var A M*}
 DO $a :=$ (*SOME a.* $a < n \wedge a \notin M$); $b :=$ *match A a*;
  IF $b \notin$ *match A ‘ M*
  THEN $M := M \cup \{a\}$
  ELSE $a' :=$ (*SOME a'.* $a' \in M \wedge$ *match A a'* $= b$);
    IF Q ! *match A a'* $\vdash a < a'$
    THEN $A[a'] := A!a'+1$; $M := M - \{a'\} \cup \{a\}$
    ELSE $A[a] := A!a+1$
    FI
  FI
 OD
 [*matching A* $\{<n\}$ $\wedge$ *stable A* $\{<n\}$ $\wedge$ *optiA A*]
$\langle proof \rangle$

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

2.6 Algorithm 2: List of unmatched As

abbreviation *invas* **where**
invas as == (*set as* $\subseteq \{<n\} \wedge$ *distinct as*)

lemma *Gale-Shapley2*: *VARS A a a' as b*
 [$as = [0..<n] \wedge A =$ *replicate n 0*]
 WHILE $as \neq []$
 INV { *invAM A* ($\{<n\}$ − *set as*) $\wedge$ *invas as*}
 VAR {*var A* ($\{<n\}$ − *set as*)}
 DO $a :=$ *hd as*; $b :=$ *match A a*;
  IF $b \notin$ *match A ‘* ($\{<n\}$ − *set as*)
  THEN *as* := *tl as*
  ELSE $a' :=$ (*SOME a'.* $a' \in \{<n\}$ − *set as* $\wedge$ *match A a'* $= b$);
    IF Q ! *match A a'* $\vdash a < a'$
    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⟩
```

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⟩


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⟩
```

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⟩
```

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⟩
```

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* $M \neq \{<n\}$ $a < n$
shows *var2* $(A[a := A!a + 1]) <$ *var2 A*
⟨*proof*⟩

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*} ∧ *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 {*var2 A*}
  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*⟩

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*
 [$ai = 0 \wedge a = 0 \wedge A = replicate\ n\ 0 \wedge B = (\lambda\text{-}.\ None)$]
 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
$[matching\ A\ \{<n\} \wedge stable\ A\ \{<n\} \wedge optiA\ A]$
$\langle proof \rangle$

2.10 Algorithm 5: Data refinement of B

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

lemma α-$Some[simp]$: $\alpha\ B\ N\ b = Some\ a \longleftrightarrow b < n \wedge N!b \wedge a = B!b$
$\langle proof \rangle$

lemma $\alpha update1$: $[\![\ \neg\ N\ !\ b;\ b < length\ B;\ b < length\ N;\ n = length\ N\ ]\!]$
$\Longrightarrow ran(\alpha\ (B[b := a])\ (N[b := True])) = ran(\alpha\ B\ N) \cup \{a\}$
$\langle proof \rangle$

lemma $\alpha update2$: $[\![\ N!b;\ b < length\ B;\ b < length\ N;\ length\ N = n\ ]\!]$
$\Longrightarrow \alpha\ (B[b := a])\ N = (\alpha\ B\ N)(b := Some\ a)$
$\langle proof \rangle$

abbreviation $invAB2$ :: $nat\ list \Rightarrow nat\ list \Rightarrow bool\ list \Rightarrow nat\ set \Rightarrow bool$ **where**
$invAB2\ A\ B\ N\ M == (invAB\ A\ (\alpha\ B\ N)\ M \wedge (length\ B = n \wedge length\ N = n))$

definition $invar1$ **where**
$[simp]$: $invar1\ A\ B\ N\ ai = (invAM\ A\ \{<ai\} \wedge invAB2\ A\ B\ N\ \{<ai\} \wedge ai \le n)$

definition $invar2$ **where**
$[simp]$: $invar2\ A\ B\ N\ ai\ a \equiv (invAM\ A\ (\{<ai{+}1\} - \{a\}) \wedge invAB2\ A\ B\ N\ (\{<ai{+}1\} - \{a\}) \wedge a \le ai \wedge ai < n)$

