Nash Equilibria for Finite Games in Isabelle/HOL

Arthur Freitas Ramos David Barros Hulak
Ruy J. G. B. de Queiroz

June 25, 2026

Abstract

This development formalizes Nash equilibria for finite strategic-form games, following Nash's equilibrium concept [5, 4]. It gives reusable definitions of profiles, unilateral deviations, best responses, dominant strategies, and pure Nash equilibria; proves existence for finite ordinal potential games [3] and games with dominant strategies; and verifies matching pennies as a finite game with no pure Nash equilibrium. It also formalizes mixed-strategy profiles for finite games, support lemmas for equilibrium strategies, Dirac embeddings of pure profiles, and the existence of a mixed Nash equilibrium using Brouwer's fixed point theorem. Worked examples cover the Prisoner's Dilemma, a coordination game, matching pennies, and rock-paper-scissors. AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

Contributions and Scope

This entry gives a reusable Isabelle/HOL formalization of pure and mixed Nash equilibria for finite strategic-form games. The pure-game locale supports a finite player set with player-indexed finite strategy sets and is used to formalize unilateral deviations, best responses, dominant strategies, and ordinal potential games. The mixed-game development proves Nash's finite-game existence theorem by defining an excess-payoff map on a compact convex product of simplices and applying Brouwer's fixed point theorem from HOL-Analysis.

The mixed-game locale is intentionally less general than the pure-game locale: players and pure strategies are represented by finite HOL types, so every player uses the same finite pure-strategy type. This choice makes mixed profiles Cartesian vectors indexed by player/strategy pairs, which gives direct access to the compactness, convexity, continuity, and fixed-point infrastructure needed for the Brouwer proof. The entry also includes support lemmas for mixed equilibria, Dirac embeddings of pure profiles, and checked examples of the Prisoner's Dilemma, a coordination game, matching pennies, and rock-paper-scissors.

Related Work

Le Roux, Martin-Dorel, and Smaus formalized a Nash-equilibrium existence theorem in both Coq and Isabelle for finite-outcome games derived from win/lose games [2]. Bagnall, Merten, and Stewart developed an Ssreflect/Coq library for algorithmic game theory, including pure and mixed Nash equilibria, potential games, smooth games, approximate equilibria, and applications to routing and congestion games [1]. The present entry is narrower in mathematical scope but focuses on an AFP-style Isabelle/HOL development for finite strategic-form games, with a Brouwer-based mixed-equilibrium existence proof and small canonical examples.

Contents

theory *Nash-Equilibrium*
  imports *Main*
begin

1 Pure Nash Equilibria

Nash's equilibrium concept says that a strategy profile is stable when no player can improve by changing only her own strategy. This entry formalizes the pure-strategy version for finite strategic-form games with a common strategy type.

The central locale below fixes a finite set of players, a finite nonempty strategy set for each player, and a payoff function. The player and strategy types are finite; this keeps the profile space finite while still allowing player-indexed strategy restrictions.

The development is intended as a reusable finite-game layer: the basic locale gives pure Nash equilibria and best responses, later locales derive existence from ordinal potentials and dominant strategies, and the companion mixed theory proves the finite mixed-equilibrium theorem using HOL-Analysis.

```
locale finite-game =
  fixes players :: 'p::finite set
    and strategies :: 'p ⇒ 's::finite set
    and payoff :: 'p ⇒ ('p ⇒ 's) ⇒ 'u::preorder
  assumes nonempty-strategies: i ∈ players ⟹ strategies i ≠ {}
begin

definition profiles :: ('p ⇒ 's) set where
  profiles = {s. ∀ i∈players. s i ∈ strategies i}

definition deviation :: ('p ⇒ 's) ⇒ 'p ⇒ 's ⇒ ('p ⇒ 's) where
  deviation s i x = s(i := x)

definition Nash-equilibrium :: ('p ⇒ 's) ⇒ bool where
  Nash-equilibrium s ⟷
    s ∈ profiles ∧
    (∀ i∈players. ∀ x∈strategies i.
      payoff i (deviation s i x) ≤ payoff i s)

definition best-response-to :: ('p ⇒ 's) ⇒ 'p ⇒ 's ⇒ bool where
  best-response-to s i x ⟷
    i ∈ players ∧ x ∈ strategies i ∧
    (∀ y∈strategies i. payoff i (deviation s i y) ≤ payoff i (deviation s i x))

definition dominant-strategy :: 'p ⇒ 's ⇒ bool where
  dominant-strategy i x ⟷
    i ∈ players ∧ x ∈ strategies i ∧
    (∀ s∈profiles. ∀ y∈strategies i.
```

payoff i (*deviation s i y*) ≤ *payoff i* (*deviation s i x*))

lemma *profiles-iff*:
s ∈ *profiles* ⟷ (∀ *i*∈*players*. *s i* ∈ *strategies i*)
by (*auto simp*: *profiles-def*)

lemma *profile-strategy*:
assumes *s* ∈ *profiles i* ∈ *players*
shows *s i* ∈ *strategies i*
using *assms* **by** (*simp add*: *profiles-iff*)

lemma *finite-profiles* [*simp*, *intro*]: *finite profiles*
by *simp*

lemma *profiles-nonempty*: *profiles* ≠ {}
proof −
have ∀ *i*∈*players*. ∃ *x*. *x* ∈ *strategies i*
using *nonempty-strategies* **by** *auto*
then obtain *f* **where** *f*: ⋀*i*. *i* ∈ *players* ⟹ *f i* ∈ *strategies i*
by *metis*
define *g* **where** *g i* = *f i* **for** *i*
have *g* ∈ *profiles*
using *f* **by** (*auto simp*: *profiles-iff g-def*)
then show *?thesis*
by *blast*
qed

lemma *deviation-apply* [*simp*]:
deviation s i x j = (*if j* = *i then x else s j*)
by (*simp add*: *deviation-def*)

lemma *deviation-self* [*simp*]: *deviation s i* (*s i*) = *s*
by (*simp add*: *deviation-def*)

lemma *deviation-in-profiles*:
assumes *s* ∈ *profiles i* ∈ *players x* ∈ *strategies i*
shows *deviation s i x* ∈ *profiles*
using *assms* **by** (*auto simp*: *profiles-iff*)

lemma *Nash-equilibriumI*:
assumes *s* ∈ *profiles*
and ⋀*i x*. *i* ∈ *players* ⟹ *x* ∈ *strategies i* ⟹
payoff i (*deviation s i x*) ≤ *payoff i s*
shows *Nash-equilibrium s*
using *assms* **by** (*auto simp*: *Nash-equilibrium-def*)

