HAL
archives-ouvertes.fr

Acyclicity of Preferences, Nash Equilibria, and Subgame Perfect Equilibria: a Formal and Constructive Equivalence

Stéphane Le Roux

▶ **To cite this version:**

Stéphane Le Roux. Acyclicity of Preferences, Nash Equilibria, and Subgame Perfect Equilibria: a Formal and Constructive Equivalence. [Research Report] 2007, pp.41. <inria-00148103>

HAL Id: inria-00148103
https://hal.inria.fr/inria-00148103

Submitted on 22 May 2007

Acyclicity of Preferences, Nash Equilibria, and Subgame Perfect Equilibria: a Formal and Constructive Equivalence

Stéphane Le Roux[* * *]

École normale supérieure de Lyon, LIP, CNRS, INRIA, UCBL

Abstract. Sequential game and Nash equilibrium are basic key concepts in game theory. In 1953, Kuhn showed that every sequential game has a Nash equilibrium. The two main steps of the proof are as follows: First, a procedure expecting a sequential game as an input is defined as "backward induction" in game theory. Second, it is proved that the procedure yields a Nash equilibrium. "Backward induction" actually yields Nash equilibria that define a proper subclass of Nash equilibria. In 1965, Selten named this proper subclass subgame perfect equilibria. In game theory, payoffs are rewards usually granted at the end of a game. Although traditional game theory mainly focuses on real-valued payoffs that are implicitly ordered by the usual total order over the reals, there is a demand for results dealing with non totally ordered payoffs. In the mid 1950's, works of Simon or Blackwell already involved partially ordered payoffs. This paper further explores the matter: it generalises the notion of sequential game by replacing real-valued payoff functions with abstract atomic objects, called outcomes, and by replacing the usual total order over the reals with arbitrary binary relations over outcomes, called preferences. This introduces a general abstract formalism where Nash equilibrium, subgame perfect equilibrium, and "backward induction" can still be defined. Using a lemma on topological sorting, this paper proves that the following three propositions are equivalent: 1) Preferences over the outcomes are acyclic. 2) Every sequential game has a Nash equilibrium. 3) Every sequential game has a subgame perfect equilibrium. The result is fully computer-certified using the (highly reliable) constructive proof assistant called Coq. Beside the additional guarantee of correctness provided by the proof in Coq, the activity of formalisation also helps clearly identify the useful definitions and the main articulations of the proof.

Abstract sequential game, Nash equilibrium, subgame perfect equilibrium, constructivism, induction, proof assistant

---

1 Introduction

1.1 Game Theory in Short

Game theory embraces the theoretical study of processes involving (more or less conscious) possibly interdependent decision makers. Game theory originates in economics, politics, law, and also games dedicated to entertainment. Instances of game theoretic issues may be traced back to Babylonian times when the Talmud would prescribe marriage contracts that seem to be solutions of some relevant games described in [7]. In 1713, a simple card game raised questions and solutions involving probabilities, as discussed in [2]. During the XVIIth and XVIIIth centuries, philosophers such as Hobbes [6] adopted an early game theoretical approach to study political systems. In 1838, Cournot [5] introduced the notion of equilibrium for pricing in duopoly. It is said that game theory became a major field in 1944, when von Neumann and Morgenstern [15] published a summary of prior works and a systematic study of a few classes of games. In 1950, the notion of equilibrium and the corresponding solution concept discussed by Cournot were generalised by Nash [14] for a class of games called *strategic games*. In 2006, the notion of Nash equilibrium was abstracted [12] as a situation that no stakeholder can convert into another situation that he prefers. In addition to economics, politics and law, modern game theory is consciously involved in many other fields such as biology, computer science, and sociology.

1.2 Sequential Game Theory

Another class of games is that of *sequential games*, also called *games in extensive form*. It traditionally refers to games where players play in turn till the play ends and *payoffs* are granted. For instance, the game of chess is naturally modelled by a sequential game where payoffs are "win", "lose" and "draw". Sequential games are often represented by finite rooted trees each of whose internal nodes is owned by a player, and each of whose external nodes encloses one payoff per player. In 1912, Zermelo [20] proved about the game of chess that either white can win, or black can win, or both sides can force a draw. This is sometimes considered as the first non-trivial theoretical results in game theory. Although the concept of Nash equilibrium referred to strategic games in the first place, it is natural and relevant to extend that concept to sequential games. In 1953, Kuhn [9] showed the existence of Nash equilibrium for sequential games. For this, he built a specific Nash equilibrium through what is called "backward induction" in game theory. In 1965, Selten ([16] and [17]) introduced the concept of subgame perfect equilibrium in sequential games. This is a refinement of Nash equilibrium that seems to be even more meaningful than Nash equilibrium for sequential games. The two concepts of "backward induction" and subgame perfect equilibrium happen to coincide, so Kuhn's result also guarantees existence of subgame perfect equilibrium in sequential games. In 2006, Vestergaard [19] formalised part of Kuhn's result with the proof assistant Coq, for the subclass of games represented by binary trees and whose payoffs range over the natural numbers. For this, he defined sequential games and corresponding strategy profiles inductively.

1.3 Ordering Payoffs

Game theory has mostly studied games with real-valued payoffs, perhaps for the following reason: In 1944, von Neumann and Morgernstern [15] suggested that the notion of payoff in economics could be reduced to real numbers. They argued that more and more physical phenomena were measurable; therefore, one could reasonably expect that payoffs in economics, although not yet measurable, would become reducible to real numbers some day. However, game theory became popular soon thereafter, and its scope grew larger. As a result, several scientists and philosophers questioned the reducibility of payoffs to real numbers. In 1955, Simon [18] discussed games where agents are awarded (only partially ordered) vectors of real-valued payoffs instead of single real-valued payoffs. In 1956, Blackwell [4] proved a result involving vectors of payoffs. Those vectors model agents that take several non-commensurable dimensions into consideration; such games are sometimes called multi-criteria games. In 1994, Osborne and Rubinstein [13] mentioned arbitrary preferences, without any further results. In 2003, Krieger [8] noticed that "backward induction" on sequential multi-criteria games may not yield Nash equilibria, and yet showed that sequential multi-criteria games have Nash equilibria. The proof seems to invoke probabilities and Nash's theorem for strategic games. In 2006, a paper [10] discussed sequential games with arbitrary partially ordered payoffs and proved in Coq the existence of non-deterministic Nash equilibria *via* "backward induction".

1.4 Contribution

This paper contributes at both the technical and the presentation level.

There are five main technical contributions: First, an inductive formalism is designed to represent sequential games in the constructive proof assistant Coq ([1] and [3]), and all the results in this paper are proved in Coq. Second, the new formalism allows the paper to introduce an abstraction of traditional sequential games and of a few related concepts. The abstraction preserves the tree structure of the game but replaces the real-valued payoff functions, enclosed in the leaves of the trees, by arbitrary outcomes. Each agent has a preference over outcomes, which is given *via* an explicit and arbitrary binary relation. This preference replaces the implicit and traditional "usual total order" over the real numbers. Nash equilibria and subgame perfect equilibria are defined accordingly. Third, Kuhn's result [9] is translated into the new formalism when agents' preferences are totally ordered. Fourth, the notion of "backward induction" is naturally generalised for arbitrary preferences, but a simple example shows that total ordering of preferences is needed for "backward induction" to guarantee subgame perfect equilibrium: the two notions of "backward induction" and subgame perfect equilibrium coincide for total orders but not in general. Fifth, Kuhn's result is substantially generalised as follows. On the one hand, an intermediate result proves that smaller preferences, *i.e.*, binary relations with less arcs, yield more equilibria than bigger preferences. On the other hand, a topological sorting result was formally proved in [11]. By both results mentioned above, acyclicity of

the preferences proves to be a necessary and sufficient condition for every game to have a Nash equilibrium/subgame perfect equilibrium.

This paper deals with basic notions of game theory that are all exemplified and defined before they are used. Most of the time, these notions are explained in three different ways, with the second one helping make the connection between the two others: First, the notions are presented in a graphical formalism close to traditional game theory. Second, they are presented in a graphical formalism suitable for induction. Third, they are presented in a light Coq formalism close to traditional mathematics, so that only a basic understanding of Coq is needed. (A quick look at the first ten pages of [11] will introduce the reader to the required notions.) The proofs are structured along the corresponding Coq proofs but are written in plain English.

1.5 Contents

Section 2 gives an intuitive and informal presentation of traditional sequential games through graphical examples. Section 3 explores further the relevance of non-totally ordered payoffs. Section 4 discusses general concepts that are not specially related to game theory but required in the remainder of the paper. Section 5 presents the above-mentioned abstraction of sequential games and their new formalism. Section 6 presents the notion of strategy profile at the same level of abstraction as for sequential games. Section 7 defines convertibility between strategy profiles. Section 8 discusses the notion of preference, happiness, Nash equilibrium, and subgame perfect equilibrium. Section 9 generalises "backward induction", translates Kuhn's result into the new formalism, and proves the triple equivalence between acyclicity of preferences and existence of Nash equilibrium/subgame perfect equilibrium.
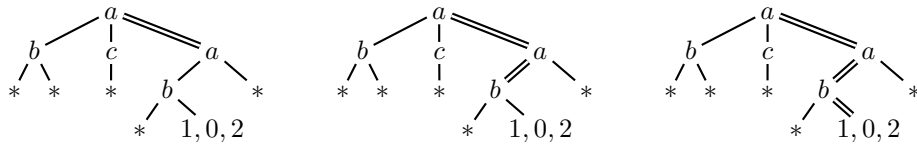
2 Traditional Sequential Game Theory

Through graphical examples and explanations in plain English, this section gives an intuitive and informal presentation of traditional sequential games and of a few related concepts such as Nash equilibrium.

A traditional sequential game involves finitely many agents and payoffs. Payoffs usually are real numbers. A payoff function is a function from the agents to the payoffs. A sequential game is a rooted finite tree with internal nodes labelled with agents and external nodes, *i.e.*, leaves, labelled with payoff functions. Consider the following graphical example of such games. It involves the two agents a and b. At every leaf, a payoff function is represented by two numbers separated by a comma: the payoff function maps agent a to the first number and agent b to the second number.

$$
\begin{array}{c}
\;\overset{a}{\diagup\;\diagdown} \\
\overset{b}{\diagup\;\diagdown}\quad 2,2 \\
1,0\quad 3,1
\end{array}
$$
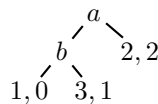
Such game trees are interpreted as follows: A play of a game starts at the root of the game tree. If the tree is a leaf then agents are rewarded according to the enclosed payoff function. Otherwise, the agent owning the root chooses the next node among the children of the root. The subtree rooted at the chosen child is considered and the play continues from there. In the game above, if a chooses to go right then both agents get 2. If a chooses left then b has to choose too, etc. Now, a specific play of a game is described and drawn. Double lines represent choices made during the play. Agent a first chooses to go right, then left, then b chooses to go right. Eventually, a gets 1, b gets 0, and c gets 2. The stars $*$ represent arbitrary payoff functions irrelevant to the discussion.

$$
\begin{array}{ccc}
\vcenter{\hbox{tree 1}} & \vcenter{\hbox{tree 2}} & \vcenter{\hbox{tree 3}}
\end{array}
$$

In game theory, the strategy of an agent is an object that accounts for the decisions of the agent in all situations that the agent might encounter. A strategy profile is a tuple combining one strategy per agent. So, for sequential games, a strategy profile amounts to choices made at all internal nodes of the tree. Below is an example of a strategy profile. Double lines between nodes represent choices and the stars $*$ represent arbitrary payoff functions irrelevant to the discussion. The choice of b at the leftmost internal node may seem rather ineffective, but it can be interpreted as b's choice if a play ever reach this very node.
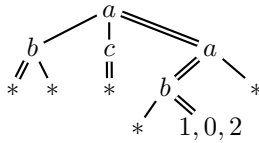
Given a strategy profile, starting from the root and following the agents' consecutive choices leads to one leaf. The payoff function enclosed in that specific leaf is called the induced payoff function. The induced payoff function of the strategy profile above is: a gets 1, b gets 0, and c gets 2.

The (usually implicit) preference of agents for strictly greater payoffs induces a (usually implicit) preference of an agent for payoff functions that grants him strictly greater payoffs. This, in turn, yields a (usually implicit) preference of an agent for strategy profiles inducing preferred payoff functions. Below, agent a prefers the left-hand strategy profile to the right-hand one since 3 is greater than 2, but it is the opposite for agent b since 1 is less than 2.

