

A Constructive Approach to Sequential Nash Equilibria

René Vestergaard

School of Information Science, JAIST, Nomi, Ishikawa 923-1292, Japan

Abstract

We present a Coq-formalised proof that all non-cooperative, sequential games have a Nash equilibrium point. Our proof methodology follows the style advocated by LCF-style theorem provers, i.e., it is based on inductive definitions and is computational in nature. The proof i) uses simple computational means, only, ii) basically is by construction, and iii) reaches a constructively stronger conclusion than informal efforts. We believe the development is a first as far as formalised game theory goes.

Key words: Nash equilibria, automatic theorem proving, constructivity

1 Introduction

Game theory was given its first comprehensive exposition by von Neumann and Morgenstern [11] but the first equilibrium-existence result, pertaining to chess,¹ dates back to Zermelo [15]. Game theory has been applied to other kinds of actual games: poker, bridge, etc., and has been used as a model for, and to aid with, business management, legal and military analysis, stock invest-

ment, and more [2,7]. The best known application area is economics where the 1994 Nobel Prize was awarded to the inventor and the early proponents in economics of *Nash equilibria* [12]. Nash addressed the following question (through fixed-points on suitable convex sets) [8–10]:

Can all agents in a game choose such that no agent single-handedly can secure a better outcome, i.e., can it be ruled out that two or more agents can get in each other's way?

Email address:

`vester@jaist.ac.jp` (René Vestergaard).

URL:

`http://www.jaist.ac.jp/~vester/`
(René Vestergaard).

¹ “Either white or black can guarantee a win or both can guarantee a draw.”

1.1 Our Contribution

We establish formally the existence of (pure) Nash equilibria for arguably the simplest structure studied

in game theory, namely *sequential games* (in their purest form [4]).² The result is due to Kuhn [5], generalising [15] and refining [8–10]. Our proof uses *Backward-Induction equilibrium points*, which appear to have been introduced as a stand-alone concept (then called sub-game perfectness) by Selten [13,14]. That said, all proofs we are aware of, including ours, use the same basic idea of having agents choose locally optimally from the end of a game and ‘backwards’ to the beginning in order to guarantee global ‘optimality’. Our technical contributions are i) to base the proof on an inductive definition of game trees and ii) to use computation over the trees to construct an existential witness of the sought-after result. This enables us iii) to formally verify our proof in the Coq theorem prover [3] and to point out that iv) our formal proof basically is by construction.

More informally speaking, we also establish initial evidence that formal reasoning and game theory are good matches.

The Coq (v8.0) code underlying the paper is available from our home page and has been used to generate the LaTeX source of the formal statements in the article, except for the results headed “Proposition”, which exist internally (and by construction) in Coq. The proofs in Coq and in the article have the same struc-

² Work is under way to address a range of the known refinements as well as alternative forms, in particular *simultaneous games*.

ture and size, and share most of their vernacular.

1.2 This Article

Section 2 introduces the relevant parts of game theory and the very limited proof-machinery we employ (in Coq). Section 3 formalises the concepts needed for expressing the sought-after theorem and proves some immediate consequences. Section 4 sums these up as our stronger version of the considered result and shows how to obtain the standard version from there. Appendix A gives more information on Coq.

2 Games and Strategies

A sequential game (in extensive form) is a tree with pay-off functions in the leaves, dictating the win or loss of each player when the game finishes there. Internal nodes belong to the player whose available options the node formalises at that particular juncture. A *play* of a game, in this framework, is therefore a path from the root to a leaf.

2.1 Games, Inductively

Binary games, i.e., games with exactly two options in internal nodes, can be formalised in Coq as follows.³

³ We note, without proof, that the theorems in the article are equivalent when stated for binary games and for games

Definition $A := \text{nat}$.

Definition $P := \text{nat}$.

Definition $Ps := A \rightarrow P$.

Inductive $G : \text{Set} :=$

— $gLeaf : Ps \rightarrow G$

— $gNode : A \rightarrow G \rightarrow G \rightarrow G$.