lemma *Nash-equilibrium-profile*:
assumes *Nash-equilibrium s*
shows *s* ∈ *profiles*

```
  using assms by (simp add: Nash-equilibrium-def)

lemma Nash-equilibriumD:
  assumes Nash-equilibrium s i ∈ players x ∈ strategies i
  shows payoff i (deviation s i x) ≤ payoff i s
  using assms by (simp add: Nash-equilibrium-def)

lemma Nash-equilibrium-iff-best-responses:
  assumes s ∈ profiles
  shows Nash-equilibrium s ⟷ (∀ i∈players. best-response-to s i (s i))
 using assms by (auto simp: Nash-equilibrium-def best-response-to-def profile-strategy)

lemma dominant-strategy-profile-is-Nash:
  assumes s ∈ profiles
    and dominant: ⋀i. i ∈ players ⟹ dominant-strategy i (s i)
  shows Nash-equilibrium s
proof (rule Nash-equilibriumI)
  show s ∈ profiles
    using assms by simp
next
  fix i x
  assume ix: i ∈ players x ∈ strategies i
  have dom: dominant-strategy i (s i)
    using dominant ix by blast
  then have ⋀t y. t ∈ profiles ⟹ y ∈ strategies i ⟹
      payoff i (deviation t i y) ≤ payoff i (deviation t i (s i))
    by (simp add: dominant-strategy-def)
  from this[OF assms(1) ix(2)] show payoff i (deviation s i x) ≤ payoff i s
    by simp
qed

end
```

2 Existence for Finite Potential Games

Pure Nash equilibria need not exist in arbitrary finite games. A standard positive result is that every finite game with an ordinal potential has a pure Nash equilibrium: choose a profile whose potential is maximal. The definition used here is the one needed for the argument: every strict unilateral payoff improvement strictly increases the potential. Exact and ordinal potential games in the sense of Monderer and Shapley satisfy this assumption.

```
locale finite-potential-game =
  finite-game players strategies payoff
  for players :: 'p::finite set
    and strategies :: 'p ⇒ 's::finite set
    and payoff :: 'p ⇒ ('p ⇒ 's) ⇒ 'u::linorder +
  fixes potential :: ('p ⇒ 's) ⇒ 'v::linorder
  assumes potential-increases:
```

```
    ⟦s ∈ profiles; i ∈ players; x ∈ strategies i;
      payoff i s < payoff i (deviation s i x)⟧
     ⟹ potential s < potential (deviation s i x)
begin

lemma maximal-potential-profile:
  obtains s where
    s ∈ profiles
    ⋀t. t ∈ profiles ⟹ potential t ≤ potential s
proof −
  let ?M = Max (potential ‘ profiles)
  have fin: finite (potential ‘ profiles)
    by simp
  have nonempty: potential ‘ profiles ≠ {}
    using profiles-nonempty by blast
  have M-in: ?M ∈ potential ‘ profiles
    by (rule Max-in[OF fin nonempty])
  then obtain s where s-prof: s ∈ profiles and s-Max: potential s = ?M
    by (auto elim!: imageE)
  have max-s: potential t ≤ potential s if t ∈ profiles for t
    using Max-ge[OF fin, of potential t] s-Max that by auto
  show ?thesis
    using max-s s-prof that by auto
qed

theorem exists-Nash-equilibrium:
  ∃ s∈profiles. Nash-equilibrium s
proof −
  obtain s where s: s ∈ profiles
    and max: ⋀t. t ∈ profiles ⟹ potential t ≤ potential s
    using maximal-potential-profile by blast
  have Nash-equilibrium s
  proof (rule Nash-equilibriumI)
    show s ∈ profiles
      by fact
  next
    fix i x
    assume ix: i ∈ players x ∈ strategies i
    show payoff i (deviation s i x) ≤ payoff i s
    proof (rule ccontr)
      assume ¬ payoff i (deviation s i x) ≤ payoff i s
      then have improve: payoff i s < payoff i (deviation s i x)
        by simp
      have dev: deviation s i x ∈ profiles
        using s ix by (rule deviation-in-profiles)
      have potential s < potential (deviation s i x)
        using potential-increases[OF s ix improve] .
      moreover have potential (deviation s i x) ≤ potential s
        using max[OF dev] .
```

```
      ultimately show False
        by simp
    qed
  qed
  with s show ?thesis
    by blast
qed

end
```

3 Dominant Strategies as a Degenerate Existence Result

Dominant strategies give another elementary source of equilibria. The next locale packages the hypothesis that every player has a distinguished dominant strategy and derives existence by constructing the corresponding profile.

```
locale finite-dominant-strategy-game =
  finite-game players strategies payoff
  for players :: 'p::finite set
    and strategies :: 'p ⇒ 's::finite set
    and payoff :: 'p ⇒ ('p ⇒ 's) ⇒ 'u::preorder +
  fixes dominant :: 'p ⇒ 's
  assumes dominant: i ∈ players ⟹ dominant-strategy i (dominant i)
begin

definition dominant-profile :: 'p ⇒ 's where
  dominant-profile i = (if i ∈ players then dominant i else undefined)

lemma dominant-profile-in-profiles:
  dominant-profile ∈ profiles
 using dominant by (auto simp: dominant-profile-def profiles-iff dominant-strategy-def)

theorem dominant-profile-is-Nash:
  Nash-equilibrium dominant-profile
  by (simp add: dominant dominant-profile-def dominant-profile-in-profiles dominant-strategy-profile-is-Nash)

end
```

4 Matching Pennies

The following two-player zero-sum game shows why an existence theorem for pure Nash equilibria needs additional hypotheses. The row player wants the two coins to match; the column player wants them to differ. After every pure profile, exactly one player can improve by switching sides.

datatype *penny-player* = *Row* | *Column*

datatype *coin-side* = *Heads* | *Tails*

instantiation *penny-player* :: *finite*
begin

instance
proof
  show *finite* (*UNIV* :: *penny-player set*)
    by (*rule finite-subset*[*of* - {*Row*, *Column*}]) (*auto intro*: *penny-player*.*exhaust*)
qed

end

instantiation *coin-side* :: *finite*
begin

instance
proof
  show *finite* (*UNIV* :: *coin-side set*)
    by (*rule finite-subset*[*of* - {*Heads*, *Tails*}]) (*auto intro*: *coin-side*.*exhaust*)
qed

end

definition *matching-pennies-payoff* :: *penny-player* ⇒ (*penny-player* ⇒ *coin-side*) ⇒ *int* **where**
  matching-pennies-payoff p s =
    (*case p of*
       Row ⇒ *if s Row* = *s Column then 1 else 0*
     | *Column* ⇒ *if s Row* = *s Column then 0 else 1*)

interpretation *matching-pennies*:
  finite-game UNIV λ-. *UNIV matching-pennies-payoff*
  by *standard auto*

definition *switch-coin* :: *coin-side* ⇒ *coin-side* **where**
  switch-coin x = (*case x of Heads* ⇒ *Tails* | *Tails* ⇒ *Heads*)

lemma *switch-coin-neq* [*simp*]: *switch-coin x* ≠ *x*
  by (*cases x*) (*simp-all add*: *switch-coin-def*)