An agent is (usually implicitly) granted the ability to change his choices at all nodes he owns. For instance, below, the agent b can convert the strategy

profile on the left to the one on the right by changing his choices exactly at the nodes where **b** is displayed in bold font. The stars $*$ represent arbitrary payoff functions irrelevant to the discussion.

An agent is said to be happy with a strategy profile if he cannot convert it to another strategy profile that he prefers. A Nash equilibrium is a strategy profile that makes all agents happy. Below, the strategy profile to the left is not a Nash equilibrium since its sole player gets 0 but can convert it to the right-hand strategy profile and get 1. However, the right-hand strategy profile is a Nash equilibrium since a cannot convert it and get a payoff strictly greater than 1.

Here is another example of Nash equilibrium.
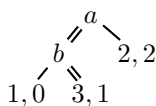
Indeed, agent a could only convert the strategy profile above to the left-hand strategy profile below, and would get 2 instead of 3. Therefore a is happy. Agent b could only convert the strategy profile above to the right-hand strategy profile below, and would get 0 instead of 1. Therefore b is happy too. The strategy profile above makes all players happy; it is a Nash equilibrium.

The underlying game of a strategy profile is computed by forgetting all the choices made at the internal nodes of the strategy profile. The next picture displays a strategy profile, to the left, and its underlying game, to the right.

Two Nash equilibria inducing different payoff functions may have the same underlying game: indeed, consider the previous Nash equilibrium and the following one, where b's choice is ineffective in terms of induced payoff function.
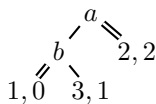
A given game is said to have a Nash equilibrium if there exists a Nash equilibrium whose underlying game is the given game. In order to prove that every sequential game has a Nash equilibrium, one can use a construction called "backward induction" in game theory. It consists in building a strategy profile from a sequential game. Performing "backward induction" on the following example will help describe and interpret the idea of the construction.
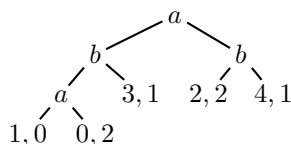
If a play starts at the leftmost and lowest node of the game above, then agent a faces the following game:
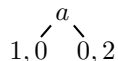
So, provided that a is "rational" in some informal sense, he chooses left and get 1 instead of 0. In the same way, if the play starts at the rightmost node, b chooses left and get 2 instead of 1. These two remarks correspond to the leftmost picture below. Provided that agent b is aware of a's "rational" behaviour, if a play starts at the left node owned by b, then b chooses right and get 1 instead of 0, as shown on the second picture below. The last picture shows that when a play starts at the root, as in the usual interpretation of a sequential game, a chooses left and gets 3 instead of 2. In such a process, an agent facing several options equivalent in terms of payoffs may choose either of them.
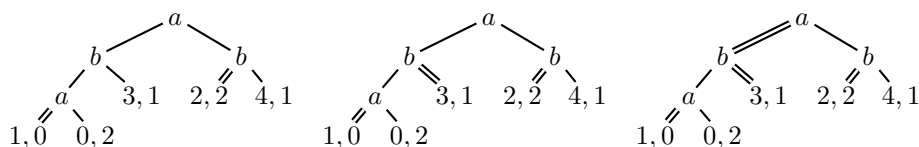

A strategy profile built by "backward induction" is a Nash equilibrium whose underlying game is the original game. (A formal proof relying on formal definitions is presented in a later section.) This way, it is proved that all sequential games have Nash equilibria. However, the next example shows that not all Nash equilibria are obtained by "backward induction". Even stronger, a Nash equilibrium may induce a payoff function induced by no "backward induction" strategy profile. Indeed, the left-hand strategy profile below is a Nash equilibrium that is not a "backward induction", and the only "backward induction" on the same underlying game is shown on the right-hand side.

Traditional game theory uses the following definition of subtree: all the descendants of a node of a tree define a subtree. For instance consider the following game.
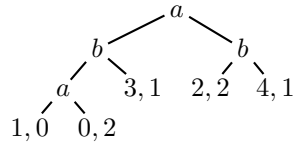
In addition to the leaves, the proper subtrees of the game above are listed below.
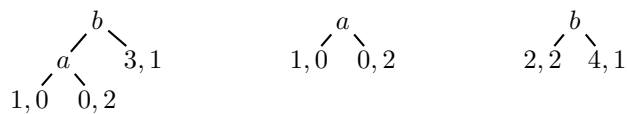
With this definition of subtree, a subgame perfect equilibrium is defined as a Nash equilibrium all of whose substrategy profiles are also Nash equilibria. In traditional game theory, the notions of subgame perfect equilibrium and "backward induction" strategy profile coincide.

3 Why Total Order?

Section 2 only deals with real-valued, totally ordered payoffs. However, as mentioned in subsection 1.3, there is a need for game theoretic results involving non totally ordered payoffs. This section adds simple and informal arguments to the discussion. In particular, it shows that the class of traditional sequential games naturally induces two classes of games slightly more general than itself. For these classes of games, the question whether Nash equilibria/subgame perfect equilibria exist or not is still relevant, and has yet not been addressed by Kuhn's result.

3.1 Selfishness Refinements

An agent that gives priority to his own payoffs without taking other agents into consideration is called selfish. It is the case in traditional game theory. Now consider a benevolent agent that takes all agents, including him, into account, $e.g.$, in a Pareto style. More specifically, consider two payoff functions p and p'. If for each agent, p grants a payoff greater than or equal to the one granted by p', and if there exists one agent to whom p grants strictly greater payoff than p', then the benevolent agent prefers p to p'. For instance, consider three agents a, b and c, and three payoff functions $(1, 3, 0)$, $(1, 2, 1)$, and $(1, 2, 0)$. (The first component corresponds to a, the second to b, and the third to c.) A benevolent

agent prefers the first two to the last one, but has no preference among the first two.

An agent is selfish-benevolent if his preference is the union of selfish and benevolent preferences. Put otherwise, an agent prefers a payoff function to another one if he prefers it either selfishly or benevolently. For instance, consider the previous example. Assume that all agents are selfish-benevolent. So, b prefers $(1,3,0)$ to $(1,2,1)$ since 3 is greater than 2, prefers $(1,2,1)$ to $(1,2,0)$ by benevolence, and prefers $(1,3,0)$ to $(1,2,0)$ by both selfishness and benevolence. Selfishness-benevolence induces a partial order over real-valued payoff functions.

In the same way, an agent may be selfish-malevolent. More specifically, consider two payoff functions p and p'. If p grants a selfish-malevolent agent a payoff greater than the one granted by p' then the agent prefers p to p'. If p and p' grant the same payoff to the selfish-malevolent agent, and if p grants every other agent a payoff lesser than or equal to the one granted by p', and if there exists one agent to whom p grants strictly lesser payoff than p', then the selfish-malevolent agent prefers p to p'. Selfishness-malevolence induces a partial order over real-valued payoff functions too.

3.2   Lack of Information

Consider an agent that prefers greater payoffs and that is "rational", *i.e.*, acts according to his preferences. Imagine the following 1-player game played by the above-mentioned agent: when a play starts, the agent has two options, say left and right. If he chooses left then he gets either 0 or 5, and if he chooses right then he gets either 1 or 2 or 3. After payoffs are granted, the play ends. This game is represented below.

$$\overset{a}{\diagup \diagdown} \\ \{0,5\} \quad \{1,2,3\}$$
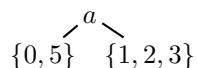
The wording "either... or...", in the phrase "either 0 or 5" does not refer to any procedure whatsoever. Therefore, in each case the agent has no clue how payoffs are going to be granted. It is worth stressing that, in particular, "either 0 or 5" does not refer to probability half for 0 and probability half for 5. It does not refer to probabilities at all. As a result, the agent cannot dismiss for sure any of his options in the game above. The two options are not comparable; the payoffs are not totally ordered. This type of game can even help understand traditional sequential games better. Indeed, consider the traditional sequential game below, where a and b are "rational", and therefore only care about maximising their own payoffs. Also assume that a knows b's being "rational".

$$\overset{a}{\diagup \diagdown} \\ \overset{b}{\diagup\diagdown} \qquad \overset{b}{\diagup|\diagdown} \\ 0,0 \quad 5,0 \quad 1,0 \ 2,0 \ 3,0$$
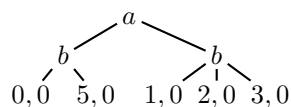
Whatever agent a may choose, b's options are equivalent since they yield the same payoff. Therefore a has no clue how b is going to choose: Go left when left and right are equivalent? Toss a coin? Phone a friend? So, from agent a's viewpoint, the game above reduces to the 1-player game discussed before. In this subsection, non totally ordered payoffs represents lack of information without invoking a formalism dedicated to knowledge representation such as epistemic logic.

When payoffs are non-empty sets of real numbers instead of single real numbers, as in the first example above, there are several ways to define relevant preferences. For instance, agents can focus either on the lower bound of the set, which amounts to guaranteeing a minimum, or on the upper bound of the set, which amounts to letting hope for a maximum, or both at once. Most of these yield partial orders over non-empty sets of reals.

4   Preliminaries

Prior to the game theoretic development presented in Coq in later sections, a few general concepts and related results are needed. Part of them are mentioned in [11]: list inclusion, subrelation, restriction of a relation to the elements of a list, etc. This section completes the inventory of the required notions, in the Coq formalism.

A first useful result reads as follows: given a binary relation and two lists, one included in the other, the restriction of the relation to the smaller list is a subrelation of the restriction of the relation to the bigger list. The proof is a straightforward unfolding of the definitions, and the result is formally written below.

Lemma $sub\_rel\_restriction\_incl$: $\forall$ $(B : Set)(l\ l' : list\ B)\ R,$
$incl\ l\ l' \rightarrow sub\_rel\ (restriction\ R\ l)\ (restriction\ R\ l').$

The remainder of this section presents the extension to lists of four concepts usually pertaining to one or two objects only.

4.1   Extension of Predicates to Lists

Let A be a *Set*. The function *listforall* expects a predicate on A, *i.e.*, an object of type $A \rightarrow Prop$, and returns a predicate on lists, *i.e.*, an object of type *list* $A \rightarrow$ *Prop*, stating that all the elements in the list comply with the original predicate. It is recursively defined along the inductive structure of the list argument.

Fixpoint $listforall$ $(Q : A \rightarrow Prop)(l : list\ A)\{struct\ l\} : Prop :=$
$match\ l\ with$
$|\ nil \Rightarrow True$
$|\ x::l' \Rightarrow Q\ x \wedge listforall\ Q\ l'$
$end.$

In the first line above, Fixpoint starts the recursive definition, *listforall* is the name of the defined function, Q is the predicate argument, l is the list argument,

{*struct l*} means that the recursion involves *l*, and *Prop* is the type of the output. The *match* performs a case splitting on the structure of *l*. The line thereafter specifies that the function returns *True* for empty list arguments. The last line is the core of the recursion: in order to compute the result for the list, it refers to the result of the computation involving the tail, which is a strictly smaller argument. This ensures termination of the computation, therefore the function is well defined. An example of computation of *listforall* is given below. The symbol $\rightsquigarrow$ represents a computation step.

listforall Q (x::y::z::nil) $\rightsquigarrow$ *Q x* $\land$ *listforall Q (y::z::nil)* $\rightsquigarrow$
Q x $\land$ *Q y* $\land$ *listforall Q (z::nil)* $\rightsquigarrow$ *Q x* $\land$ *Q y* $\land$ *Q z* $\land$ *listforall Q nil* $\rightsquigarrow$
Q x $\land$ *Q y* $\land$ *Q z* $\land$ *True*

Note that *Q x* $\land$ *Q y* $\land$ *Q z* $\land$ *True* is equivalent to *Q x* $\land$ *Q y* $\land$ *Q z*, which is what the function *listforall* is meant for.

The following four lemmas involve the notion of appending (++), also called concatenation, of two lists. It is defined in the Coq Standard Library. The four lemmas express basic properties of the *listforall* function. They are all proved by induction on the list *l*.

Lemma *listforall_app* : $\forall$ *Q l' l*,
listforall Q l $\rightarrow$ *listforall Q l'* $\rightarrow$ *listforall Q (l++l')*.

Lemma *listforall_appl* : $\forall$ *Q l' l*, *listforall Q (l++l')* $\rightarrow$ *listforall Q l*.