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

lemma $Gale$-$Shapley5$:
$VARS\ A\ B\ N\ ai\ a\ a'$
$[ai = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False]$
$WHILE\ ai < n$
$INV\ \{\ invar1\ A\ B\ N\ ai\ \}$
$VAR\ \{\ z = n - ai\}$
$DO\ a := ai;$
$WHILE\ N\ !\ match\ A\ a$
$INV\ \{\ invar2\ A\ B\ N\ ai\ a \wedge z = n{-}ai\ \}$
$VAR\ \{var\ A\ \{<ai\}\}$
$DO\ a' := B\ !\ match\ A\ a;$
$IF\ Q\ !\ match\ A\ a' \vdash a < a'$
$THEN\ B[match\ A\ a] := a;\ A[a'] := A!a'{+}1;\ a := a'$
$ELSE\ A[a] := A!a{+}1$
FI
$OD;$

$B[match\ A\ a] := a;\ N[match\ A\ a] := True;\ ai := ai{+}1$
OD
$[matching\ A\ \{{<}n\} \wedge stable\ A\ \{{<}n\} \wedge optiA\ A]$
$\langle proof \rangle$

The definitive version with variant *var2*:

lemma *Gale-Shapley5-var2*:
$VARS\ A\ B\ N\ ai\ a\ a'$
$[ai = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False]$
$WHILE\ ai < n$
$INV\ \{\ invar1\ A\ B\ N\ ai\ \}$
$VAR\ \{\ z = n - ai\}$
$DO\ a := ai;$
$WHILE\ N\ !\ match\ A\ a$
$INV\ \{\ invar2\ A\ B\ N\ ai\ a \wedge z = n{-}ai\ \}$
$VAR\ \{var2\ A\}$
$DO\ a' := B\ !\ match\ A\ a;$
$IF\ Q\ !\ match\ A\ a' \vdash a < a'$
$THEN\ B[match\ A\ a] := a;\ A[a'] := A!a'{+}1;\ a := a'$
$ELSE\ A[a] := A!a{+}1$
FI
$OD;$
$B[match\ A\ a] := a;\ N[match\ A\ a] := True;\ ai := ai{+}1$
OD
$[matching\ A\ \{{<}n\} \wedge stable\ A\ \{{<}n\} \wedge optiA\ A]$
$\langle proof \rangle$

2.10.1 Algorithm 5.1: single-loop variant

definition *invar2'* **where**
$[simp]$: $invar2'\ A\ B\ N\ ai\ a \equiv (invAM\ A\ (\{{<}ai{+}1\} - \{a\}) \wedge invAB2\ A\ B\ N\ (\{{<}ai{+}1\} - \{a\}) \wedge a \le ai \wedge ai \le n)$

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
$(\neg\ N\ !\ b \longrightarrow$
$invar2'\ A\ (B[b := a])\ (N[b := True])\ (ai + 1)\ (ai + 1) \wedge var\ A\ (\{{<}ai + 1 + 1\} - \{ai + 1\}) < v) \wedge$
$(N\ !\ b \longrightarrow$
$(Q\ !\ match\ A\ a' \vdash a < a' \longrightarrow invar2'\ A1\ (B[b := a])\ N\ ai\ a' \wedge var\ A1\ (\{{<}ai + 1\} - \{a'\}) < v) \wedge$
$(\neg\ Q\ !\ match\ A\ a' \vdash a < a' \longrightarrow invar2'\ A2\ B\ N\ ai\ a \wedge var\ A2\ (\{{<}ai + 1\} - \{a\}) < v))$
$\langle proof \rangle$

lemma *Gale-Shapley5-1*: $VARS\ A\ B\ N\ a\ a'\ ai\ b$
$[ai = 0 \wedge a = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False]$