lemma *coin-side-switch*:
  fixes *x* :: *coin-side*
  obtains *y* **where** *y* ≠ *x*
proof
  show *switch-coin x* ≠ *x*
    by *simp*

```
qed

lemma matching-pennies-no-pure-Nash:
  ¬ matching-pennies.Nash-equilibrium s
proof
  assume ne: matching-pennies.Nash-equilibrium s
  then have profile: s ∈ matching-pennies.profiles
    by (rule matching-pennies.Nash-equilibrium-profile)
  show False
  proof (cases s Row = s Column)
    case True
    obtain y where y: y ≠ s Row
      using coin-side-switch by blast
     have matching-pennies-payoff Column (matching-pennies.deviation s Column
y) = 1
      using True y by (simp add: matching-pennies-payoff-def)
    moreover have matching-pennies-payoff Column s = 0
      using True by (simp add: matching-pennies-payoff-def)
   ultimately have ¬ matching-pennies-payoff Column (matching-pennies.deviation
s Column y)
                    ≤ matching-pennies-payoff Column s
      by simp
    moreover have y ∈ (UNIV :: coin-side set)
      by simp
    ultimately show False
      using matching-pennies.Nash-equilibriumD[OF ne, of Column y] by simp
  next
    case False
    obtain y where y: y = s Column
      by simp
    have matching-pennies-payoff Row (matching-pennies.deviation s Row y) = 1
      using y by (simp add: matching-pennies-payoff-def)
    moreover have matching-pennies-payoff Row s = 0
      using False by (simp add: matching-pennies-payoff-def)
    ultimately have ¬ matching-pennies-payoff Row (matching-pennies.deviation
s Row y)
                    ≤ matching-pennies-payoff Row s
      by simp
    moreover have y ∈ (UNIV :: coin-side set)
      by simp
    ultimately show False
      using matching-pennies.Nash-equilibriumD[OF ne, of Row y] by simp
  qed
qed

end
theory Mixed-Nash-Equilibrium
  imports
    Nash-Equilibrium
```

HOL−Analysis.Cartesian-Euclidean-Space
HOL−Analysis.Brouwer-Fixpoint
begin

5 Mixed Nash Equilibria

This theory develops the mixed-strategy version of Nash equilibrium for finite games whose players and pure strategies are represented by finite HOL types. A mixed profile is a Cartesian vector indexed by player/strategy pairs.

This locale is deliberately more restrictive than the pure-game locale: every player has the same finite pure-strategy type. In return, the profile space is a finite Cartesian product of real coordinates, so compactness, convexity, continuity, and Brouwer's fixed point theorem can be applied directly.

type-synonym ('p, 's) *mixed-profile* = *real* ^ ('p × 's)

locale *finite-type-game* =
  fixes *payoff* :: 'p::*finite* ⇒ ('p ⇒ 's::*finite*) ⇒ *real*
begin

definition *prob* :: ('p, 's) *mixed-profile* ⇒ 'p ⇒ 's ⇒ *real* **where**
  prob m i x = *m* $ (*i*, *x*)

definition *mixed-profiles* :: ('p, 's) *mixed-profile set* **where**
  mixed-profiles =
    {*m* ∈ *cbox 0 1*. ∀ *i*. (∑ *x*∈*UNIV*. *prob m i x*) = *1*}

definition *uniform-mixed-profile* :: ('p, 's) *mixed-profile* **where**
  uniform-mixed-profile = (χ *ix*. *1* / *real CARD*('s))

definition *opponent-weight* :: 'p ⇒ ('p, 's) *mixed-profile* ⇒ ('p ⇒ 's) ⇒ *real* **where**
  opponent-weight i m s = (∏ *j*∈*UNIV* − {*i*}. *prob m j* (*s j*))

definition *pure-deviation-payoff* ::
    'p ⇒ 's ⇒ ('p, 's) *mixed-profile* ⇒ *real* **where**
  pure-deviation-payoff i x m =
    (∑ *s*∈{*s*. *s i* = *x*}. *opponent-weight i m s* ∗ *payoff i s*)

definition *mixed-payoff* :: 'p ⇒ ('p, 's) *mixed-profile* ⇒ *real* **where**
  mixed-payoff i m =
    (∑ *x*∈*UNIV*. *prob m i x* ∗ *pure-deviation-payoff i x m*)

definition *mixed-Nash-equilibrium* :: ('p, 's) *mixed-profile* ⇒ *bool* **where**
  mixed-Nash-equilibrium m ⟷
    m ∈ *mixed-profiles* ∧
    (∀ *i x*. *pure-deviation-payoff i x m* ≤ *mixed-payoff i m*)

definition *excess* :: 'p ⇒ 's ⇒ ('p, 's) *mixed-profile* ⇒ *real* **where**

excess i x m = *max 0* (*pure-deviation-payoff i x m* − *mixed-payoff i m*)

definition *excess-sum* :: *'p* ⇒ (*'p*, *'s*) *mixed-profile* ⇒ *real* **where**
excess-sum i m = ($\sum$ *x*∈*UNIV*. *excess i x m*)

definition *nash-map* :: (*'p*, *'s*) *mixed-profile* ⇒ (*'p*, *'s*) *mixed-profile* **where**
nash-map m = (χ *ix*. (*prob m* (*fst ix*) (*snd ix*) + *excess* (*fst ix*) (*snd ix*) *m*) /
(*1* + *excess-sum* (*fst ix*) *m*))

lemma *prob-nash-map*:
prob (*nash-map m*) *i x* = (*prob m i x* + *excess i x m*) / (*1* + *excess-sum i m*)
by (*simp add*: *prob-def nash-map-def*)

lemma *mixed-profiles-prob-nonneg*:
assumes *m* ∈ *mixed-profiles*
shows *0* ≤ *prob m i x*
using *assms* **by** (*auto simp*: *mixed-profiles-def prob-def mem-box-cart*)

lemma *mixed-profiles-prob-le-one*:
assumes *m* ∈ *mixed-profiles*
shows *prob m i x* ≤ *1*
using *assms* **by** (*auto simp*: *mixed-profiles-def prob-def mem-box-cart*)

lemma *mixed-profiles-sum-prob*:
assumes *m* ∈ *mixed-profiles*
shows ($\sum$ *x*∈*UNIV*. *prob m i x*) = *1*
using *assms* **by** (*auto simp*: *mixed-profiles-def*)

lemma *prob-uniform-mixed-profile* [*simp*]:
prob uniform-mixed-profile i x = *1* / *real CARD*(*'s*)
by (*simp add*: *prob-def uniform-mixed-profile-def*)

lemma *uniform-mixed-profile-in-mixed-profiles* [*simp*]:
uniform-mixed-profile ∈ *mixed-profiles*
proof −
have *card-pos*: *0* < *real CARD*(*'s*)
by *simp*
have *prob-le-one*: *1* / *real CARD*(*'s*) ≤ *1*
using *card-pos* **by** (*simp add*: *divide-simps*)
have *sum-one*: ($\sum$ *x*∈*UNIV*. *prob uniform-mixed-profile i x*) = *1* **for** *i*
using *card-pos* **by** *simp*
show *?thesis*
using *card-pos prob-le-one sum-one*
by (*auto simp*: *mixed-profiles-def prob-def uniform-mixed-profile-def mem-box-cart*)
qed