Lemma *listforall_appr* : $\forall$ *Q l' l*, *listforall Q (l++l')* $\rightarrow$ *listforall Q l'*.

Lemma *listforall_In* : $\forall$ *Q x l*, *In x l* $\rightarrow$ *listforall Q l* $\rightarrow$ *Q x*.

4.2 Extension of Functions to Lists

The Coq Standard Library provides a function *map* that, given a list and a function *f*, returns a list with the images by *f* of the elements of the original list. It is defined by recursion.

Fixpoint *map* (*A B* : *Set*)(*f* : *A* $\rightarrow$ *B*)(*l* : *list A*) : *list B* :=
  match l with
  | *nil* $\Rightarrow$ *nil*
  | *a::t* $\Rightarrow$ (*f a*)::(*map A B f t*)
  end.

Consider the simplified computation example below, where domains and codomains of *f* are omitted for better readability.

map f (x::y::z::nil) $\rightsquigarrow$ $\cdots$ $\rightsquigarrow$ (*f x*)::(*f y*)::(*f z*)::*nil*

The next two lemmas state *map*'s preserving two functions being inverse and commutativity between *map* and appending. Both lemmas are proved by induction on the list *l*. The second one comes from the Coq Standard Library [1].

Lemma *map_inverse* : $\forall$ (*A B* : *Set*)(*f* : *A* $\rightarrow$ *B*) *g*,

$(\forall \ x, \ g \ (f \ x)=x) \rightarrow \forall \ l, \ map \ g \ (map \ f \ l)=l.$

Lemma *map_app* : $\forall \ (A \ B \ : \ Set)l \ l' \ (f \ : \ A \rightarrow B), \ map \ f \ (l++l') = (map \ f \ l)++(map \ f \ l').$

4.3 Extension of Binary Relations to Lists

Let A be a *Set*. Given a binary relation $P : A \rightarrow A \rightarrow Prop$, the function *rel_vector* expects two lists over A and states that they are component-wise related by P. Note that if they are component-wise related then their lengths are the same.

Fixpoint *rel_vector* $(P \ : \ A \rightarrow A \rightarrow Prop)(l \ l' \ : \ list \ A) \ \{struct \ l\}: \ Prop :=$
match l with
$| \ nil \Rightarrow l'=nil$
$| \ x::l2 \Rightarrow match \ l' \ with$
$\qquad\qquad\qquad | \ nil \Rightarrow False$
$\qquad\qquad\qquad | \ x'::l2' \Rightarrow P \ x \ x' \wedge rel\_vector \ P \ l2 \ l2'$
$\qquad\qquad\qquad end$
end.

The following examples describe three typical computations of the *rel_vector* predicate.

rel_vector $P \ (x::y::nil) \ (x'::y'::nil) \leadsto P \ x \ x' \wedge rel\_vector \ P \ (y::nil) \ (y'::nil) \leadsto P \ x \ x' \wedge P \ y \ y' \wedge rel\_vector \ P \ (nil) \ (nil) \leadsto P \ x \ x' \wedge P \ y \ y' \wedge nil=nil$

Note that $P \ x \ x' \wedge P \ y \ y' \wedge nil=nil$ is equivalent to $P \ x \ x' \wedge P \ y \ y'$.

rel_vector $P \ (x::y::nil) \ (x'::nil) \leadsto P \ x \ x' \wedge rel\_vector \ P \ (y::nil) \ (nil) \leadsto P \ x \ x' \wedge False$

Note that $P \ x \ x' \wedge False$ is equivalent to *False*.

rel_vector $P \ (x::nil) \ (x'::y'nil) \leadsto P \ x \ x' \wedge rel\_vector \ P \ (nil) \ (y'::nil) \leadsto P \ x \ x' \wedge y'::nil=nil$

Note that $P \ x \ x' \wedge y'::nil=nil$ is equivalent to *False* since $y'::nil=nil$ is equivalent to *False*.

The following lemma states that if two lists are component-wise related, then two elements occurring at the same place in each list are also related.

Lemma *rel_vector_app_cons_same_length* : $\forall \ P \ a \ a' \ m \ m' \ l \ l',$
rel_vector $P \ (l++a::m) \ (l'++a'::m') \rightarrow length \ l=length \ l' \rightarrow P \ a \ a'.$

Proof Let P be a binary relation over A, let a and a' be in A, and let m and m' be lists over A. Prove $\forall \ l \ l', \ rel\_vector \ P \ (l++a::m) \ (l'++a'::m') \rightarrow length \ l=length \ l' \rightarrow P \ a \ a'$ by induction on l. For the inductive case, implying that l' is non-empty, apply the induction hypothesis with the tail of l'. □

The next result shows that if two lists are component-wise related, then given one element in the second list, one can compute an element of the first list such that both elements are related.

Lemma *rel_vector_app_cons_exists* : $\forall$ *P a q l m*,
rel_vector P l (m++a::q) $\rightarrow$ $\{x : A \mid In\ x\ l \wedge P\ x\ a\}$.

Proof   By induction on *l* and case splitting on *m* being empty. For the inductive case, if *m* is empty then the head of *l* is a witness, if *m* is not empty then use the induction hypothesis with the tail of *m*; the computable element is a witness. $\square$

The following lemma says that if two lists are component-wise related, then given one element in the first list and one in the second list, but at different places, there is another element in the first list, either before or after the element mentioned first, that is related to the element of the second list.

Lemma *rel_vector_app_cons_different_length* : $\forall$ *P l a m l' a' m'*,
rel_vector P (l++a::m) (l'++a'::m') $\rightarrow$ *length l$\neq$length l'* $\rightarrow$ $\{x : A \mid (In\ x\ l \vee In\ x\ m) \wedge P\ x\ a'\}$.

Proof   By induction on *l*. For the base case, *l* is empty, if *l'* is empty then it is straightforward, if *l'* is not empty then applying *rel_vector_app_cons_exists* gives a witness. For the inductive case, *l* is not empty, case split on *l'* being empty and use the induction hypothesis when *l'* is not empty. $\square$

4.4   No Successor

Let *A* be a *Set*. Given a binary relation, the predicate *is_no_succ* returns a proposition saying that a given element is the predecessor of no element in a given list.

Definition *is_no_succ (P : A $\rightarrow$ A $\rightarrow$ Prop)(x : A)(l : list A) :=*
listforall (fun y $\rightarrow$ $\neg$P x y) l.

The next two results show a transitivity property and decidability of *is_no_succ* when built on a decidable binary relation. Both are proved by induction on the list *l*.

Lemma *is_no_succ_trans* : $\forall$ *P x y l*,
transitive A P $\rightarrow$ *P x y* $\rightarrow$ *is_no_succ P x l* $\rightarrow$ *is_no_succ P y l*.

Lemma *is_no_succ_dec* : $\forall$ *P, rel_dec P* $\rightarrow$ $\forall$ *x l*,
$\{is\_no\_succ\ P\ x\ l\}+\{\neg is\_no\_succ\ P\ x\ l\}$.

The following lemma helps generalise the notion of "backward induction" in section 9. Assume *P* a decidable binary relation over *A*, and *x::l* a non-empty list over *A*. The list *x::l* can be computably split into a left list, a chosen element, and a right list such that 1) the chosen element has no *P*-successor in the right list, 2) the chosen element is the first (from left to right) element with such
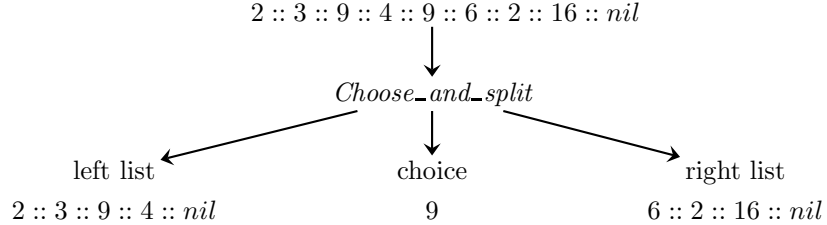
13

a property, and moreover 3) if P is irreflexive and transitive then the chosen element has no P-successor in the left list either. The form of the statement has being slightly simplified, as compared to the actual Coq code. The first conjunct corresponds to the splitting of the list x::l, and the last three conjuncts correspond to the points 1), 2) and 3)as mentioned above.

Lemma *Choose_and_split* : $\forall$ P, *rel_dec* P $\rightarrow$ $\forall$ (l : *list A*)(x : A),
{(*left*,*choice*,*right*) : *list A* $\times$ A $\times$ *list A* |

x::l=(*left*++(*choice*::*right*))
$\wedge$
is_no_succ P *choice* *right*
$\wedge$
($\forall$ *left' choice' right'*, *left*=*left'*++(*choice'*::*right'*) $\rightarrow$
$\neg$*is_no_succ* P *choice'* (*right'*++(*choice*::*right*)))
$\wedge$
(*irreflexive* P $\rightarrow$ *transitive* P $\rightarrow$ *is_no_succ* P *choice* *left*)}.

Proof Assume that P is decidable and proceed by induction on l, the tail of the non-empty list. For the base case where the tail is empty, *nil*, x, and *nil* are witnesses for the required left list, choice element, and right list. For the step case, l=a::l', case split on x having a successor in l. If not, then *nil*, x, and l are a witness. If x has a successor in l then use the induction hypothesis with a, which splits the list l into a left list, a choice element, and a right list. Put x on the top of the left list, here are the three witnesses. $\square$

For example, consider the partial order induced by divisibility of natural numbers by natural numbers. For the list $2 :: 3 :: 9 :: 4 :: 9 :: 6 :: 2 :: 16 :: nil$, the *choice* is 9, the left list is $2 :: 3 :: 9 :: 4 :: nil$, and the right list is $6 :: 2 :: 16 :: nil$.

$$2 :: 3 :: 9 :: 4 :: 9 :: 6 :: 2 :: 16 :: nil$$

$$\downarrow$$

Choose_and_split

left list	choice	right list
$2 :: 3 :: 9 :: 4 :: nil$	9	$6 :: 2 :: 16 :: nil$

Indeed, the first 2 can divide 4 written on its right, 3 can divide 9 written on its right, the first 9 can divide the other 9 written on its right, the 4 can divide 16 written on its right, but the second 9 cannot divide any number written on its right.

5   Sequential Games

By abstracting over payoff functions, subsection 5.1 generalises the notion of sequential game. In the remainder of this paper, the expression "sequential game"

refers to the new and abstract sequential games, unless stated otherwise. In addition, subsection 5.1 introduces a new formalism for sequential games. In subsection 5.2, an inductive proof principle is designed for these sequential games. Last, subsection 5.3 defines a function related to sequential games.

5.1 Definition of Sequential Games

This subsection first presents sequential games informally in the way traditional sequential games were presented in section 2. Then it describes sequential games in a both inductive and graphical formalism, and makes a link with the traditional formalism. Last, the inductive and graphical formalism is naturally translated into a definition of sequential games in Coq.

The Traditional Way: Informally, consider a collection of outcomes that are the possible end-of-the-game situations, and a collection of agents that are the possible stake-holders and decision makers. Roughly speaking, an abstract sequential game is a traditional sequential game where each real-valued payoff function (enclosed in a leaf) has been replaced by an outcome. Below, the left-hand picture represents a traditional sequential game, and the right-hand picture represents an abstract sequential game on the same tree structure.

The Inductive and Graphical Way: In what follows, a generic game g will be represented by the left-hand picture below and a generic (possibly empty) list of games by the right-hand picture.

Sequential games are inductively defined in two steps. First step: If oc is an outcome then the object below is a game.