The first three lines in the definition say that we think of the agents, A , and their payoffs, P , as natural numbers (to get easy access to the desired order and equality relations) and that payoff functions, Ps , associate the agents with (their) payoffs. The last part of the definition serves to define a **Set** of games, G , by **Inductively** stating when an element is to be considered a member of it: anything that can be constructed by repeatedly applying the two *constructor*-rules is a game and, vice versa, only an object that is thus constructed is a game. The $gLeaf$ -constructor takes one argument, a payoff function, from which it constructs a game. The $gNode$ -constructor for internal nodes is similar but note that here the substructures include two copies of game itself, in addition to specifying the agent owner.

2.2 Inductive Proof Theory

Inductive definitions amount to a least fixed-point construction and no conflict arises out of game being defined in terms of game, as long as all recursive occurrences in the constructor types are non-negative [1]. Informally speaking, this means that

with one-or-more options in internal nodes but that, coding-wise, the binary form is more readily accessible to humans.

no constructor may take an argument that, in the simplest case, takes the constructed domain as an argument. An algebraic structure defined in the style of game comes equipped (by construction) with a proof principle called *structural induction*. The most famous example is the natural numbers and weak number induction.

$$\text{Nat} ::= 0 \mid \text{succ}(\text{Nat})$$

$$\frac{P(0) \quad \forall n. P(n) \Rightarrow P(\text{succ}(n))}{\forall n \in \text{Nat}. P(n)}$$

In the case of game, structural induction allows us to prove that a property, P , holds for all games if it holds for all $gLeaf$ s and if, under the assumption that it holds for arbitrary g_1, g_2 , we can prove that it holds for derivable $gNode$ s, cf. Figure 1. The paragraph following the **Inductive** definition of games provides a detailed, albeit informal, justification for structural induction. It can be summarised to say that it suffices for the predicate in question, P , to preserve the constructor rules.

2.3 Strategies

As noted, a play is a path from the root $gNode$ of a game tree to a $gLeaf$. As is customary in game theory, we shall not use the notion of play formally but will, instead, take as primitive a structure in between games and plays, so-called *strategies*.

Inductive $C : \text{Set} :=$ — lf — rg .

Inductive $S : \text{Set} :=$

— $sLeaf : Ps \rightarrow S$

— $sNode : A \rightarrow C \rightarrow S \rightarrow S \rightarrow S$.

$$\frac{\forall po . P(gLeaf\ po) \quad \forall a, g_1, g_2 . P(g_1) \wedge P(g_2) \Rightarrow P(gNode\ a\ g_1\ g_2)}{\forall g \in game . P(g)}$$

Fig. 1. Structural-induction proof principle for game.

A strategy is essentially a game in which a choice between the available options is made in each node: left or right. Formally, and more generally speaking, however, game and strategy are two entirely separate entities. If required, as it will be in our case, we must explicitly prove that a relationship exists between them.

2.4 Recursion

Inductive definitions give us access to a very simple form of computation called structural recursion.⁴ In terms of the earlier Nat example, structural recursion says that an f defined with a concrete value for 0 and some value derived from $f(n)$ in the case of $f(\text{succ}(n))$ is a total, computable function. Using this, we can note that a strategy, unlike a game, induces a (non-trivial) play dictated by the indicated left/right choices. The resulting payoff function can be found as follows.

```

Fixpoint S_Ps (s:S) {struct s} : Ps :=
  match s with
  — (sLeaf po) ⇒ po
  — (sNode a c sl sr) ⇒
      match c with
      — lf ⇒ (S_Ps sl)
      — rg ⇒ (S_Ps sr)
      end
  end.

```

⁴ This is also called *primitive recursion* or, in programming-language terminology, *recursive descent*.

In the above, “{*struct s*}” and “*match s with ...*” is Coq-syntax indicating the use of structural recursion and we can conclude that S_Ps i) is functional, i.e., is a relation that is not one-to-many, because the case-splitting is non-overlapping, ii) is computable because all recursive calls are made on sub-structures of the considered case (over the well-founded S), and iii) is total (on S) because the case-splitting also is exhaustive.

Proposition 1 S_Ps is a total, computable function.