lemma *excess-nonneg* [*simp*]: *0* ≤ *excess i x m*
by (*simp add*: *excess-def*)

```
lemma excess-sum-nonneg [simp]: 0 ≤ excess-sum i m
  by (simp add: excess-sum-def sum-nonneg)

lemma denom-pos [simp]: 0 < 1 + excess-sum i m
  using excess-sum-nonneg[of i m] by linarith

lemma nash-map-in-mixed-profiles:
  assumes m: m ∈ mixed-profiles
  shows nash-map m ∈ mixed-profiles
proof −
  have nonneg: 0 ≤ prob (nash-map m) i x for i x
    using mixed-profiles-prob-nonneg[OF m, of i x]
    by (simp add: prob-nash-map divide-nonneg-pos)
  have le-one: prob (nash-map m) i x ≤ 1 for i x
  proof −
    have ex-le: excess i x m ≤ excess-sum i m
      unfolding excess-sum-def by (intro member-le-sum) auto
    have prob m i x + excess i x m ≤ 1 + excess-sum i m
      using mixed-profiles-prob-le-one[OF m, of i x] ex-le by linarith
    then show ?thesis
      by (simp add: prob-nash-map field-simps)
  qed
  have sum-one: (∑x∈UNIV. prob (nash-map m) i x) = 1 for i
  proof −
    have (∑x∈UNIV. prob (nash-map m) i x) =
        (∑x∈UNIV. (prob m i x + excess i x m) / (1 + excess-sum i m))
      by (simp add: prob-nash-map)
    also have ... =
        ((∑x∈UNIV. prob m i x) + excess-sum i m) / (1 + excess-sum i m)
      by (simp add: sum-divide-distrib[symmetric] sum.distrib excess-sum-def)
    also have ... = 1
      by (simp add: add-nonneg-eq-0-iff m mixed-profiles-sum-prob)
    finally show ?thesis .
  qed
  show ?thesis
    using nonneg le-one sum-one
    by (auto simp: mixed-profiles-def prob-def mem-box-cart)
qed

lemma continuous-prob:
  continuous-on S (λm. prob m i x)
  by (simp add: prob-def) (intro continuous-intros)

lemma continuous-opponent-weight:
  continuous-on S (λm. opponent-weight i m s)
  unfolding opponent-weight-def
  by (intro continuous-intros continuous-prob)

lemma continuous-pure-deviation-payoff:
```

continuous-on S (λ*m. pure-deviation-payoff i x m*)
unfolding *pure-deviation-payoff-def*
by (*intro continuous-intros continuous-opponent-weight*)

lemma *continuous-mixed-payoff*:
continuous-on S (λ*m. mixed-payoff i m*)
unfolding *mixed-payoff-def*
by (*intro continuous-intros continuous-prob continuous-pure-deviation-payoff*)

lemma *continuous-excess*:
continuous-on S (λ*m. excess i x m*)
unfolding *excess-def*
by (*intro continuous-intros continuous-pure-deviation-payoff continuous-mixed-payoff*)

lemma *continuous-excess-sum*:
continuous-on S (λ*m. excess-sum i m*)
unfolding *excess-sum-def*
by (*intro continuous-intros continuous-excess*)

lemma *continuous-nash-map*:
continuous-on mixed-profiles nash-map
proof −
have *nz*: $m \in$ *mixed-profiles* $\Longrightarrow$ *1* + *excess-sum* (*fst ix*) *m* $\neq$ *0* **for** *ix m*
using *denom-pos*[*of fst ix m*] **by** *linarith*
show *?thesis*
unfolding *nash-map-def*
apply (*intro continuous-intros continuous-prob continuous-excess continuous-excess-sum*)
using *nz* **by** *blast*
qed

lemma *mixed-profiles-closed*: *closed mixed-profiles*
proof −
have *closed-constraint*: *closed* {*m.* ($\sum$ *x*$\in$*UNIV. prob m i x*) = *1*} **for** *i*
by (*intro closed-Collect-eq continuous-intros continuous-prob*)
have *eq*: *mixed-profiles* =
cbox 0 1 $\cap$ ($\bigcap$ *i*$\in$(*UNIV* :: *'p set*). {*m.* ($\sum$ *x*$\in$*UNIV. prob m i x*) = *1*})
by (*auto simp*: *mixed-profiles-def*)
show *?thesis*
unfolding *eq* **using** *closed-constraint* **by** (*auto intro*!: *closed-Int closed-INT closed-cbox*)
qed