Second step: if a is an agent and if the first two objects below are a game and a list of games, then the rightmost object is also a game.
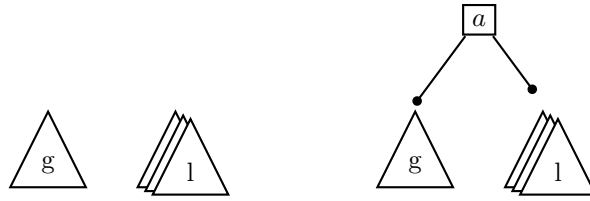
15

The major difference between the traditional formalism and the new formalism is as follows: In the traditional formalism, an internal node had one or arbitrarily many more children, which are games. In the new formalism instead, an internal node has one left child, which is a game, and one right child, which is a list of games. The next two pictures represent the same game in both formalisms, where for all i between 0 and n, the object g_i' is the translation of g_i from the traditional formalism into the new formalism.
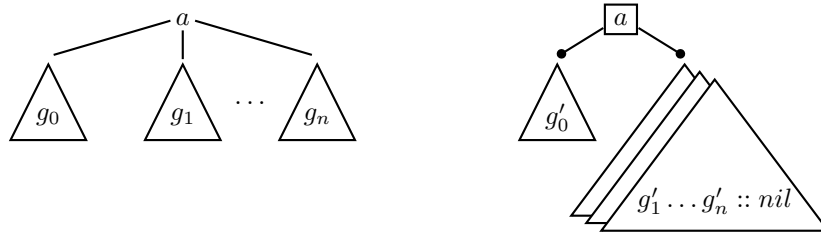
For instance, let a, b and c be three agents, and oc_1, oc_2, oc_3 and oc_4 be four outcomes. Consider the following game in the traditional formalism.
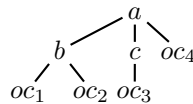
The game above is represented by the following game in the new formalism.

16
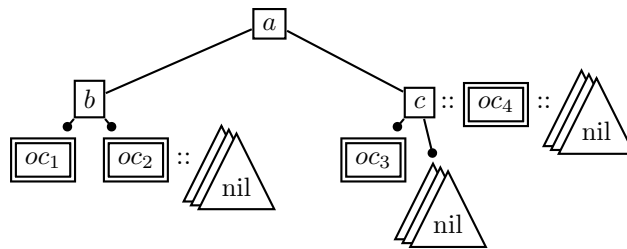
The Inductive and Formal Way: Formally in Coq, games are defined as follows. First define *Outcome* and *Agent*, two arbitrary objects of type *Set*.

Variable *Outcome* : *Set*.
Variable *Agent* : *Set*.

Next, the object *Game* is defined by induction.

Inductive *Game* : *Set* :=
   | *gL* : *Outcome* → *Game*
   | *gN* : *Agent* → *Game* → *list Game* → *Game*.

gL stands for "game leaf" and *gN* for "game node". If *oc* is an *Outcome*, then *gL oc* is a *Game*. If *a* is an *Agent*, *g* is a *Game*, and *l* is a list of objects in *Game*, then *gN a g l* is a *Game*. The interpretation of such an object is as follows: the agent *a* "owns" the root of *gN a g l*, and *g* and *l* represent *a*'s options, *i.e.*, the subgames *a* can decide to continue the play in. The structure ensures that any node of the tree meant to be internal has a non-zero number of children. The inductive Coq formalism and the inductive graphical formalism are very close to each other. For instance, compare the game *gN a* (*gN b oc1 oc2*::*nil*) *oc3*::*nil* with its representation in the inductive graphical formalism below.
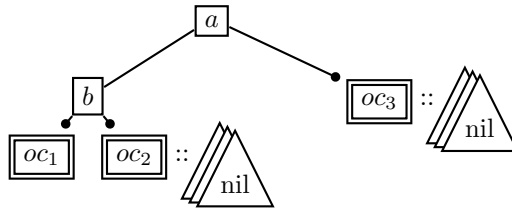
Note that the inductive definition of games involves lists, which are already an inductive structure. Therefore, the game structure is inductive "horizontally and vertically".

5.2 Induction Principle for Sequential Games

This subsection first discusses what would be an inductive proof principle for games in the traditional formalism. The failure of the former leads to an induction principle in the inductive graphical formalism. Then, the induction principle is easily translated in Coq formalism.

The Traditional Way: In the traditional formalism, a so-called induction principle would read as follows. In order to check that a predicate holds for all games, check two properties: First, check that the predicate holds for all games that are leaves (enclosing an outcome). Second, check that if the predicate holds for all the following games,
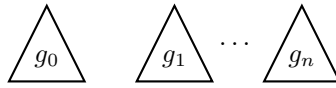
then, for any agent a, it holds for the following game.
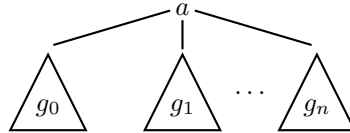
However, dots, etc's and so on, very seldom suit formal proofs. While some dots may be easily formalised, some others cannot. In the new formalism proposed for games, they are formalised by lists. So, an induction principle for games must take lists into account.

The Inductive and Graphical Way: In order to prove that the predicate P holds for all games, it suffices to design a predicate Q expecting a game and a list of games, and then to check that all the properties listed below hold.

– For all outcome oc, P ($\boxed{oc}$).

– For all game , if P ( ) then Q (,)

– For all game and and all list of game ,

if P ( ) and Q (,) then Q (,)

– For all agent a, all game and all list of game ,

if Q (,) then P ( )

18

The Inductive and Formal Way: The induction principle that Coq automatically associates to the inductively defined structure of games is not efficient. A convenient principle has to be built (and hereby proved) manually, *via* a recursive function and the command *fix*. That leads to the following induction principle, which is therefore a theorem in Coq.

$Game\_ind2 : \forall (P : Game \rightarrow Prop) (Q : Game \rightarrow list\ Game \rightarrow Prop),$
$(\forall\ oc,\ P\ (gL\ oc)) \rightarrow$
$(\forall\ g,\ P\ g \rightarrow Q\ g\ nil) \rightarrow$
$(\forall\ g,\ P\ g \rightarrow \forall\ g'\ l,\ Q\ g'\ l \rightarrow Q\ g'\ (g :: l)) \rightarrow$
$(\forall\ g\ l,\ Q\ g\ l \rightarrow \forall\ a,\ P\ (gN\ a\ g\ l)) \rightarrow$
$\forall\ g,\ P\ g$

Two facts are worth noting: First, this principle corresponds to the one stated above in the inductive graphical formalism. Second, in order to prove a property $\forall\ g : Game,\ P\ g$ with the induction principle $Game\_ind2$, the user has to imagine a workable predicate Q.

5.3 Outcomes Used in a Sequential Game

This subsection discusses the notion of outcomes used in a sequential game and defines a function that returns the "left-to-right" list of all the outcomes involved in a given game. Two lemmas follow the definition of the function in the Coq formalism.

The Traditional Way: The example below explains what function is intended.
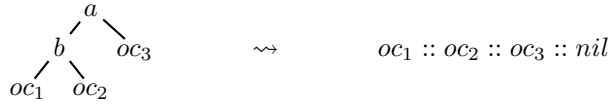
The Inductive and Graphical Way: To prepare the full formalisation of the idea above, the intended function is defined inductively with the inductive graphical formalism, in two steps along the structure of games. First step: the only outcome used in a leaf is the outcome enclosed in the leaf.

Second step: recall that $++$ refers to lists appending/concatenation.
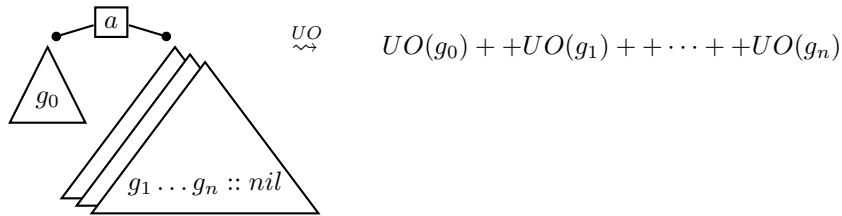
The Inductive and Formal Way: The intended function, called *UsedOutcomes*, is inductively and formally defined in Coq.

Fixpoint *UsedOutcomes* (*g* : *Game*) : *list Outcome* :=
match g with
| *gL oc* ⇒ *oc::nil*
| *gN _ g' l* ⇒ ((*fix ListUsedOutcomes* (*l'* : *list Game*) : *list Outcome* :=
                    match l' with
                    | *nil* ⇒ *nil*
                    | *x::m* ⇒ (*UsedOutcomes x*)++(*ListUsedOutcomes m*)
                    end) (*g'::l*))
end.

The following lemma states that the outcomes used in a game are also used in a structurally bigger game.

Lemma *UsedOutcomes_gN* : ∀ *a g g' l*,
In g (*g'::l*) → *incl* (*UsedOutcomes g*) (*UsedOutcomes* (*gN a g' l*)).

Proof By induction on *l*. For the inductive step case split as follows. If *g* equals the head of *l* then invoke *incl_appr*, *incl_appl*, and *incl_refl*. If *g* either equals *g'* or occurs in the tail of *l* then the induction hypothesis says that the outcomes used by *g* are used by *g'* and *l* together. The remainder mainly applies *in_app_or* and *in_or_app*. □

The next result shows that if the outcomes used in a game all occur in a given list, then so do the outcomes used in any subgame of the original game.

Lemma *UsedOutcomes_gN_listforall* : ∀ *a g loc l*,
incl (*UsedOutcomes* (*gN a g l*)) *loc* → *listforall* (*fun g'* : *Game* ⇒ *incl* (*UsedOutcomes g'*) *loc*) (*g::l*).

Proof All the following cases invoke *in_or_app*. It is straightforward for *g*. For elements of *l* proceed by induction on *l*. For the inductive step, it is straightforward for the head of *l*, and use the induction hypothesis for elements of the tail of *l*. □

6 Strategy Profiles

Consistent with subsection 5.1, subsection 6.1 generalises the notion of strategy profile by abstracting over payoff functions, and introduces a new formalism for them. In the remainder of this paper, the expression "strategy profile" refers to the new and abstract strategy profiles, unless stated otherwise. In subsection 6.2, an inductive proof principle is designed for these strategy profiles. Subsection 6.3 defines the underlying game of a strategy profile. Last, subsection 6.4 defines the induced outcome of a strategy profile.

6.1 Definition of Strategy Profiles

This subsection first presents strategy profiles informally in the way traditional strategy profiles were presented in section 2. Then it describes strategy profiles in a both inductive and graphical formalism consistent with the one used to define sequential games. A correspondence between the traditional formalism and the new one is established. Finally, the inductive and graphical formalism is naturally translated into a definition of strategy profiles in Coq.

The Traditional Way: Roughly speaking, an abstract strategy profile is a traditional strategy profile where each real-valued payoff function (enclosed in a leaf) has been replaced by an outcome. Below, the left-hand picture represents a traditional strategy profile, and the right-hand picture represents an abstract strategy profile on the same tree structure.


The Inductive and Graphical Way: A generic strategy profile s will be represented by the left-hand picture below and a generic (possibly empty) list of strategy profiles by the right-hand picture.

Strategy profiles are inductively defined in two steps. First step: If oc is an outcome then the object below is a strategy profile.


Second step: if a is an agent and the first three objects below are a strategy profile and two lists of strategy profiles, then the rightmost object is also a strategy profile.
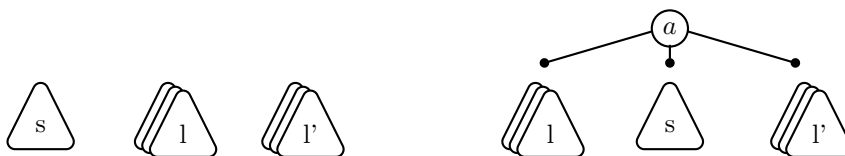

The major difference between the traditional formalism and the new formalism is as follows: In the traditional formalism, an internal node has one or

arbitrarily many more children, which are strategy profiles, and in addition an internal node is linked to one and only one of its children by a double line. In the new formalism, an internal node has one left child, which is a list of strategy profiles, one central child, which is a strategy profile corresponding to the double line, and one right child, which is a list of strategy profiles. The next two pictures represent the same strategy profile in both formalisms, where for all i between 0 and n, the object s_i' is the translation of s_i from the traditional formalism into the new formalism.
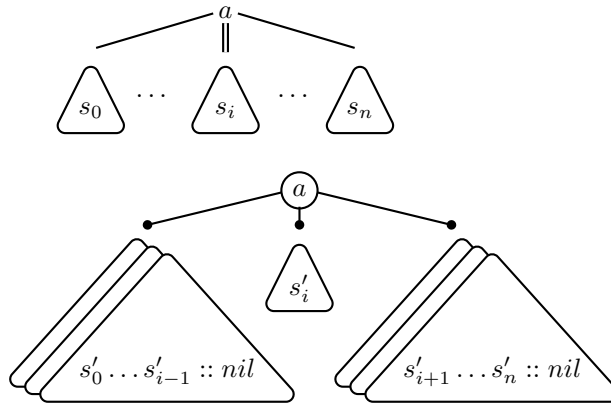
For instance, let a and b be two agents, and oc_1, oc_2 and oc_3 be three outcomes. Consider the following strategy profile in the traditional formalism.
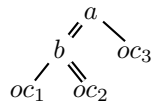
The strategy profile above is represented by the following strategy profile in the new formalism.