Proof By construction. \square

We saw that a strategy is a game with choices made in the nodes. Indeed, for a given strategy and using the structural-recursion principle just considered, we can access the underlying game simply by removing the listed choices.

```

Fixpoint s2g (s:S) {struct s} : G :=
  match s with
  — (sLeaf po) ⇒ (gLeaf po)
  — (sNode a c sl sr)
    ⇒ (gNode a (s2g sl) (s2g sr))
  end.

```

Proposition 2 $s2g$ is a total, computable function.

Proof By construction, as $s2g$ is defined by structural recursion. \square

```

Fixpoint agentConv (a:A) (s1 s2:S) {struct s1} : Prop :=
match s1 with
— (sLeaf po1) ⇒ match s2 with
— (sLeaf po2) ⇒ ∀ a, po1 a = po2 a
— (sNode _ _ _ ) ⇒ False
end
— (sNode a1 c1 s11 s12) ⇒ match s2 with
— (sLeaf _) ⇒ False
— (sNode a2 c2 s21 s22)
⇒ a1 = a2 ∧ (a1 = a ∨ c1 = c2)
∧ agentConv a s11 s21
∧ agentConv a s12 s22
end
end.

```

Fig. 2. agent-convertible strategy-equivalence.

3 Equilibria

A Nash equilibrium is a strategy in which no agent can change one or more of his choices to obtain a better overall result for himself. The options available to an agent are accounted for by the relation on strategies that share their underlying game and only differ in choices in nodes owned by the agent.

Proposition 3 *agentConv, cf. Figure 2, is a total, computable predicate.*⁵

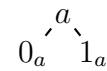
Proof By construction, as *agentConv* is defined by structural recursion. \square

The *agentConv*-predicate reduces the question of choice-equivalence to (extensional) equality for the payoff functions: $\forall a, po1 a = po2 a$, provided two strategies share their “shape”. A Nash equilibrium is a strategy that majorises any *agent-*

Conv-equivalent strategy in terms of the payoffs that the agents can expect.

Definition *Eq* ($s:S$) : Prop :=
 $\forall s', \forall a, agentConv a s s'$
 $\rightarrow (S_Ps s') a \leq (S_Ps s) a$.

The idea behind the definition is that in an equilibrium, no agent will have an (obvious) incentive to change his mind. In the game below, for example, we have one internal node, presenting a choice between left and right to its owner, a . If a chooses left, the game finishes and he receives a pay-off of 0. Similarly if he chooses right, with a pay-off of 1. The strategy where a chooses to go right is the only equilibrium point.



3.1 Backward Induction

We now present a formal strategy-notion, to be called Backward Induction, in which all choices are required to result in a local equilibrium, even if

⁵ In this context, ‘predicate’ is synonymous with ‘function into proposition, Prop’.

that particular choice is not encountered in the overall induced play.

```

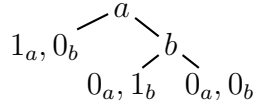
Fixpoint BI (s:S) {struct s} : Prop :=
  match s with
  — (sLeaf po) ⇒ True
  — (sNode a c sl sr) ⇒
    (BI sl) ∧ (BI sr) ∧
    match c with
    — lf ⇒ (S_Ps sr) a ≤ (S_Ps sl) a
    — rg ⇒ (S_Ps sl) a ≤ (S_Ps sr) a
    end
  end.

```

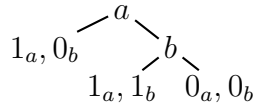
Proposition 4 *BI is a total, computable predicate.*

Proof By construction, as *BI* is defined by structural recursion. \square

The differences between Nash and Backward-Induction equilibria are seen in the following game.



The game has two equilibria: *a* chooses to go left and *b* can choose either left or right, as the latter choice, which must be made to make up a strategy, does not affect the overall outcome: 1 and 0, respectively. Only one of these is a Backward-Induction equilibrium, however, namely the one where *b* also chooses to go left. In our final example, indicating the non-triviality of the concepts, there are two Backward-Induction equilibria, one of which is better for *b* (*a* right, *b* left) than the other (*a* left, *b* left).