lemma *mixed-profiles-compact*: *compact mixed-profiles*
proof −
have *mixed-profiles* $\subseteq$ *cbox 0 1*
by (*auto simp*: *mixed-profiles-def*)
then show *?thesis*
using *compact-cbox mixed-profiles-closed compact-eq-bounded-closed bounded-cbox bounded-subset*

```
    by blast
qed

lemma mixed-profiles-convex: convex mixed-profiles
proof (rule convexI)
  fix m n :: ('p, 's) mixed-profile
  fix u v :: real
  assume mn: m ∈ mixed-profiles n ∈ mixed-profiles
    and uv: 0 ≤ u 0 ≤ v u + v = 1
  have in-box: u *R m + v *R n ∈ cbox 0 1
  proof (auto simp: mem-box-cart)
    fix a b
    have 0 ≤ m $ (a, b) 0 ≤ n $ (a, b)
      using mn by (auto simp: mixed-profiles-def mem-box-cart)
    then show 0 ≤ u ∗ m $ (a, b) + v ∗ n $ (a, b)
      using uv by (intro add-nonneg-nonneg mult-nonneg-nonneg) auto
  next
    fix a b
    have m-le: m $ (a, b) ≤ 1 and n-le: n $ (a, b) ≤ 1
      using mn by (auto simp: mixed-profiles-def mem-box-cart)
    have u ∗ m $ (a, b) ≤ u
      using uv m-le by (metis mult.right-neutral mult-left-mono)
    moreover have v ∗ n $ (a, b) ≤ v
      using uv n-le by (metis mult.right-neutral mult-left-mono)
    ultimately show u ∗ m $ (a, b) + v ∗ n $ (a, b) ≤ 1
      using uv by linarith
  qed
  have sum-one: (∑x∈UNIV. prob (u *R m + v *R n) i x) = 1 for i
  proof −
    have (∑x∈UNIV. prob (u *R m + v *R n) i x) =
        u ∗ (∑x∈UNIV. prob m i x) + v ∗ (∑x∈UNIV. prob n i x)
      by (simp add: prob-def sum.distrib sum-distrib-left)
    also have ... = 1
      by (simp add: mixed-profiles-sum-prob mn uv)
    finally show ?thesis .
  qed
  show u *R m + v *R n ∈ mixed-profiles
    using in-box sum-one by (auto simp: mixed-profiles-def)
qed

lemma mixed-profiles-nonempty: mixed-profiles ≠ {}
  using uniform-mixed-profile-in-mixed-profiles by blast

lemma mixed-Nash-equilibrium-profile:
  assumes mixed-Nash-equilibrium m
  shows m ∈ mixed-profiles
  using assms by (simp add: mixed-Nash-equilibrium-def)

lemma mixed-Nash-equilibriumD:
```

```
  assumes mixed-Nash-equilibrium m
  shows pure-deviation-payoff i x m ≤ mixed-payoff i m
  using assms by (simp add: mixed-Nash-equilibrium-def)

lemma mixed-Nash-support-payoff-eq:
  assumes ne: mixed-Nash-equilibrium m and px: prob m i x > 0
  shows pure-deviation-payoff i x m = mixed-payoff i m
proof −
  have m: m ∈ mixed-profiles
    using ne by (rule mixed-Nash-equilibrium-profile)
  have gap-nonneg:
    0 ≤ prob m i y ∗ (mixed-payoff i m − pure-deviation-payoff i y m) for y
     using mixed-profiles-prob-nonneg[OF m, of i y] mixed-Nash-equilibriumD[OF
ne, of i y]
    by (intro mult-nonneg-nonneg) auto
   have (∑ y∈UNIV. prob m i y ∗ (mixed-payoff i m − pure-deviation-payoff i y
m)) =
        (∑ y∈UNIV. prob m i y ∗ mixed-payoff i m) −
        (∑ y∈UNIV. prob m i y ∗ pure-deviation-payoff i y m)
    by (simp add: algebra-simps sum-subtractf)
  also have . . . =
      (∑ y∈UNIV. prob m i y) ∗ mixed-payoff i m − mixed-payoff i m
    unfolding mixed-payoff-def by (simp add: sum-distrib-right)
  also have . . . =
      mixed-payoff i m ∗ (∑ y∈UNIV. prob m i y) − mixed-payoff i m
    by (simp add: mult.commute)
  also have . . . = 0
    using mixed-profiles-sum-prob[OF m, of i] by simp
  finally have gaps-zero:
    ⋀y. y ∈ UNIV =⇒
      prob m i y ∗ (mixed-payoff i m − pure-deviation-payoff i y m) = 0
    using gap-nonneg by (simp add: sum-nonneg-eq-0-iff)
  have mixed-payoff i m − pure-deviation-payoff i x m = 0
    using gaps-zero[of x] px by simp
  then show ?thesis
    by simp
qed

lemma mixed-Nash-zero-probability-if-less:
  assumes ne: mixed-Nash-equilibrium m
    and less: pure-deviation-payoff i x m < mixed-payoff i m
  shows prob m i x = 0
proof (rule ccontr)
  assume prob m i x ≠ 0
  moreover have 0 ≤ prob m i x
   using mixed-Nash-equilibrium-profile[OF ne] by (rule mixed-profiles-prob-nonneg)
  ultimately have pure-deviation-payoff i x m = mixed-payoff i m
    using mixed-Nash-support-payoff-eq[OF ne] by simp
  then show False
```

```
    using less by simp
qed

definition dirac-mixed-profile :: ('p ⇒ 's) ⇒ ('p, 's) mixed-profile where
  dirac-mixed-profile s = (χ ix. if snd ix = s (fst ix) then 1 else 0)

lemma prob-dirac-mixed-profile [simp]:
  prob (dirac-mixed-profile s) i x = (if x = s i then 1 else 0)
  by (simp add: prob-def dirac-mixed-profile-def)

lemma dirac-mixed-profile-in-mixed-profiles [simp]:
  dirac-mixed-profile s ∈ mixed-profiles
  by (auto simp: mixed-profiles-def prob-def dirac-mixed-profile-def mem-box-cart)

lemma opponent-weight-dirac-mixed-profile:
  opponent-weight i (dirac-mixed-profile s) t =
    (if ∀ j. j ≠ i ⟶ t j = s j then 1 else 0)
proof (cases ∀ j. j ≠ i ⟶ t j = s j)
  case True
  then show ?thesis
    by (auto simp: opponent-weight-def)
next
  case False
  then obtain j where j: j ≠ i t j ≠ s j
    by blast
  have j-in: j ∈ UNIV − {i}
    using j by simp
  have j-zero: prob (dirac-mixed-profile s) j (t j) = 0
    using j by simp
  have zero-ex: ∃ k∈UNIV − {i}. prob (dirac-mixed-profile s) k (t k) = 0
    using j-in j-zero by blast
  have fin: finite (UNIV − {i})
    by simp
  have (∏ j∈UNIV − {i}. prob (dirac-mixed-profile s) j (t j)) = 0
    by (rule prod-zero[OF fin zero-ex])
  then show ?thesis
    using False by (simp add: opponent-weight-def)
qed

lemma pure-deviation-payoff-dirac-mixed-profile:
  pure-deviation-payoff i x (dirac-mixed-profile s) = payoff i (s(i := x))
proof −
  have term-eq:
    ⋀t. t ∈ {t. t i = x} ⟹
      opponent-weight i (dirac-mixed-profile s) t ∗ payoff i t =
        (if t = s(i := x) then payoff i (s(i := x)) else 0)
  proof −
    fix t
    assume t-in: t ∈ {t. t i = x}
```

```
    show opponent-weight i (dirac-mixed-profile s) t * payoff i t =
        (if t = s(i := x) then payoff i (s(i := x)) else 0)
    proof (cases ∀ j. j ≠ i ⟶ t j = s j)
      case True
      then have t = s(i := x)
        using t-in by (auto simp: fun-eq-iff)
      then show ?thesis
        using True by (simp add: opponent-weight-dirac-mixed-profile)
    next
      case False
      then have t ≠ s(i := x)
        by (auto simp: fun-eq-iff)
      moreover have opponent-weight i (dirac-mixed-profile s) t = 0
        using False by (simp add: opponent-weight-dirac-mixed-profile)
      then show ?thesis
        using calculation by simp
    qed
  qed
  have pure-deviation-payoff i x (dirac-mixed-profile s) =
      (∑ t∈{t. t i = x}. if t = s(i := x) then payoff i (s(i := x)) else 0)
    unfolding pure-deviation-payoff-def
    by (intro sum.cong) (auto simp: term-eq)
  also have ... = payoff i (s(i := x))
    by simp
  finally show ?thesis .
qed

lemma mixed-payoff-dirac-mixed-profile:
  mixed-payoff i (dirac-mixed-profile s) = payoff i s
proof –
  have mixed-payoff i (dirac-mixed-profile s) =
      (∑ x∈UNIV. if x = s i then payoff i (s(i := x)) else 0)
    unfolding mixed-payoff-def
    by (intro sum.cong) (simp-all add: pure-deviation-payoff-dirac-mixed-profile)
  also have ... = payoff i s
    by simp
  finally show ?thesis .
qed

lemma dirac-mixed-Nash-equilibrium:
  assumes ⋀i x. payoff i (s(i := x)) ≤ payoff i s
  shows mixed-Nash-equilibrium (dirac-mixed-profile s)
  using assms
  by (auto simp: mixed-Nash-equilibrium-def
      pure-deviation-payoff-dirac-mixed-profile mixed-payoff-dirac-mixed-profile)

lemma fixed-point-imp-excess-zero:
  assumes m: m ∈ mixed-profiles and fp: nash-map m = m
  shows excess i x m = 0
```

```
proof −
  let ?G = excess-sum i m
  have ex-eq: excess i y m = prob m i y ∗ ?G for y
  proof −
    have eq: prob m i y = (prob m i y + excess i y m) / (1 + ?G)
      using arg-cong[OF fp, of λm. prob m i y] by (simp add: prob-nash-map)
    then have mult-eq: prob m i y ∗ (1 + ?G) = prob m i y + excess i y m
      using denom-pos[of i m] by (simp add: field-simps)
    then show excess i y m = prob m i y ∗ ?G
      by (simp add: algebra-simps)
  qed
  show ?thesis
  proof (cases ?G = 0)
    case True
    then show ?thesis
      using ex-eq[of x] by simp
  next
    case False
    then have G-pos: ?G > 0
      using excess-sum-nonneg[of i m] by linarith
    have pos-imp:
      pure-deviation-payoff i x m = mixed-payoff i m + prob m i x ∗ ?G
      if prob m i x > 0 for x
    proof −
      have ex-pos: excess i x m > 0
        using that G-pos ex-eq[of x] by simp
      then have pure-deviation-payoff i x m − mixed-payoff i m = excess i x m
        by (simp add: excess-def)
      then show ?thesis
        using ex-eq[of x] by simp
    qed
    have payoff-eq:
      prob m i y ∗ pure-deviation-payoff i y m =
        prob m i y ∗ mixed-payoff i m + ?G ∗ (prob m i y)^2 for y
    proof (cases prob m i y = 0)
      case True
      then show ?thesis
        by simp
    next
      case False
      have 0 ≤ prob m i y
        by (rule mixed-profiles-prob-nonneg[OF m])
      with False have prob m i y > 0
        by simp
      then show ?thesis
        using pos-imp[of y] by (simp add: algebra-simps power2-eq-square)
    qed
    have payoff-average:
      mixed-payoff i m =
```

```
    mixed-payoff i m ∗ (∑ y∈UNIV. prob m i y) +
    ?G ∗ (∑ y∈UNIV. (prob m i y)^2)
proof −
  have mixed-payoff i m =
      (∑ y∈UNIV. prob m i y ∗ pure-deviation-payoff i y m)
    by (simp add: mixed-payoff-def)
  also have ... =
      (∑ y∈UNIV.
        prob m i y ∗ mixed-payoff i m + ?G ∗ (prob m i y)^2)
    by (intro sum.cong) (simp-all add: payoff-eq)
  also have ... =
      (∑ y∈UNIV. prob m i y ∗ mixed-payoff i m) +
      (∑ y∈UNIV. ?G ∗ (prob m i y)^2)
    by (simp add: sum.distrib)
  also have ... =
      (∑ y∈UNIV. prob m i y) ∗ mixed-payoff i m +
      ?G ∗ (∑ y∈UNIV. (prob m i y)^2)
    by (simp add: sum-distrib-left sum-distrib-right)
  also have ... =
      mixed-payoff i m ∗ (∑ y∈UNIV. prob m i y) +
      ?G ∗ (∑ y∈UNIV. (prob m i y)^2)
    by (simp add: mult.commute)
  finally show ?thesis .
qed
have mixed-payoff i m =
    mixed-payoff i m + ?G ∗ (∑ y∈UNIV. (prob m i y)^2)
  using payoff-average mixed-profiles-sum-prob[OF m, of i] by simp
then have G-squares-zero: ?G ∗ (∑ y∈UNIV. (prob m i y)^2) = 0
  by simp
have sq-zero: (∑ y∈UNIV. (prob m i y)^2) = 0
  using G-pos G-squares-zero by simp
have ∃ x. prob m i x > 0
proof (rule ccontr)
  assume no-pos: ¬ (∃ x. prob m i x > 0)
  have zero: prob m i x = 0 for x
    by (smt (verit) m mixed-profiles-prob-nonneg no-pos)
  have (∑ x∈UNIV. prob m i x) = 0
    by (simp add: zero)
  then show False
    using mixed-profiles-sum-prob[OF m, of i] by simp
qed
then obtain y where y: prob m i y > 0
  by blast
have 0 < (prob m i y)^2
  using y by simp
moreover have (prob m i y)^2 ≤ (∑ x∈UNIV. (prob m i x)^2)
  by (intro member-le-sum) auto
ultimately show ?thesis
  using sq-zero by simp
```