22
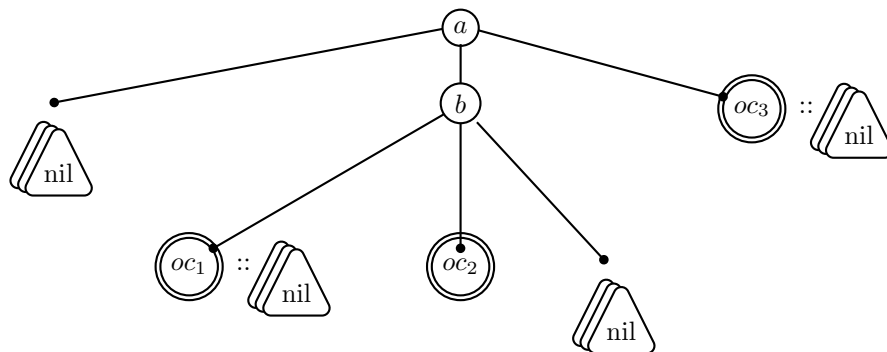
The Inductive and Formal Way: Formally in Coq, strategy profiles are defined as follows.

Inductive *Strat* : *Set* :=
   | *sL* : *Outcome* → *Strat*
   | *sN* : *Agent* → *list Strat* → *Strat* → *list Strat* → *Strat*.

sL stands for "strategy profile leaf" and *sN* for "strategy profile node". If *oc* is an *Outcome*, then *sL oc* is a *Strat*. If *a* is an *Agent*, if *s* is a *Strat*, and if *l* and *l'* are lists of objects in *Strat*, then *sN a l s l'* is a *Strat*. The interpretation of such an object is as follows: as for sequential games, the agent *a* "owns" the root of *sN a l s l'*. Moreover, *s* is the substrategy profile *a* has decided the play shall continue in, and *l* and *l'* represent the options dismissed by *a* on the left and on the right of *s*. The structure ensures that any node of the tree meant to be internal has a non-zero number of children, and that one and only one child has been chosen. Like for sequential games, the inductive graphical formalism and the inductive Coq formalism are very close to each other for strategy profiles. For instance, compare the strategy profile *sN a nil* (*sN b oc1::nil oc2 nil*) *oc3::nil* with its representation in the inductive graphical formalism above.

6.2 Induction Principle for Strategy Profiles

This subsection discusses what would be an inductive proof principle for strategy profiles in the traditional formalism. The failure of the former leads to an induction principle in the inductive Coq formalism.

The Traditional Way: In the traditional formalism, a so-called induction principle would read as follows. In order to check that a predicate holds for all strategy profiles, check two properties: First, check that the predicate holds for all strategy profiles that are leaves (enclosing an outcome). Second, check that if the predicate holds for all the following strategy profiles,

then, for any agent *a* and any *i* between 0 and *n*, it holds for the following strategy profile.
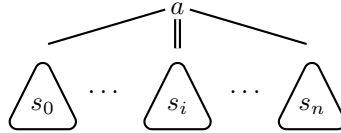
However, dots, etc's and so on, very seldom suit formal proofs. While some dots may be easily formalised, some others cannot. In the inductive formalism proposed in this paper, they are formalised by lists. So, an induction principle for games must take lists into account.
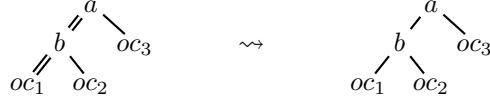
The Inductive and Formal Way: The induction principle that Coq automatically associates to the inductively defined structure of strategy profiles is not efficient. A convenient principle has to be built (and hereby proved) manually, *via* a recursive function and the command *fix*. That leads to the following induction principle, which is therefore a theorem in Coq. In order to prove that a predicate P holds for all strategy profiles, it suffices to design a predicate Q expecting a strategy profile and a list of strategy profiles, and then to check that several fixed properties hold, as formally stated below.

$Strat\_ind2 : \forall (P : Strat \rightarrow Prop) (Q : list\ Strat \rightarrow Strat \rightarrow list\ Strat \rightarrow Prop),$
$(\forall\ oc,\ P\ (sL\ oc)) \rightarrow$
$(\forall\ sc,\ P\ sc \rightarrow Q\ nil\ sc\ nil) \rightarrow$
$(\forall\ s,\ P\ s \rightarrow \forall\ sc\ sr,\ Q\ nil\ sc\ sr \rightarrow Q\ nil\ sc\ (s :: sr)) \rightarrow$
$(\forall\ s,\ P\ s \rightarrow \forall\ sc\ sl\ sr,\ Q\ sl\ sc\ sr \rightarrow Q\ (s :: sl)\ sc\ sr) \rightarrow$
$(\forall\ sc\ sl\ sr,\ Q\ sl\ sc\ sr \rightarrow \forall\ a,\ P\ (sN\ a\ sl\ sc\ sr)) \rightarrow$
$\forall\ s,\ P\ s$

6.3 Underlying Game of a Strategy Profile

This subsection discusses the notion of underlying game of a given strategy profile and defines a function that returns such a game. One lemma follows the definition of the function in the Coq formalism.

The Traditional Way: The example below explains what function is intended. It amounts to forgetting all the choices.

The Inductive and Graphical Way: To prepare the full formalisation of the idea above, the intended function is defined inductively with the inductive graphical formalism, in two steps along the structure of strategy profiles. First step: the underlying game of a strategy profile where no choice has been made, *i.e.*, a leaf strategy profile, is a game where no choice is required, *i.e.*, a leaf game.
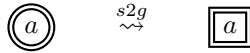
The second step needs case splitting along the structure of the first list, *i.e.*, whether it is empty or not.
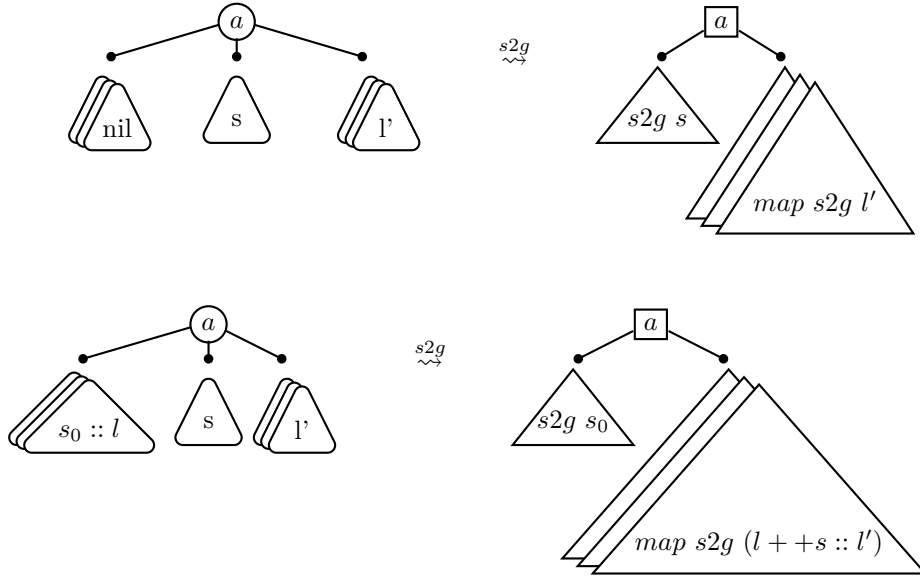
The Inductive and Formal Way: The intended function is inductively and formally defined in Coq, and called *s2g*, which stands for "strategy profile to game".

Fixpoint *s2g* (*s* : *Strat*) : *Game* :=
match s with
| *sL oc* ⇒ *gL oc*
| *sN a sl sc sr* ⇒
      match sl with
      | *nil* ⇒ *gN a* (*s2g sc*) (*map s2g sr*)
      | *s'::sl'* ⇒ *gN a* (*s2g s'*) ((*map s2g sl'*)++(*s2g sc*)::(*map s2g sr*))
      end
end.

The next result states that if the two lists of substrategy profiles of two strategy profiles have, component-wise, the same underlying games, then the two original strategy profiles also have the same underlying game.

Lemma *map_s2g_sN_s2g* : ∀ *a sl sc sr sl' sc' sr'*,
*map s2g* (*sl ++ sc :: sr*) = *map s2g* (*sl' ++ sc' :: sr'*) →
s2g (*sN a sl sc sr*) = *s2g* (*sN a sl' sc' sr'*).

Proof   Double case split on *sl* and *sl'* being empty and use *map_app*.   □
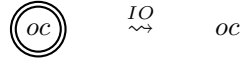
6.4   Induced Outcome of a Strategy Profile

This subsection discusses the notion of outcome induced by a strategy profile, and defines a function that computes such an outcome. Three lemmas follow the definition of the function in the Coq formalism.

The Traditional Way: Starting at the root of a strategy profile and following the agents' consecutive choices leads to a leaf, and hereby to an outcome. The following example explains what function is intended.
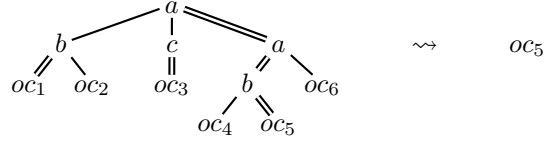
The Inductive and Graphical Way: To prepare the full formalisation of the idea above, the intended function is defined inductively with the inductive graphical formalism, in two steps along the structure of strategy profiles. First step: the outcome induced by a leaf strategy profile is the enclosed outcome.

Second step: follow the choice at the root and consider the chosen substrategy profile.
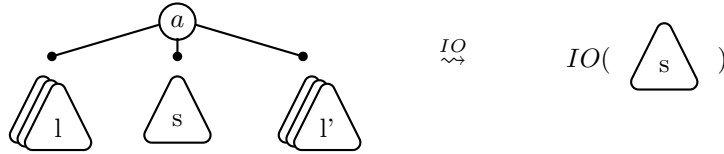
The Inductive and Formal Way: The intended function, called *InducedOutcome*, is inductively and formally defined in Coq.

Fixpoint *InducedOutcome* $(s : Strat) : Outcome :=$
match s with
| *sL oc* $\Rightarrow$ *oc*
| *sN a sl sc sr* $\Rightarrow$ *InducedOutcome sc*
end.

The following lemma, proved by induction on *sl*, states that the outcomes used by the underlying game of a strategy profile are all used by the underlying game of a bigger strategy profile whose chosen substrategy profile is the original strategy profile.

Lemma *UsedOutcomes_sN_incl* : $\forall$ *a sl sc sr*,
incl (*UsedOutcomes* (*s2g sc*)) (*UsedOutcomes* (*s2g* (*sN a sl sc sr*))).

The next result says that the outcome induced by a strategy profile is also an outcome used in the underlying game.

Lemma *Used_Induced_Outcomes* : ∀ *s* : *Strat*,
In (*InducedOutcome s*) (*UsedOutcomes* (*s2g s*)).

Proof Write the claim as a predicate on *s* and apply the induction principle *Strat_ind2* where *Q l s l'* is *In* (*InducedOutcome s*) (*UsedOutcomes* (*s2g s*)). For the last induction case invoke *UsedOutcomes_sN_incl*. □

Note that if the underlying game of a strategy profile is a leaf game enclosing a given outcome, then the strategy profile is a leaf strategy profile enclosing the same outcome, so that the outcome induced by the strategy profile is also the same outcome: if *s2g s=gL oc* then *InducedOutcome s=oc*. The following lemma is the list version of the remark above. It is prove by induction on *ls*.

Lemma *map_s2g_gL_InducedOutcome* : ∀ *ls loc*,
*map s2g ls =map* (*fun y ⇒ gL y*) *loc → map InducedOutcome ls = loc*.

7 Convertibility

The first subsection defines convertibility, which will be used to formally define the notion of Nash equilibrium in section 8. The second subsection designs an inductive proof principle for convertibility.
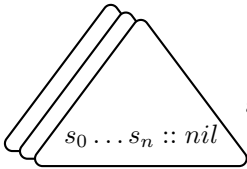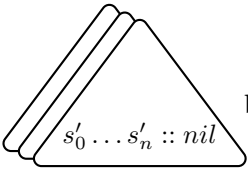
7.1 Definition of Convertibility

This subsection discusses the notion of convertibility, which is the ability of an agent to convert a strategy profile into another strategy profile, and defines a predicate accounting for it. Four lemmas follow its definition in Coq.