$\mathit{WHILE}\ ai < n$
$\mathit{INV}\ \{\ \mathit{invar2}'\ A\ B\ N\ ai\ a\ \}$
$\mathit{VAR}\ \{\mathit{var}\ A\ (\{<ai{+}1\} - \{a\})\}$
$\mathit{DO}\ b := \mathit{match}\ A\ a;$
$\quad \mathit{IF}\ \neg\ N\ !\ b$
$\quad \mathit{THEN}\ B[b] := a;\ N[b] := \mathit{True};\ ai := ai + 1;\ a := ai$
$\quad \mathit{ELSE}\ a' := B\ !\ b;$
$\qquad \mathit{IF}\ Q\ !\ \mathit{match}\ A\ a' \vdash a < a'$
$\qquad \mathit{THEN}\ B[b] := a;\ A[a'] := A!a'{+}1;\ a := a'$
$\qquad \mathit{ELSE}\ A[a] := A!a{+}1$
$\qquad \mathit{FI}$
$\quad \mathit{FI}$
OD
$[\mathit{matching}\ A\ \{<n\} \wedge \mathit{stable}\ A\ \{<n\} \wedge \mathit{optiA}\ A]$
$\langle \mathit{proof} \rangle$

2.11 Algorithm 6: replace Q by ranking R

lemma *inner-to-outer*:
assumes *inv*: $\mathit{invar2}\ A\ B\ N\ ai\ a \wedge b = \mathit{match}\ A\ a$ **and** *not-b*: $\neg\ N\ !\ b$
shows $\mathit{invar1}\ A\ (B[b := a])\ (N[b := \mathit{True}])\ (ai{+}1)$
$\langle \mathit{proof} \rangle$

lemma *inner-pres*:
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: $\mathit{invar2}\ A\ B\ N\ ai\ a$ **and** *m*: $N\ !\ b$ **and** *v*: $\mathit{var}\ A\ \{<ai\} = v$
and *after*: $A1 = A[a' := A\ !\ a' + 1]\ A2 = A[a := A\ !\ a + 1]$
$a' = B!b\ r = R\ !\ \mathit{match}\ A\ a'\ b = \mathit{match}\ A\ a$
shows $(r\ !\ a < r\ !\ a' \longrightarrow \mathit{invar2}\ A1\ (B[b{:=}a])\ N\ ai\ a' \wedge \mathit{var}\ A1\ \{<ai\} < v) \wedge$
$(\neg\ r\ !\ a < r\ !\ a' \longrightarrow \mathit{invar2}\ A2\ B\ N\ ai\ a \wedge \mathit{var}\ A2\ \{<ai\} < v)$
$\langle \mathit{proof} \rangle$

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

lemma *Gale-Shapley6*:
assumes $R = \mathit{map}\ \mathit{ranking}\ Q$
shows
$\mathit{VARS}\ A\ B\ N\ ai\ a\ a'\ b\ r$
$[ai = 0 \wedge A = \mathit{replicate}\ n\ 0 \wedge \mathit{length}\ B = n \wedge N = \mathit{replicate}\ n\ \mathit{False}]$
$\mathit{WHILE}\ ai < n$
$\mathit{INV}\ \{\ \mathit{invar1}\ A\ B\ N\ ai\ \}$
$\mathit{VAR}\ \{z = n - ai\}$
$\mathit{DO}\ a := ai;\ b := \mathit{match}\ A\ a;$
$\quad \mathit{WHILE}\ N\ !\ b$
$\quad \mathit{INV}\ \{\ \mathit{invar2}\ A\ B\ N\ ai\ a \wedge b = \mathit{match}\ A\ a \wedge z = n{-}ai\ \}$
$\quad \mathit{VAR}\ \{\mathit{var}\ A\ \{<ai\}\}$
$\quad \mathit{DO}\ a' := B\ !\ b;\ r := R\ !\ \mathit{match}\ A\ a';$
$\qquad \mathit{IF}\ r\ !\ a < r\ !\ a'$
$\qquad \mathit{THEN}\ B[b] := a;\ A[a'] := A!a'{+}1;\ a := a'$
$\qquad \mathit{ELSE}\ A[a] := A!a{+}1$

FI;
$b := match\ A\ a$
OD;
$B[b] := a$; $N[b] := True$; $ai := ai+1$
OD
$[matching\ A\ \{<n\} \wedge stable\ A\ \{<n\} \wedge optiA\ A]$
$\langle proof \rangle$

lemma *inner-pres-var2*:
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: $var2\ A = v$
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 $(r\ !\ a < r\ !\ a' \longrightarrow invar2\ A1\ (B[b:=a])\ N\ ai\ a' \wedge var2\ A1 < v) \wedge$
$(\neg\ r\ !\ a < r\ !\ a' \longrightarrow invar2\ A2\ B\ N\ ai\ a \wedge var2\ A2 < v)$
$\langle proof \rangle$