  qed
qed

theorem *exists-mixed-Nash-equilibrium*:
  $\exists$ *m*$\in$*mixed-profiles*. *mixed-Nash-equilibrium m*
proof −
  obtain *m* **where** *m*: *m* $\in$ *mixed-profiles* **and** *fp*: *nash-map m* = *m*
  proof (*rule brouwer*[
      OF mixed-profiles-compact mixed-profiles-convex mixed-profiles-nonempty continuous-nash-map])
    show *nash-map* $\in$ *mixed-profiles* $\rightarrow$ *mixed-profiles*
      using *nash-map-in-mixed-profiles* **by** *auto*
  qed
  have *no-excess*: *excess i x m* = *0* **for** *i x*
    by (*rule fixed-point-imp-excess-zero*[*OF m fp*])
  have *deviation-le*: *pure-deviation-payoff i x m* $\leq$ *mixed-payoff i m* **for** *i x*
  proof −
    have *pure-deviation-payoff i x m* − *mixed-payoff i m*
        $\leq$ *max 0* (*pure-deviation-payoff i x m* − *mixed-payoff i m*)
      by *simp*
    also have ... = *0*
      using *no-excess*[*of i x*] **by** (*simp add*: *excess-def*)
    finally show *?thesis*
      by *simp*
  qed
  have *mixed-Nash-equilibrium m*
    using *m deviation-le* **by** (*auto simp*: *mixed-Nash-equilibrium-def*)
  then show *?thesis*
    using *m* **by** *blast*
qed

end

end
theory *Nash-Equilibrium-Examples*
  imports *Mixed-Nash-Equilibrium*
begin

lemma *UNIV-penny-player*: (*UNIV* :: *penny-player set*) = {*Row*, *Column*}
  by (*auto intro*: *penny-player.exhaust*)

lemma *UNIV-coin-side* [*simp*]: (*UNIV* :: *coin-side set*) = {*Heads*, *Tails*}
  by (*auto intro*: *coin-side.exhaust*)

lemma *players-except-Row* [*simp*]: (*UNIV* :: *penny-player set*) − {*Row*} = {*Column*}
  by (*auto simp*: *UNIV-penny-player*)

lemma *players-except-Column* [*simp*]: (*UNIV* :: *penny-player set*) − {*Column*} = {*Row*}

by (*auto simp*: *UNIV-penny-player*)

lemma *player-insert-except-Row* [*simp*]: {*Row*, *Column*} − {*Row*} = {*Column*}
by *auto*

lemma *player-insert-except-Column* [*simp*]: {*Row*, *Column*} − {*Column*} = {*Row*}
by *auto*