The Traditional Way: An agent is (usually implicitly) granted the ability to change his choices at all nodes he owns, as shown in the following example. Agent *b* can change his choices at the nodes where **b** is displayed in bold font, and only at those nodes.
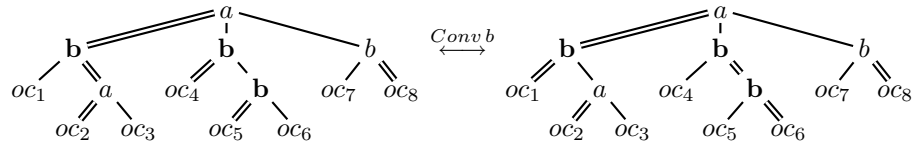
The Inductive and Graphical Way: To prepare the full formalisation of the idea above, the intended function is defined inductively with the inductive graphical formalism, in two steps along the structure of strategy profiles. Let *b* be an agent. First step: Let *oc* be an outcome. Agent *b* can convert the leaf strategy profile enclosing *oc* into itself by changing some, actually none of his choices since none has been made.

$$oc \overset{Conv\,b}{\longleftrightarrow} oc$$
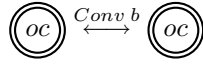
Second step: let $s_0 \ldots s_n :: nil$ and $s'_0 \ldots s'_n :: nil$ be two lists

of strategy profiles such that for all i between 0 and n, $s_i \overset{Conv\,b}{\longleftrightarrow} s'_i$.

The second step involves compound strategy profiles, and needs case splitting on the "converting agent" owning the roots or not. Also let a be another agent. For all i between 0 and n,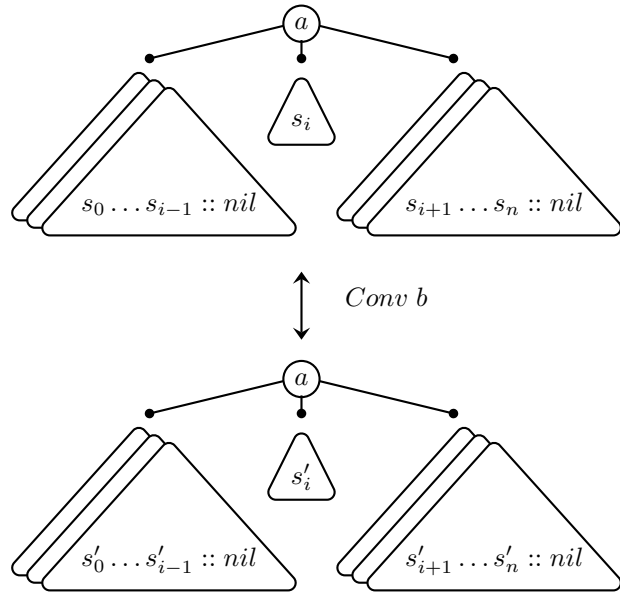 agent b can perform the following conversion by combining at once all his conversion abilities in all the substrategy profiles.

$$a \quad s_i \quad [\, s_0 \ldots s_{i-1} :: nil \,] \quad [\, s_{i+1} \ldots s_n :: nil \,]$$

$$\updownarrow \quad Conv\,b$$

$$a \quad s'_i \quad [\, s'_0 \ldots s'_{i-1} :: nil \,] \quad [\, s'_{i+1} \ldots s'_n :: nil \,]$$

When a equals b, the agent b can also change his choice at the root owned by a. In that case, the agent b can perform the following conversion for all i and k between 0 and n.
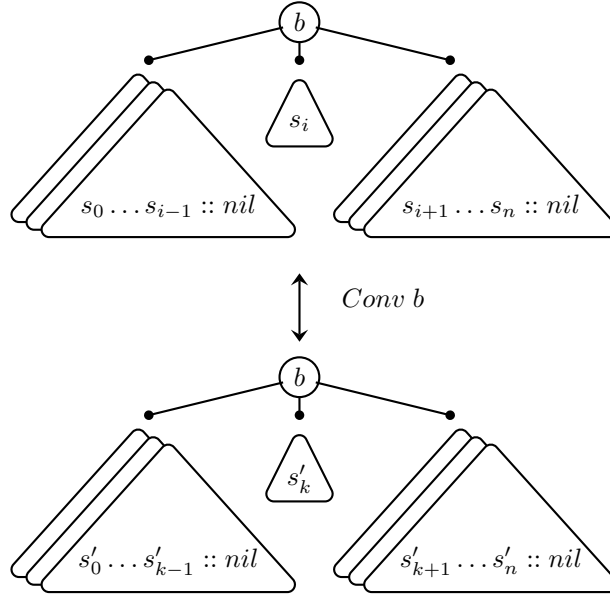
The Inductive and Formal Way: The following definition accounts for agents being able to unilaterally change part of their choices. *Conv*, which means strategy profile convertibility, and *ListConv*, which means component-wise convertibility of list of strategy profiles, are defined by mutual induction *via* the word *with*, within the same definition.

Inductive $Conv : Agent \rightarrow Strat \rightarrow Strat \rightarrow Prop :=$
| $convLeaf : \forall b\ oc,\ Conv\ b\ (sL\ oc)(sL\ oc)$
| $convNode : \forall b\ a\ sl\ sl'\ sc\ sc'\ sr\ sr',\ (length\ sl=length\ sl' \vee a = b) \rightarrow$
$ListConv\ b\ (sl{+}{+}(sc{::}sr))\ (sl'{+}{+}(sc'{::}sr')) \rightarrow$
$Conv\ b\ (sN\ a\ sl\ sc\ sr)(sN\ a\ sl'\ sc'\ sr')$

$with$

$ListConv : Agent \rightarrow list\ Strat \rightarrow list\ Strat \rightarrow Prop :=$
| $lconvnil : \forall b,\ ListConv\ b\ nil\ nil$
| $lconvcons : \forall b\ s\ s'\ tl\ tl',\ Conv\ b\ s\ s' \rightarrow ListConv\ b\ tl\ tl' \rightarrow$
$ListConv\ b\ (s{::}tl)(s'{::}tl').$

The formula *length sl=length sl'* $\vee$ *a = b* ensures that only the owner of a node can change his choice at that node. It corresponds to the case splitting in the inductive and graphical definition above. *ListConv b* $(sl{+}{+}(sc{::}sr))$ $(sl'{+}{+}(sc'{::}sr'))$ guarantees that this property holds also in the substrategy profiles. *ListConv* corresponds to the dots in the inductive and graphical definition above. One may think of avoiding a mutual definition for *Conv* by using the

29

already defined *rel_vector* instead of *ListConv*. However, the Coq versions 8.0 and 8.1 do not permit it, presumably because it would require to pass the whole *Conv* object as an argument (to *rel_vector*) while not yet fully defined.

The following two lemmas establish the equivalence between *ListConv* on the one hand and *rel_vector* and *Conv* on the other hand. They are both proved by induction on the list *l* and by a case splitting on *l'*.

Lemma *ListConv_rel_vector* : ∀ *a l l'*,
ListConv a l l' → *rel_vector* (*Conv a*) *l l'*.

Lemma *rel_vector_ListConv* : ∀ *a l l'*,
rel_vector (*Conv a*) *l l'* → *ListConv a l l'* .

The next two results state reflexivity property of *Conv* and *ListConv*.

Lemma *Conv_refl* : ∀ *a s* , *Conv a s s*.

Proof   Let *a* be an agent and *s* a strategy profile. It suffices to prove that *P s* where *P s'* is *Conv a s' s'* by definition. Apply the induction principle *Strat_ind2* where *Q sl sc sr* is *ListConv a* (*sl++sc::sr*) (*sl++sc::sr*) by definition. Checking all cases is straightforward.   □

Lemma *ListConv_refl* : ∀ *a l*, *ListConv a l l*.

Proof   By *rel_vector_ListConv*, induction on the list *l*, and *Conv_refl*.   □

7.2   Induction Principle for Convertibility

The Inductive and Formal Way: A suitable induction principle for convertibility is automatically generated in Coq with the command *Scheme*, and hereby is a theorem in Coq. The principle states that under four conditions, *Conv a* is a subrelation of the binary relation *P a* for all agents *a*. Note that in order to prove such a property, the user has to imagine a workable predicate *P0*.

conv_lconv_ind : ∀ (*P* : *Agent* → *Strat* → *Strat* → *Prop*)
(*P0* : *Agent* → *list Strat* → *list Strat* → *Prop*),

(∀ *b oc*, *P b* (*sL oc*) (*sL oc*)) →

(∀ *b a sl sl' sc sc' sr sr'*, *length sl* = *length sl'* ∨ *a* = *b* → *ListConv b* (*sl* ++ *sc* :: *sr*) (*sl'* ++ *sc'* :: *sr'*) → *P0 b* (*sl* ++ *sc* :: *sr*) (*sl'* ++ *sc'* :: *sr'*) → *P b* (*sN a sl sc sr*) (*sN a sl' sc' sr'*)) →

(∀ *b*, *P0 b nil nil*) →

(∀ *b s s' tl tl'*, *Conv b s s'* → *P b s s'* → *ListConv b tl tl'* → *P0 b tl tl'* → *P0 b* (*s* :: *tl*) (*s'* :: *tl'*)) →

∀ *a s s0*, *Conv a s s0* → *P a s s0*

This induction principle is used below to prove that two convertible strategy profiles have the same underlying game.

30

Lemma $Conv\_s2g$: $\forall$ $(a : Agent)(s\ s' : Strat)$, $Conv\ a\ s\ s' \rightarrow s2g\ s = s2g\ s'$.

Proof   Assume s and s' convertible by a. Write $s2g\ s = s2g\ s'$ as $P\ a\ s\ s'$ for some P and proceed by the induction principle $conv\_lconv\_ind$ where $P0\ b$ $l\ l'$ is $(ListConv\ b\ l\ l' \rightarrow map\ s2g\ l{=}map\ s2g\ l')$ by definition. The remainder invokes $map\_s2g\_sN\_s2g$.   □

8   Concepts of Equilibrium

This section defines the notions of preference, happiness, Nash equilibrium, and subgame perfect equilibrium in the new and abstract formalism. Two lemmas follow these definitions.

In traditional game theory, the agents implicitly prefer strictly greater payoffs, and thus also prefer payoff functions granting them strictly greater payoffs. In the abstract formalism, the agents' preferences for outcomes are explicitly represented by binary relations over the outcomes, one relation per agent. Below, $OcPref\ a$ is the preference of agent a.

Variable $OcPref$: $Agent \rightarrow Outcome \rightarrow Outcome \rightarrow Prop$.

Since every strategy profile induces an outcome, the preferences over outcomes yield preferences over strategy profiles. For instance, if agent a prefers oc_1 to oc_3, then he prefers the following right-hand strategy profile to the left-hand one. As to agent b, it could be either way since no specific property is assumed about preferences.

Formally in Coq, preference over strategy profiles is defined as follows.

Definition $StratPref$ $(a : Agent)(s\ s' : Strat)$: $Prop$:= $OcPref\ a\ (InducedOutcome\ s)(InducedOutcome\ s')$.

If an agent cannot convert a given strategy profile to any preferred one, then he is said to be happy with respect to the given strategy profile. For instance the following strategy profile makes agent a happy *iff* agent a does not prefer oc_2 to oc_3, whatever his other preferences are.
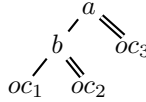
Formally in Coq, happiness of an agent is defined as follows.

Definition $Happy$ $(s : Strat)(a : Agent)$: $Prop$:= $\forall$ s', $Conv\ a\ s\ s' \rightarrow \neg StratPref\ a\ s\ s'$.

A strategy profile that makes every agent happy is called a Nash equilibrium.

Definition *Eq (s : Strat) : Prop := ∀ a, Happy s a*.

A subgame perfect equilibrium is a Nash equilibrium such that all of its substrategy profiles are subgame perfect equilibria. Compare this definition with the informal and more complicated one in section 2.

Fixpoint *SPE (s : Strat) : Prop := Eq s ∧*
match s with
| sL oc ⇒ True
| sN a sl sc sr ⇒ (listforall SPE sl) ∧ SPE sc ∧ (listforall SPE sr)
end.