The definitive version with variant *var2*:

lemma *Gale-Shapley6-var2*:
assumes $R = map\ ranking\ Q$
shows
$VARS\ A\ B\ N\ ai\ a\ a'\ b\ r$
$[ai = 0 \wedge A = replicate\ n\ 0 \wedge length\ B = n \wedge N = replicate\ n\ False]$
$WHILE\ ai < n$
$INV\ \{\ invar1\ A\ B\ N\ ai\ \}$
$VAR\ \{z = n - ai\}$
$DO\ a := ai$; $b := match\ A\ a$;
$WHILE\ N\ !\ b$
$INV\ \{\ invar2\ A\ B\ N\ ai\ a \wedge b = match\ A\ a \wedge z = n-ai\ \}$
$VAR\ \{var2\ A\}$
$DO\ a' := B\ !\ b$; $r := R\ !\ match\ A\ a'$;
$IF\ r\ !\ a < r\ !\ a'$
$THEN\ B[b] := a$; $A[a'] := A!a'+1$; $a := a'$
$ELSE\ A[a] := A!a+1$
FI;
$b := match\ A\ a$
OD;
$B[b] := a$; $N[b] := True$; $ai := ai+1$
OD
$[matching\ A\ \{<n\} \wedge stable\ A\ \{<n\} \wedge optiA\ A]$
$\langle proof \rangle$

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*⟩

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*⟩

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*⟩

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*⟩

lemma *Gale-Shapley6-1*:

assumes *R = map ranking Q*
shows *VARS A B N a a′ ai b r*
 [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; 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} ∧ stable A {<n} ∧ optiA A]
⟨*proof*⟩

lemma *Gale-Shapley6-1-explicit*:
assumes *R = map ranking Q*
shows *VARS A B N a a′ ai b r*
 [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; 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} ∧ stable A {<n} ∧ optiA A]
⟨*proof*⟩

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 $(\lambda(A,B,N,ai).\ ai < n)$
$(\lambda(A,B,N,ai).$
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 $(\lambda(A,B,N,ai,a).\ ai < n)$
$(\lambda(A,B,N,ai,a).$
let $b = P\ !\ a\ !\ (A\ !\ a)$ *in*
if $\neg\ N\ !\ b$
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* $P\ R\ N\ (A,\ B,\ a,\ b) = (A',B',a',b') \longrightarrow$ *invar1* $A'\ (B'[b' := a'])$ $(N[b' := True])\ (ai{+}1)$
⟨*proof*⟩

lemma *gs*: **assumes** $R =$ *map ranking Q*
shows *gs n P R* = (*A*,*BNai*) ⟶ *matching A* $\{<n\} \wedge$ *stable A* $\{<n\} \wedge$ *optiA A*
⟨*proof*⟩

lemma *gs1*: **assumes** $R =$ *map ranking Q*
shows *gs1 n P R* = (*A*,*BNaia*) ⟶ *matching A* $\{<n\} \wedge$ *stable A* $\{<n\} \wedge$ *optiA A*
⟨*proof*⟩

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*: $\llbracket$ *Pref P Q*; n = *length P* $\rrbracket \Longrightarrow$
$\exists$ *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*
⟨*proof*⟩

declare *Pref-def* [*code*]

definition
Gale-Shapley1 P Q = (*if Pref P Q then Some* (*fst* (*gs1* (*length P*) *P* (*map ranking Q*))) *else None*)

theorem *gs1*: $\llbracket$ *Pref P Q*; n = *length P* $\rrbracket \Longrightarrow$
$\exists$ *A*. *Gale-Shapley1 P Q* = *Some*(*A*) $\wedge$ *Pref.matching P A* {<*n*} $\wedge$
Pref.stable P Q A {<*n*} $\wedge$ *Pref.optiA P Q A*
⟨*proof*⟩