3.2 BI is Nash Equilibrium

As suggested, a Backward-Induction equilibrium is also a Nash equilibrium.

Lemma *BI_is_Eq* : $\forall s, BI\ s \rightarrow Eq\ s$.

The proof, which is by structural induction in *s* and a case-splitting following Figure 2, is lengthier than any of the other proofs we consider. It is available in the accompanying Coq theory but we shall not discuss the details here, in part because the result is unrelated to the existence question we are concerned with, i.e., it is not needed for proving **Theorem** *BI_fctEx* in Section 4. Informally speaking, the result holds because *BI* is the locally-enforced version of *Eq*'s global notion of optimality.

3.3 BI Computation

Our reason for using Backward-Induction, in addition to Nash equilibria more generally, is that its definition uses structural recursion, cf. Proposition 4, in a way that can be simulated by (the weaker) game-structural recursion. This means that we can reformulate the *BI* predicate as a function from game to strategy, rather than from strategy to *Prop*, that i) remains total and computable by construction and ii) still guarantees *BI*-ness of the outcome.

```

Fixpoint cBI (g:G) {struct g} : S :=
  match g with
  — (gLeaf po) ⇒ (sLeaf po)
  — (gNode a gl gr) ⇒
    let sl := cBI gl in
    let sr := cBI gr in
    if (le_ge_dec ((S_Ps sl) a))

```

((S_Ps sr) a))
 then ($sNode$ a rg sl sr)
 else ($sNode$ a lf sl sr)
end.

Proposition 5 *cBI is a total, computable function.*

Proof By construction, as *cBI* is defined by structural recursion. \square

We can, of course, verify also ii) above.

Lemma *cBI_BI* : $\forall g, BI (cBI g)$.

Proof The proof is by structural induction in g .

gLeaf case: Trivial, after unfolding the definitions of *cBI* and *BI*.

gNode case: We proceed by a case-splitting on the conditional obtained by unfolding the definition of *cBI*. With *BI* unfolded, as well, the two cases follow by the induction hypotheses and the considered conditional value. \square

We reemphasise a point made in Section 1.1: the above informal proof is basically a complete Coq proof of the result, differing only (slightly) in vernacular. The structure and size of the two proofs are directly comparable. Going back to our initial discussion about strategy and how the structure is intended to refine game, we see that we have the expected result.

Lemma *s2g_cBI* : $\forall g, g = s2g (cBI g)$.

Proof The proof is by structural induction in g .

gLeaf case: Trivial, after unfolding the definitions of *cBI* and *s2g*.

gNode case: We proceed by a case-splitting on the conditional ob-

tained by unfolding the definition of *cBI*. With *s2g* unfolded, as well, the two cases follow by observing that the two *gNodes* being considered have equal arguments, which, in turn, follow trivially or by the induction hypotheses. \square

4 Compendium

Summarising the results in Section 3.3, we have established the following.

Theorem *BI_fctEx* :

$$\exists F, \forall g, BI (F g) \wedge g = s2g (F g).$$

Proof *cBI* is a witness for F , cf. Lemmas *cBI_BI* and *s2g_cBI*. \square

This is the strongest result we prove. It says that there exists a function that sends any game to a Backward-Induction equilibrium for it. The result basically follows for free from the fact that we can define *cBI* by structural recursion over the inductively defined games (and from the two simple induction proofs for Lemmas *cBI_BI* and *s2g_cBI*). The result does not look exactly like the informal version and we discuss the two steps required to weaken our version. First, we relax Backward-Induction equilibrium to Nash equilibrium.

Theorem *Eq_fctEx* :

$$\exists F, \forall g, Eq (F g) \wedge g = s2g (F g).$$

Proof *cBI* is a witness for F , cf. Lemma *BI_is_Eq* and Theorem *BI_fctEx*. \square

Secondly and finally, we *deskolemise*, which means that we no longer require that the sought-after equilib-

rium is computed from the considered game.