Therefore, a subgame perfect equilibrium is a Nash equilibrium.

Lemma *SPE_is_Eq : ∀ s : Strat, SPE s → Eq s*.

The following provides a sufficient condition for a strategy profile to be a Nash equilibrium: at the root of a compound strategy profile, if the chosen substrategy profile is a Nash equilibrium, and if the owner of the root cannot convert any of his other options into a substrategy profile that he prefers to his current choice, then the compound strategy profile is also a Nash equilibrium. This is stated in Coq below.

Lemma *Eq_subEq_choice : ∀ a sl sc sr*,
(∀ s s', In s sl ∨ In s sr → Conv a s s' → ¬StratPref a sc s') →
Eq sc → Eq (sN a sl sc sr).

Proof   Let be *a, sl, sc, sr* and assume the two premises. Also assume that an agent *a'* can convert *sN a sl sc sr* to *s'* that he prefers. Now try to derive a contradiction from the hypothesis. For this, note that *s'* equals *sN a sl'* *sc' sr'* for some *sl', sc'*, and *sr'*. Case split on *sl* and *sl'* having or not the same length. If they have the same length then use the equilibrium assumption together with *rel_vector_app_cons_same_length* and *ListConv_rel_vector*. If *sl* and *sl'* have different lengths then *a'* equals *a*. The remainder invokes *rel_vector_app_cons_different_length* and *ListConv_rel_vector*.   □

The converse of this lemma also holds, but it is omitted in this paper.

9   Existence of Equilibria

This section generalises the notion of "backward induction" for abstract sequential games, and shows that it yields a subgame perfect equilibrium when preferences are totally ordered. But it also shows that it may not always yield a subgame perfect equilibrium for arbitrary preferences. However, this section eventually proves that acyclicity of decidable preferences is a necessary and sufficient condition for guaranteeing computable existence of Nash equilibrium/subgame perfect equilibrium.
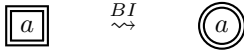
9.1 "Backward Induction"

This subsection starts with an informal discussion, and continues with definitions in the inductive graphical formalism. Eventually, a "backward induction" function is defined in Coq, and one lemma follows.
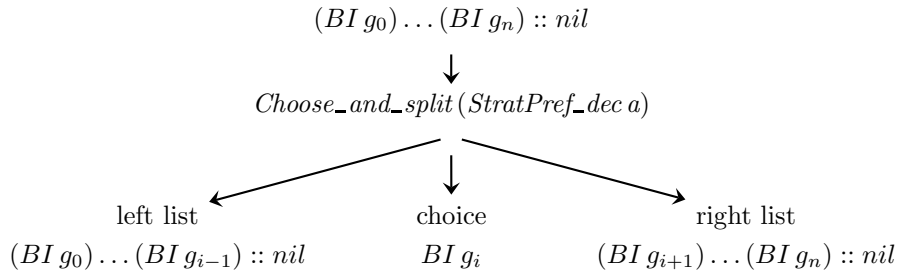
The Traditional Way: Informally, the idea is to perform "backward induction" on all subgames first, and then to let the owner of the root choose one strategy profile that suits him best among the newly built strategy profiles. When preferences are partially ordered, the agent can choose in order to maximise his preference; when they are not partially ordered, a procedure slightly more general may be needed. Lemma *Choose_and_split* defined in subsection 4.4 relates to such a procedure. (More specifically, the proof of the lemma is such a procedure.) For instance, let $oc_1 \dots oc_6$ be six outcomes such that a prefers oc_5 to oc_2, and b prefers oc_2 to oc_1 and oc_1 to oc_2, and nothing else. A "backward induction" process is detailed below. On the left-hand picture, b chooses by *Choose_and_split*. On the right-hand picture, agent a being "aware" of b's choice procedure chooses accordingly, also by *Choose_and_split*.


The Inductive and Graphical Way: To prepare the full formalisation of the idea above, the intended function is defined inductively with the inductive graphical formalism, in two steps along the structure of strategy profiles. First step: performing "backward induction" on a leaf game that encloses an outcome yields a leaf strategy profile that encloses the same outcome.

Second step: Assume that "backward induction" is defined for the games $g_0, \dots, g_n$. An agent a can choose one strategy profile among $BI\, g_0, \dots, BI\, g_n$ by *Choose_and_split* and his own preference, as below.

$$(BI\, g_0) \dots (BI\, g_n) :: nil$$

$$\downarrow$$

$$Choose\_and\_split\,(StratPref\_dec\, a)$$

left list	choice	right list
$(BI\, g_0) \dots (BI\, g_{i-1}) :: nil$	$BI\, g_i$	$(BI\, g_{i+1}) \dots (BI\, g_n) :: nil$

Then, "backward induction" can be defined on the following compound game.

$$\downarrow BI$$
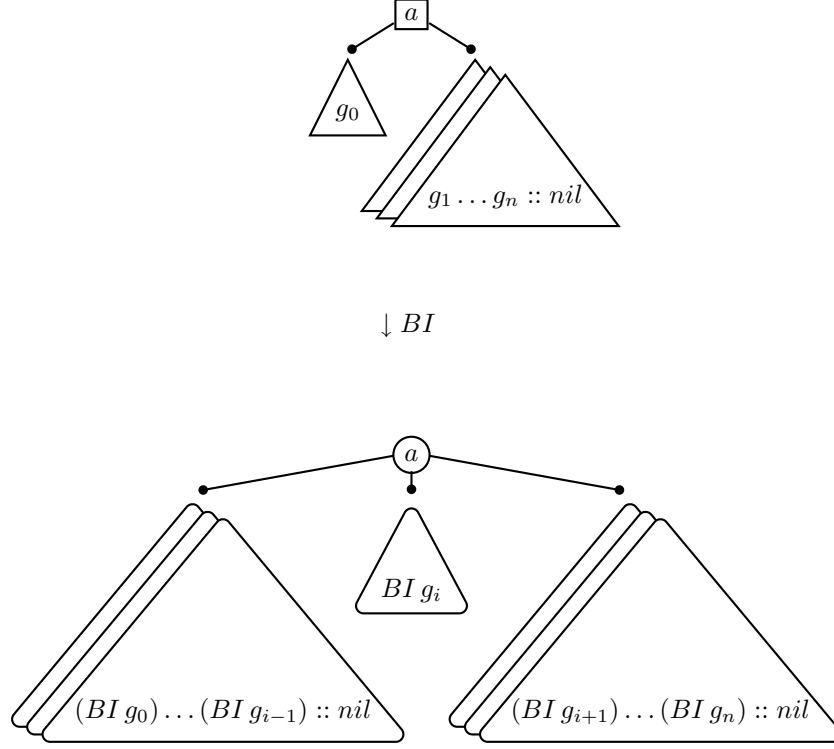
The Inductive and Formal Way: The definition using the inductive graphical formalism above is translated into Coq formalism. First, assume that preferences over outcomes are decidable.

Hypothesis $OcPref\_dec : \forall\ (a : Agent),\ rel\_dec\ (OcPref\ a)$.

Subsequently, preferences over strategy profiles are also decidable.

Lemma $StratPref\_dec : \forall\ (a : Agent),\ rel\_dec\ (StratPref\ a)$.

Next, the generalisation of "backward induction", with respect to the preferences above, is defined by recursion. For the sake of readability, the definition displayed below is a slight simplification of the actual Coq code.

Fixpoint $BI\ (g : Game) : Strat :=$
match g with
| gL oc $\Rightarrow$ sL oc
| gN a g l $\Rightarrow$ let (sl,sc,sr):=
          Choose_and_split (StratPref_dec a) (map BI l) (BI g) in
          sN a sl sc sr
end.

As stated below, the underlying game of the image by *BI* of a given game is the same game.

Lemma *BI_s2g* : ∀ *g* : *Game*, *s2g* (*BI g*)=*g*.

Proof   Rewrite the claim as ∀ *g*, *P g* for some *P*. Apply the induction principle *Game_ind2* where *Q g l* is *s2g* (*BI g*)=*g* ∧ *map s2g* (*map BI*) *l*=*l*. For the last induction case, invoke *Choose_and_split* and *map_app*.   □

9.2   The Total Order Case

In this subsection only, assume transitivity and irreflexivity of preferences over outcomes. Subsequently, those propertiesalso hold for preferences over strategy profiles.

Hypothesis *OcPref_irrefl* : ∀ (*a* : *Agent*), *irreflexive* (*OcPref a*).

Hypothesis *OcPref_trans* : ∀ (*a* : *Agent*), *transitive* (*OcPref a*).

Lemma *StratPref_irrefl* : ∀ (*a* : *Agent*), *irreflexive* (*StratPref a*).

Lemma *StratPref_trans* : ∀ (*a* : *Agent*), *transitive* (*StratPref a*).

Irreflexivity of preferences guarantees that leaf strategy profiles are Nash equilibria.

Lemma *Leaf_Eq* : ∀ *oc* : *Outcome*, *Eq* (*sL oc*).

If preferences are total over a given list of outcomes, then for any sequential game using only outcomes from the list, "backward Induction" yields subgame perfect equilibrium. This is the translation of Kuhn's result into abstract sequential game formalism.

Lemma *BI_SPE* : ∀ *loc*, (∀ (*a* : *Agent*), *total* (*OcPref a*) *loc*) →
∀ *g* : *Game*, *incl* (*UsedOutcomes g*) *loc* → *SPE* (*BI g*).

Proof   Assume *loc*, a list of outcomes, and the totality property. Write *incl* (*UsedOutcomes g*) *loc* → *SPE* (*BI g*) as *P g* and proceed by the induction principle *Game_ind2* where *Q g l* is *listforall* (*fun g'* ⇒ *incl* (*UsedOutcomes g'*) *loc*) (*g::l*) → *SPE* (*BI g*) ∧ *listforall SPE* (*map BI l*). The first three cases are straightforward. For the fourth and last case assume *g*, *l*, and the other premises, such as an agent *a*. In order to prove that the "backward induction" of the compound game *gN a g l* is a subgame perfect equilibrium, first note that all substrategy profiles of this "backward induction" are subgame perfect equilibria, by the induction hypotheses, *listforall_appr* and *listforall_appr*. Then, the main difficulty is to prove that it is a Nash equilibrium. For this, invoke *Eq_subEq_choice* after proving its two premises: The induction hypothesis and lemma *SPE_is_Eq* shows that the substrategy profile chosen by *a* is a Nash equilibrium. For the other premise required for invoking *Eq_subEq_choice*, assume a substrategy profile not chosen by *a*. The induction hypothesis and lemma *SPE_is_Eq* show that it is a Nash equilibrium. Next, assume that this Nash equilibrium is convertible

by a into another strategy profile, and show that a does not prefer this new strategy profile to his current choice. For this, invoke decidability, irreflexivity, transitivity, and totality of preferences, as well as lemmas *listforall_In*, *UsedOutcomes_gN*, *map_inverse*, *Used_Induced_Outcomes* and *Conv_s2g*. □

9.3 Limitation

Now, no property is assumed about the preferences. Consider the three outcomes oc_1, oc_2 and oc_3, and an agent a that prefers oc_2 to oc_3, and nothing else. The strategy profile below is obtained by the "backward induction" function define in subsection 9.1. However, it is not a Nash equilibrium since the current induced outcome is oc_3, but it can be converted by a into the preferred oc_2.
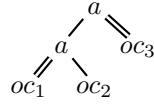
Informally, when dealing with totally ordered preferences, the notions of "backward induction" and subgame perfect equilibrium coincide although their definitions differ; they are the same in extension, but are different in intension. This difference in intension is critical when dealing with partially ordered preferences: as shown by the example above, "backward induction" no longer yields Nash equilibrium, let alone subgame perfect equilibrium.

9.4 General Case

Until this subsection, equilibrium concepts and related notions have been defined with respect to given preferences: a preference binary relation was associated with an agent once for all. In the following, equilibrium concepts and related notions are abstracted over preferences. It means that preferences become a parameter of the definitions and lemmas. For instance, instead of writing *Eq s* to say that s is a Nash equilibrium, one shall write *Eq OcPref s* to say that s is a Nash equilibrium with respect to the family of preferences defined by *OcPref*.

As formally stated in the two lemmas below, given two families of preferences and given a strategy profile, if for every agent the restriction of his first preference to the outcomes used by the strategy profile is a subrelation of his second preference, and if the strategy profile is a Nash equilibrium/subgame perfect equilibrium with respect to the second preferences then it is also a Nash equilibrium/subgame perfect equilibrium with respect to the first preferences. Informally, the less arcs an agent's preference has, the more strategy profiles make the agent happy.