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]
⟨*proof*⟩

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]
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *array-get*: *a* !! *i* = *list a* ! *i*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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 ∧ a = 0 ∧ A = array 0 n ∧ B = array 0 n ∧ 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-array A a;
  IF ¬ N !! b
  THEN B := B[b ::= a]; N := N[b ::= True]; ai := ai + 1; a := ai
  ELSE 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
  FI
 OD
 [matching (list A) {<n} ∧ stable (list A) {<n} ∧ optiA (list A)]
⟨proof⟩

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
```

$gs1\ n\ P\ R =$
$\quad while\ (\lambda(A,B,N,ai,a).\ ai < n)$
$\quad\ (\lambda(A,B,N,ai,a).$
$\quad\quad let\ b = P\ !!\ a\ !!\ (A\ !!\ a)$
$\quad\quad in\ if\ \neg\ N\ !!\ b$
$\quad\quad\quad then\ (A,\ B[b ::= a],\ N[b ::= True],\ ai{+}1,\ ai{+}1)$
$\quad\quad\quad else\ let\ a' = B\ !!\ b;$
$\quad\quad\quad\quad\quad\quad r = R\ !!\ (P\ !!\ a'\ !!\ (A\ !!\ a'))$
$\quad\quad\quad\quad in\ if\ r\ !!\ \ a < r\ !!\ a'$
$\quad\quad\quad\quad\quad then\ (A[a' ::= A!!a' + 1],\ B[b ::= a],\ N,\ ai,\ a')$
$\quad\quad\quad\quad\quad else\ (A[a ::= A!!a + 1],\ B,\ N,\ ai,\ a))$
$\quad (array\ 0\ n,\ array\ 0\ n,\ array\ False\ n,\ 0,\ 0)$

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

lemma *list-list-pref-array*: $i < length\ Pa \Longrightarrow list\ (list\ (pref\text{-}array\ Pa)\ !\ i) = Pa\ !\ i$
⟨*proof*⟩

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

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

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*]: $array\text{-}length(rk\text{-}of\text{-}pref\ v\ vs\ P) = array\text{-}length\ vs$
⟨*proof*⟩

lemma *nth-rk-of-pref*:
$[\![\ length\ P \le array\text{-}length\ rs;\ i \in set\ P;\ distinct\ P;\ set\ P \subseteq \{<array\text{-}length\ rs\}\ ]\!]$
$\Longrightarrow rk\text{-}of\text{-}pref\ r\ rs\ P\ !!\ i = index\ P\ i + r$
⟨*proof*⟩

lemma *rank-array1-iff-pref*: $[\![\ set\ P = \{<length\ P\};\ i < length\ P;\ j < length\ P\ ]\!]$
$\Longrightarrow rank\text{-}array1\ P\ !!\ i < rank\text{-}array1\ P\ !!\ j \longleftrightarrow P \vdash i < j$
⟨*proof*⟩

definition *Gale-Shapley* **where**
Gale-Shapley P Q =
$\quad (if\ Pref\ P\ Q$
$\quad\ then\ Some\ (fst\ (gs\ (length\ P)\ (pref\text{-}array\ P)\ (rank\text{-}array\ Q)))$
$\quad\ else\ None)$

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')
$\longrightarrow$ *invar1* (*list* A') ((*list* B')[b' := a']) ((*list N*)[b' := *True*]) (*ai*+*1*)
⟨*proof*⟩

lemma *gs*: **assumes** R = *rank-array Q*
shows *gs n* (*pref-array P*) R = (*A*,*B*,*N*,*ai*) $\longrightarrow$ *matching* (*list A*) $\{<n\}$ $\wedge$ *stable* (*list A*) $\{<n\}$ $\wedge$ *optiA* (*list A*)
⟨*proof*⟩

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' $\vdash$ *a* < a')
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

theorem *gs1*: $[\![$ *Pref P Q*; *n* = *length P* $]\!] \Longrightarrow$
$\exists$ *A*. *Gale-Shapley1 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*)
⟨*proof*⟩

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*]
⟨*proof*⟩

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*]
⟨*proof*⟩

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.