6 Prisoner's Dilemma

The Prisoner's Dilemma gives a small example using the dominant-strategy existence result. Defection is dominant for both players, hence the all-defect profile is a pure Nash equilibrium.

datatype *prisoner* = *Prisoner1* | *Prisoner2*

datatype *prisoner-move* = *Cooperate* | *Defect*

instantiation *prisoner* :: *finite*
begin

instance
proof
show *finite* (*UNIV* :: *prisoner set*)
by (*rule finite-subset*[*of* - {*Prisoner1*, *Prisoner2*}]) (*auto intro*: *prisoner.exhaust*)
qed

end

instantiation *prisoner-move* :: *finite*
begin

instance
proof
show *finite* (*UNIV* :: *prisoner-move set*)
by (*rule finite-subset*[*of* - {*Cooperate*, *Defect*}]) (*auto intro*: *prisoner-move.exhaust*)
qed

end

fun *other-prisoner* :: *prisoner* ⇒ *prisoner* **where**
other-prisoner Prisoner1 = *Prisoner2*
| *other-prisoner Prisoner2* = *Prisoner1*

definition *prisoners-dilemma-payoff* :: *prisoner* ⇒ (*prisoner* ⇒ *prisoner-move*) ⇒ *int* **where**
prisoners-dilemma-payoff p s =
(*case* (*s p*, *s* (*other-prisoner p*)) *of*
(*Cooperate*, *Cooperate*) ⇒ *3*

| (*Defect*, *Cooperate*) ⇒ *5*
| (*Cooperate*, *Defect*) ⇒ *0*
| (*Defect*, *Defect*) ⇒ *1*)

interpretation *prisoners-dilemma*:
finite-game UNIV λ-. UNIV prisoners-dilemma-payoff
by *standard auto*

interpretation *prisoners-dilemma-dominant*:
finite-dominant-strategy-game UNIV λ-. UNIV prisoners-dilemma-payoff λ-. Defect
proof
fix *i* :: *prisoner*
show *prisoners-dilemma.dominant-strategy i* ((λ-. *Defect*) *i*)
by (*cases i*)
(*auto simp*: *prisoners-dilemma.dominant-strategy-def prisoners-dilemma-payoff-def split*: *prisoner-move.splits*)
qed

lemma *prisoners-dilemma-defect-defect-Nash*:
prisoners-dilemma.Nash-equilibrium (λ-. *Defect*)
proof −
have *prisoners-dilemma-dominant.dominant-profile* = (λ-. *Defect*)
by (*simp add*: *fun-eq-iff prisoners-dilemma-dominant.dominant-profile-def*)
then show *?thesis*
using *prisoners-dilemma-dominant.dominant-profile-is-Nash* **by** *simp*
qed

7 Coordination Game

A two-player coordination game has two pure equilibria. Both players receive payoff one when their choices agree and zero otherwise.

datatype *coordination-choice* = *Choice-A* | *Choice-B*

instantiation *coordination-choice* :: *finite*
begin

instance
proof
show *finite* (*UNIV* :: *coordination-choice set*)
by (*rule finite-subset*[*of* - {*Choice-A*, *Choice-B*}])
(*auto intro*: *coordination-choice.exhaust*)
qed

end

definition *coordination-payoff* ::
penny-player ⇒ (*penny-player* ⇒ *coordination-choice*) ⇒ *int* **where**

coordination-payoff p s = (*if s Row* = *s Column then 1 else 0*)

interpretation *coordination*:
finite-game UNIV λ-. *UNIV coordination-payoff*
by *standard auto*

lemma *coordination-A-A-Nash*:
coordination.Nash-equilibrium (λ-. *Choice-A*)
proof (*rule coordination.Nash-equilibriumI*)
show (λ-. *Choice-A*) ∈ *coordination.profiles*
by (*simp add*: *coordination.profiles-def*)
next
fix *i* :: *penny-player*
fix *x* :: *coordination-choice*
show *coordination-payoff i* (*coordination.deviation* (λ-. *Choice-A*) *i x*)
≤ *coordination-payoff i* (λ-. *Choice-A*)
by (*cases i*; *cases x*) (*simp-all add*: *coordination-payoff-def coordination.deviation-def*)
qed

lemma *coordination-B-B-Nash*:
coordination.Nash-equilibrium (λ-. *Choice-B*)
proof (*rule coordination.Nash-equilibriumI*)
show (λ-. *Choice-B*) ∈ *coordination.profiles*
by (*simp add*: *coordination.profiles-def*)
next
fix *i* :: *penny-player*
fix *x* :: *coordination-choice*
show *coordination-payoff i* (*coordination.deviation* (λ-. *Choice-B*) *i x*)
≤ *coordination-payoff i* (λ-. *Choice-B*)
by (*cases i*; *cases x*) (*simp-all add*: *coordination-payoff-def coordination.deviation-def*)
qed

8 Two-Player Profile Sums

definition *two-player-profile* :: *'a* ⇒ *'a* ⇒ *penny-player* ⇒ *'a* **where**
two-player-profile r c p = (*case p of Row* ⇒ *r* | *Column* ⇒ *c*)

lemma *two-player-profile-simps* [*simp*]:
two-player-profile r c Row = *r*
two-player-profile r c Column = *c*
by (*simp-all add*: *two-player-profile-def*)

lemma *sum-profiles-fixed-Row*:
fixes *F* :: (*penny-player* ⇒ *'a*::*finite*) ⇒ *'b*::*comm-monoid-add*
shows (∑ *s*∈{*s*. *s Row* = *x*}. *F s*) =
(∑ *y*∈*UNIV*. *F* (*two-player-profile x y*))
proof (*rule sum.reindex-bij-witness*[
where *i* = λ*y*. *two-player-profile x y* **and** *j* = λ*s*. *s Column*])
fix *s* :: *penny-player* ⇒ *'a*

```
  assume s: s ∈ {s. s Row = x}
  show profile-eq: two-player-profile x (s Column) = s
  proof
    fix p
    show two-player-profile x (s Column) p = s p
      using s by (cases p) auto
  qed
  show F (two-player-profile x (s Column)) = F s
    by (simp add: profile-eq)
qed auto

lemma sum-profiles-fixed-Column:
  fixes F :: (penny-player ⇒ 'a::finite) ⇒ 'b::comm-monoid-add
  shows (∑ s∈{s. s Column = x}. F s) =
    (∑ y∈UNIV. F (two-player-profile y x))
proof (rule sum.reindex-bij-witness[
    where i = λy. two-player-profile y x and j = λs. s Row])
  fix s :: penny-player ⇒ 'a
  assume s: s ∈ {s. s Column = x}
  show profile-eq: two-player-profile (s Row) x = s
  proof
    fix p
    show two-player-profile (s Row) x p = s p
      using s by (cases p) auto
  qed
  show F (two-player-profile (s Row) x) = F s
    by (simp add: profile-eq)
qed auto
```