Lemma *Eq_order_inclusion* : $\forall$ *OcPref OcPref' s*,
($\forall$ *a* : *Agent*,
sub_rel (*restriction* (*OcPref a*) (*UsedOutcomes* (*s2g s*))) (*OcPref' a*)) $\rightarrow$
Eq OcPref' s $\rightarrow$ *Eq OcPref s*.

Proof Invoke *Used_Induced_Outcomes* and *Conv_s2g*. □

Lemma *SPE_order_inclusion* : $\forall$ *OcPref OcPref' s*,
($\forall$ *a* : *Agent*,
sub_rel (*restriction* (*OcPref a*) (*UsedOutcomes* (*s2g s*))) (*OcPref' a*)) $\rightarrow$
SPE OcPref' s $\rightarrow$ *SPE OcPref s*.

Proof   Assume two families of preferences and rewrite the claim as $\forall$ *s*, *P s*
for some *P*. Then apply *Strat_ind2* where *Q sl sc sr* is ($\forall$ *a*, *sub_rel* (*restriction*
(*OcPref a*) (*UsedOutcomes* (*s2g* (*sN a sl sc sr*)))) (*OcPref' a*)) $\rightarrow$ *listforall*
(*SPE OcPref'*) *sl* $\rightarrow$ *SPE OcPref' sc* $\rightarrow$ *listforall* (*SPE OcPref'*) *sr* $\rightarrow$ *list-*
forall (*SPE OcPref*) *sl* $\wedge$ *SPE OcPref sc* $\wedge$ *listforall* (*SPE OcPref*) *sr*. For
the first and fifth induction steps, apply *Eq_order_inclusion*. For the third and
fourth induction step, invoke lemmas *transitive_sub_rel*, *sub_rel_restriction_incl*,
incl_appr, *incl_appl*, and *incl_refl*. $\square$

The following lemma generalises Kuhn's result to acyclic preferences (instead
of totally ordered). It invokes a result related to topological sorting proved in [11].

Theorem *acyclic_SPE* : $\forall$ *OcPref*,
($\forall$ *a*, *rel_dec* (*OcPref a*)) $\rightarrow$
($\forall$ *a*, *irreflexive* (*clos_trans Outcome* (*OcPref a*))) $\rightarrow$
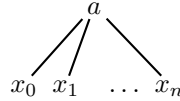 $\forall$ *g*, {*s* : *Strat* | *s2g s=g* $\wedge$ *SPE OcPref s*}.

Proof   Assume a family of decidable acyclic preferences. Let *g* be a game.
According to [11], by topological sorting there exists a family of decidable acyclic
preferences that are strict total orders including the original preferences on the
outcomes used by *g*. A subgame perfect equilibrium can be computed by *BI_SPE*.
Conclude by *SPE_order_inclusion*. $\square$

The following property relates to *SPE_is_Eq*.

Theorem *SPE_Eq* : $\forall$ *OcPref*,
($\forall$ *a*, *rel_dec* (*OcPref a*)) $\rightarrow$
($\forall$ *g* : *Game*, {*s* : *Strat* | *s2g s=g* $\wedge$ *SPE OcPref s*}) $\rightarrow$
$\forall$ *g* : *Game*, {*s* : *Strat* | *s2g s=g* $\wedge$ *Eq OcPref s*}.

The next result says that if all games have Nash equilibria with respect to a
given family of preferences, then these preferences are acyclic.

The Traditional Way   Informally, let an agent *a* prefer x_1 to x_0, x_2 to x_1,
and so on, and x_0 to x_n. The game displayed below has no Nash equilibrium,
as suggested graphically. The symbol $s \longrightarrow_a s'$ means that agent *a* both prefers
s' to *s* and can convert *s* to *s'*. So, the formula $s \longrightarrow_a s'$ witnesses the agent's
non-happiness.
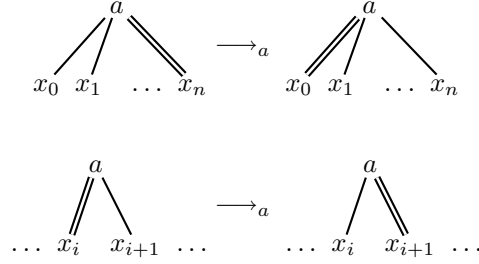
The Formal Way Now, the corresponding formal statement and its proof.

Theorem $Eq\_acyclic : \forall \ OcPref$,
$(\forall \ a, \ rel\_dec \ (OcPref \ a)) \rightarrow$
$(\forall \ g : Game, \ \{s : Strat \mid s2g \ s{=}g \land Eq \ OcPref \ s\}) \rightarrow$
$\forall \ a : Agent, \ irreflexive \ (clos\_trans \ Outcome \ (OcPref \ a))$.

Proof   Assume all premises. In particular, let a be an agent and oc be an outcome related to itself by the transitive closure of the agent's preference. Prove a contradiction by building a game such that every strategy profile for the game can be improved upon, as follows. By lemma $clos\_trans\_path$, get an actual path loc from oc to itself with respect to the preference. If loc is empty then invoke lemma $Conv\_refl$. If loc is not empty then, thanks to the assumption, compute a Nash equilibrium for the game with root owned by agent a and the children being leaves enclosing oc for the first and the elements of loc for the others. Case split on the right-hand substrategy profile list of the Nash equilibrium being empty. If it is empty, and the left-hand substrategy profile list of the Nash equilibrium as well, then the root of the game has two or more children and the strategy profile has one only, hence a contradiction. If the left-hand substrategy profile list is not empty then $ListConv\_refl$, $map\_s2g\_gL\_InducedOutcome$, $path\_app\_elim\_right$, and map_app may be needed. If the right-hand substrategy profile list of the Nash equilibrium is not empty then invoke associativity of list appending as well as the lemmas mentioned just above, and a case splitting on the left-hand substrategy profile list again.   □

9.5   Examples

Partial Order and Subgame Perfect Equilibrium As in section 9.3, consider three outcomes oc_1, oc_2 and oc_3, and an agent a that only prefers oc_2 to oc_3. Each of the three possible linear extensions of a's preference, prior to "backward induction", leads to a subgame perfect equilibrium as shown below, where the symbol $<$ represents the different linear extensions. Compare with the "backward induction" without prior linear extension of subsection 9.3.

$$oc_2 < oc_3 < oc_1 \qquad\qquad oc_3 < oc_2 < oc_1 \qquad\qquad oc_3 < oc_1 < oc_2$$
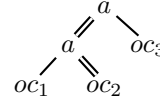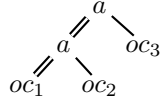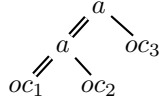
In the same way, the results obtained in this section show that subgame perfect equilibrium exists for games with preferences discussed in section 3.

Benevolent and Malevolent Selfishness In subsection 3.1 were defined benevolent and malevolent selfishnesses. It is not surprising that, sometimes, benevolent selfishness yields better payoffs for all agents than malevolent selfishness. For instance compare the induced payoff functions below. The lefthand "backward induction" corresponds to benevolent selfishness, and the righthand one to malevolent selfishness.
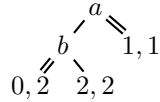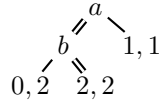
On the contrary, malevolent selfishness may yield better payoffs for all agents than benevolent selfishness, as shown below. The lefthand "backward induction" corresponds to benevolent selfishness, and the righthand one to malevolent selfishness.
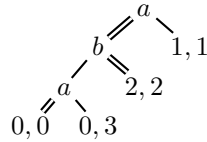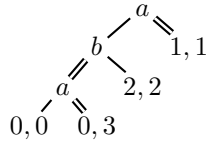
10   Conclusion

This paper introduces a new inductive formalism for sequential games. it also replaces real-valued payoff functions with atomic objects called outcomes, and the usual total order over the reals with arbitrary preferences. This way, it also defines an abstract version of sequential games, with similar tree structure and notion of convertibility as for traditional sequential games. The notions of Nash equilibrium, subgame perfect equilibrium, and "backward induction" are translated into the new formalism. When preferences are totally ordered, "backward induction" guarantees existence of subgame perfect equilibrium for all sequential games, thus translating Kuhn's result in the new formalism. However, an example shows that "backward induction" fails to provide equilibrium for non-totally ordered preferences, *e.g.*, partial orders. But when preferences are acyclic, it is still possible to perform "backward induction" on a game whose preferences have

been linearly extended. This yields a subgame perfect equilibrium of the game with respect to the acyclic preferences, because removing arcs from a preference relation amounts to removing reasons for being unhappy. So, given a collection of outcomes, the following three propositions are equivalent:

- The agents' preferences over the given outcomes are acyclic.
- Every game built over the given outcomes has a Nash equilibrium.
- Every game built over the given outcomes has a subgame perfect equilibrium.

The formalism introduced in this paper is suitable for proofs in Coq, which is a (highly reliable) constructive proof assistant. This way, the above-mentioned equivalence is fully computer-certified using Coq. Beside the additional guarantee of correctness provided by the Coq proof, the activity of formalisation also helps clearly identify the useful definitions and the main articulations of the proof.

Informally, a result due to Aumann states that, when dealing with traditional sequential games, "common knowledge of rationality among agents" is equivalent to "backward induction" (where "rationality" means playing in order to maximise one's payoff). This is arguable in abstract sequential games if one expects "common knowledge of rationality among agents" to imply Nash equilibrium. Indeed, "backward induction" may not imply Nash equilibrium, as seen in this paper. Therefore, "backward induction" may not imply "common knowledge of rationality among agents". Instead, one may wonder whether "common knowledge of rationality among agents" is equivalent to subgame perfect equilibrium, whatever it may mean, when preferences are acyclic, *i.e.* rational in some sense. In this case, the difference between "backward induction" and subgame perfect equilibrium seems critical again.

11   Acknowledgement

References

1. The Coq proof assistant, version 8.1, http://coq.inria.fr/.
2. W. J. Baumol and S. M. Goldfeld. *Precursors in Mathematical Economics: An Anthology*, pages 3–9. London School of Economics and Political Science, 1968. Kuhn, H. W., Preface to Waldegrave's Comments: Excerpt from Montmort's Letter to Nicholas Bernoulli.
3. Yves Berthot and Pierre Castéran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.
4. David Blackwell. An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics*, 6:1–8, 1956.
5. Augustin A. Cournot. *Recherches sur les Principes Mathematiques de la Theorie des Richesses*. Hachette, Paris, 1838.
6. Thomas Hobbes. Leviathan, 1651.

7. Aumann R. J. and M. Maschler. Game theoretic analysis of a bankruptcy problem from the Talmud. *Journal of Economic Theory*, 36:195–213, 1985.

8. Thomas Krieger. On Pareto equilibria in vector-valued extensive form games. *Mathematical Methods of Operations Research*, 58:449–458, 2003.

9. Harold W. Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games II*, 1953.

10. Stéphane Le Roux. Non-determinism and Nash equilibria for sequential game over partial order. In *Proceedings Computational Logic and Applications, CLA '05.* Discrete Mathematics and Theoretical Computer Science Proceedings, 2006.

11. Stéphane Le Roux. Acyclicity and finite linear extendability: a formal and constructive equivalence. Research report RR2007-14, LIP, École normale supérieure de Lyon, 2007. http://www.ens-lyon.fr/LIP/Pub/rr2007.php.

12. Stéphane Le Roux, Pierre Lescanne, and René Vestergaard. A discrete Nash theorem with quadratic complexity and dynamic equilibria. Research report IS-RR-2006-006, JAIST, 2006.

13. Ariel Rubinstein Martin J. Osborne. *A Course in Game Theory*. The MIT Press, 1994.

14. John Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36:48–49, 1950.

15. John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton Univ. Press, Princeton, 1944.

16. Reinhard Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. *Zeitschrift für die desamte Staatswissenschaft*, 121, 1965.

17. Reinhard Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4, 1975.

18. Herbert A. Simon. A behavioral model of rational choice. *The Quarterly Journal of Economics*, 69(1):99–118, 1955.

19. René Vestergaard. A constructive approach to sequential Nash equilibria. *Information Processing Letter*, 97:46–51, 2006.

20. Ernst Zermelo. Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. *Proceedings of the Fifth International Congress of Mathematicians*, 2, 1912.