Theorem *Eq_Ex* :

$$\forall g, \exists s, Eq\ s \wedge g = s2g\ s.$$

Proof As we saw, (*cBI g*) is a witness for *s*. \square

Skolemisation, the converse of deskolemisation, is not constructively (intuitionistically) valid, although it preserves classical validity (intuitionistic logic plus, e.g., excluded middle). The transformation we give, using deskolemisation, is constructively acceptable and, thus, **Theorems *BI_fctEx*, *Eq_fctEx*, and *Eq_Ex*** mark a gradual weakening. For the first transition, this is true both constructively and classically, while in the last step, we lose constructive information in the statement of the result (that, however, can be recovered from the proof).

5 Conclusion

We have presented a fully-formalised proof that a Nash equilibrium exists for all non-cooperative, sequential games. Our proof is constructive and shows i) that the equilibrium can be chosen uniformly in the considered game (starting from within), ii) that the choice actually is computable, and iii) that the computed equilibrium point is of the more restricted kind known as Backward Induction. The basic properties we identified and took advantage of are a) that games are formal structures in their own right and b) that the Backward-Induction predicate can be made into a total function on

games that computes a corresponding Backward-Induction equilibrium, using structural recursion.

Acknowledgements

We thank Olivier Danvy, Mamoru Kaneko, Stéphane Le Roux, Pierre Lescanne, the editor, and the referees for their comments.

A “What is Coq?” [3]

“Developed in the LogiCal project, the Coq tool is a formal proof management system: a proof done with Coq is mechanically checked by the machine. In particular, Coq allows:

- the definition of functions or predicates,
- to state mathematical theorems and software specifications,
- to develop interactively formal proofs of these theorems,
- to check these proofs by a small certification “kernel”.

Coq is based on a logical framework called “Calculus of Inductive Constructions” extended by a modular development system for theories. Coq also includes

- a mechanism for automatic generation of certified programs from proofs of their specifications,
- a graphical user interface based on gtk (CoqIde),
- a documentation tool (coqdoc),
- dependency and makefile generation tools for Coq (coq_makefile and coqdep),
- a preprocessor for TeX files that include Coq commands (coqtex).

Coq is written in the Objective Caml language and uses the Camlp4 Pre-processor-pretty-printer for Objective Caml. Coq is distributed under the GNU Lesser General Public Licence Version 2.1 (LGPL). The current stable version of Coq is the 8.0. It is currently available for Unix (including Mac OS X) and Windows 95/98/NT/XP systems.”

[3]

References

- [1] Peter Aczel. An introduction to inductive definitions. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.7, pages 739–782. North-Holland, Amsterdam, 1977.
- [2] Robert J. Aumann and Sergiu Hart, editors. *Handbook of Game Theory with Economic Applications, volumes 1, 2, 3*. North-Holland, 1992, 1994, 2002.
- [3] Gilles Dowek, Christine Paulin-Mohring, et al. Coq. <http://coq.inria.fr/>.
- [4] Sergiu Hart. *Games in Extensive and Strategic Form*, chapter 2. Volume 1 of Aumann and Hart [2], 1992.
- [5] Harold W. Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games II*, 1953. Reprinted in [6].
- [6] Harold W. Kuhn, editor. *Classics in Game Theory*. Princeton University Press, 1997.
- [7] Elliott Mendelson. *Introducing game theory and its applications*. CRC, 2004.
- [8] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36, 1950. Reprinted in [6].
- [9] John F. Nash. *Non-Cooperative Games*. PhD thesis, Princeton University, 1950.
- [10] John F. Nash. Non-cooperative games. *Annals of Mathematics*, 54, 1951. Reprinted in [6].
- [11] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.
- [12] The Royal Swedish Academy of Sciences. Citation for the 1994 Nobel prize in economics to John C. Harsanyi, John F. Nash, and Reinhard Selten. <http://nobelprize.org/economics/laureates/1994/presentation-speech.html>.
- [13] Reinhard Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfragerträgeit. *Zeitschrift für die gesamte Staatswissenschaft*, 121, 1965.
- [14] Reinhard Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4, 1975. Reprinted in [6].
- [15] Ernst Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In *Proceedings of the Fifth International Congress of Mathematicians*, volume II, 1912.