9 Matching Pennies as a Mixed Equilibrium

```
definition matching-pennies-payoff-real ::
    penny-player ⇒ (penny-player ⇒ coin-side) ⇒ real where
  matching-pennies-payoff-real p s = real-of-int (matching-pennies-payoff p s)

interpretation matching-pennies-mixed:
  finite-type-game matching-pennies-payoff-real .

lemma matching-pennies-pure-deviation-Row:
  matching-pennies-mixed.pure-deviation-payoff Row x m =
    (∑ y∈UNIV.
      matching-pennies-mixed.prob m Column y *
      matching-pennies-payoff-real Row (two-player-profile x y))
  by (simp add: matching-pennies-mixed.pure-deviation-payoff-def
      matching-pennies-mixed.opponent-weight-def sum-profiles-fixed-Row)

lemma matching-pennies-pure-deviation-Column:
  matching-pennies-mixed.pure-deviation-payoff Column x m =
    (∑ y∈UNIV.
```

matching-pennies-mixed.prob m Row y ∗
matching-pennies-payoff-real Column (*two-player-profile y x*))
by (*simp add*: *matching-pennies-mixed.pure-deviation-payoff-def*
matching-pennies-mixed.opponent-weight-def sum-profiles-fixed-Column)

lemma *matching-pennies-uniform-pure-deviation-payoff*:
matching-pennies-mixed.pure-deviation-payoff i x
matching-pennies-mixed.uniform-mixed-profile = *1* / *2*
by (*cases i*; *cases x*)
(*simp-all add*: *matching-pennies-pure-deviation-Row*
matching-pennies-pure-deviation-Column matching-pennies-payoff-real-def
matching-pennies-payoff-def)

lemma *matching-pennies-uniform-mixed-payoff*:
matching-pennies-mixed.mixed-payoff i matching-pennies-mixed.uniform-mixed-profile
= *1* / *2*
by (*simp add*: *matching-pennies-mixed.mixed-payoff-def*
matching-pennies-uniform-pure-deviation-payoff)

lemma *matching-pennies-uniform-mixed-Nash*:
matching-pennies-mixed.mixed-Nash-equilibrium
matching-pennies-mixed.uniform-mixed-profile
by (*auto simp*: *matching-pennies-mixed.mixed-Nash-equilibrium-def*
matching-pennies-uniform-pure-deviation-payoff
matching-pennies-uniform-mixed-payoff)

10 Rock-Paper-Scissors

datatype *rps* = *Rock* | *Paper* | *Scissors*

instantiation *rps* :: *finite*
begin

instance
proof
show *finite* (*UNIV* :: *rps set*)
by (*rule finite-subset*[*of* - {*Rock*, *Paper*, *Scissors*}]) (*auto intro*: *rps.exhaust*)
qed

end

lemma *UNIV-rps* [*simp*]: (*UNIV* :: *rps set*) = {*Rock*, *Paper*, *Scissors*}
by (*auto intro*: *rps.exhaust*)

fun *beats* :: *rps* ⇒ *rps* ⇒ *bool* **where**
beats Rock Scissors = *True*
| *beats Paper Rock* = *True*
| *beats Scissors Paper* = *True*
| *beats* - - = *False*

definition *rps-payoff* :: *penny-player* ⇒ (*penny-player* ⇒ *rps*) ⇒ *real* **where**
rps-payoff p s =
(*if s Row* = *s Column then 0*
else if beats (*s Row*) (*s Column*)
then if p = *Row then 1 else* −*1*
else if p = *Row then* −*1 else 1*)

interpretation *rps-mixed*:
finite-type-game rps-payoff **.**

lemma *rps-pure-deviation-Row*:
rps-mixed.pure-deviation-payoff Row x m =
(∑ *y*∈*UNIV*. *rps-mixed.prob m Column y* ∗ *rps-payoff Row* (*two-player-profile x y*))
by (*simp add*: *rps-mixed.pure-deviation-payoff-def rps-mixed.opponent-weight-def sum-profiles-fixed-Row*)

lemma *rps-pure-deviation-Column*:
rps-mixed.pure-deviation-payoff Column x m =
(∑ *y*∈*UNIV*. *rps-mixed.prob m Row y* ∗ *rps-payoff Column* (*two-player-profile y x*))
by (*simp add*: *rps-mixed.pure-deviation-payoff-def rps-mixed.opponent-weight-def sum-profiles-fixed-Column*)

lemma *rps-uniform-pure-deviation-payoff*:
rps-mixed.pure-deviation-payoff i x rps-mixed.uniform-mixed-profile = *0*
by (*cases i*; *cases x*)
(*simp-all add*: *rps-pure-deviation-Row rps-pure-deviation-Column rps-payoff-def*)

lemma *rps-uniform-mixed-payoff*:
rps-mixed.mixed-payoff i rps-mixed.uniform-mixed-profile = *0*
by (*simp add*: *rps-mixed.mixed-payoff-def rps-uniform-pure-deviation-payoff*)

lemma *rps-uniform-mixed-Nash*:
rps-mixed.mixed-Nash-equilibrium rps-mixed.uniform-mixed-profile
by (*auto simp*: *rps-mixed.mixed-Nash-equilibrium-def*
rps-uniform-pure-deviation-payoff rps-uniform-mixed-payoff)

end

References

[1] A. Bagnall, S. Merten, and G. Stewart. A library for algorithmic game theory in ssreflect/coq. *Journal of Formalized Reasoning*, 10(1):67–95, 2017. DOI: `https://doi.org/10.6092/issn.1972-5787/7235`.

[2] S. Le Roux, E. Martin-Dorel, and J.-G. Smaus. An existence theorem

of nash equilibrium in coq and isabelle. In *Proceedings Eighth International Symposium on Games, Automata, Logics and Formal Verification*, volume 256 of *Electronic Proceedings in Theoretical Computer Science*, pages 46–60, 2017. DOI: `https://doi.org/10.4204/EPTCS.256.4`.

[3] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14(1):124–143, 1996. DOI: `https://doi.org/10.1006/game.1996.0044`.

[4] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951. DOI: `https://doi.org/10.2307/1969529`.

[5] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. DOI: `https://doi.org/10.1073/pnas.36.1.48`.