(Mechanical) Reasoning on Infinite Extensive Games

Pierre Lescanne

(Mechanical) Reasoning on Infinite Extensive Games

Pierre Lescanne
Université de Lyon, ENS de Lyon, CNRS (LIP),
46 allée d'Italie, 69364 Lyon, France

May 20, 2008

Abstract

In order to better understand reasoning involved in analyzing infinite games in extensive form, we performed experiments in proof assistant Coq that are reported here.

1 Introduction

One of the main aim of game theory is to understand how agents reason. Psychologists [14, 4, 17] would say agents are human and tries to answer the question of *how* human agents reason. In this paper, we take a radically different view; for us, agents are ideal abstract entities with unlimited formal deduction power and we attempt to answer the question of *what* a full reasoning can be. For that we decided to analyze the process in full detail on a mechanical device, namely a proof assistant run on a computer which performs at the extreme level of detail all the steps of reasoning. This way, we hope to be able to highlight concepts and deductions that are necessary and that a human would do more or less. Among the possible concepts involved in a reasoning, we claim that some are forgotten whereas irrelevant others are considered. In the first class is temporal reasoning ("always", "eventually"), in the second is the use of "excluded middle" or "double negation". Our experiments have shown that far from easy deductions are used[1], in particular with infinite games. This complexity may explain why human reasoning departs from what is expected.

Among the parts of game theory that have been overlooked, is this of infinite games, on which formal reasoning is rather subtle. In this paper we report research about the concepts underlying infinite games and experiments on the proof assistant Coq [1] to make formal reasoning effective.

2 COQ and the Constructive Logic

Around 1980 a new concept called *Curry-Howard correspondence* emerged. It relies on type theory and lambda-calculus [8] and says basically that *proofs are programs*. In this theory, all objects have a type, that is an annotation that limits its use. For instance, an object f of type $A \to B$, written $f : A \to B$ represents a function and can only be applied to an object of type A to produce an object of type B. In our development we will write, for instance, $a : Agent$ to say that a is an $Agent$ and $s : InfStrategy$ to say that s is an infinite strategy. In what follows, a node (written *iNode*) in a infinite strategy is something that takes an agent, a choice, a finite strategy and an infinite strategy and produces an infinite strategy has a type, in more precise words it has the type,

$$iNode : Agent \to Choice \to FinStrategy \to InfStrategy \to InfStrategy.$$

[1] The reader is invited to have a look at the scripts.

The Curry-Howard correspondence says also that *types are propositions* and insists essentially on the computational content of proofs and establishes the bases of the so-called constructive logic. Indeed since a proof works as a computation, an object can be taken into consideration only if it can be constructed and an existential proof is accepted if it allows constructing the object it claims the existence of. For instance, in an infinite extensive game we assert that there exists a utility for an agent a that can be associated with an infinite strategy, but this works if we prove it exists, i.e., provide a way to construct this utility, since assuming the existence without the construction is not enough.

The following formal development in Coq has been built in this framework and checked on a computer and is attached to proof scripts available on the web site of the author[2]. In this article we try to describe and comment the content of the scripts without entering in their full technicality, but the reader is anyway strongly encourage to have a look at the scripts to convince himself of the materiality of the proofs and moreover, if he has access to a Coq implementation, he should try to run them on a computer.

2.1 Natural deduction

The proof systems we describe is based on natural deduction, which is one of the main system to formalize logic. Natural deduction has been formalized by Gerhard Gentzen [6] in 1935 (see also [11, 15]). Its adjective "natural" comes form the fact that its creators considered that it is just the natural way to formalize logic. It is based on the fact that to conduct a proof, one works under hypotheses and one tries to draw a conclusion. One considers that a theorem has been proved when all the hypotheses have been discharged.

The basic concept of natural deduction is this of *sequent*. A sequent is a pair of a context Γ and a proposition φ, which is written $\Gamma \vdash \varphi$ and which means that φ is a logical consequence of the set of hypotheses Γ. A context is a set of propositions and when we write Γ, φ we mean that the sequent Γ is enriched by the proposition φ, in other terms this is a notation for the union of sets of propositions. In this calculus there is one axiom and several rules. The axiom is

$$\Gamma \vdash \varphi \qquad \text{if } \varphi \in \Gamma$$

There are two kinds of rules for each connector, namely *introduction rule* and *elimination rule*. For instance, for the implication $\to$, the introduction rule is

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}$$

whereas the elimination rule is

$$\frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$

also known as the *modus ponens*. We are now giving only the rule for the universal quantifier, the introduction rule

$$\frac{\Gamma, x : A \vdash P(x)}{\Gamma \vdash \forall x : A, P(x)}$$

and the elimination rule:

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash \forall x : A, P(x)}{P(a)}$$

Through the Curry-Howard correspondence, natural deduction is strongly connected to type theory and computation.

[2] `http://perso.ens-lyon.fr/pierre.lescanne/COQ/INFGAMES/`

2.2 COQ

In this paper we are not describing all the detail of the proof assistant Coq and we advise the reader to read the book [3]. Basically in Coq, proofs are mathematical objects in natural deduction and are considered as first class citizen that can be printed, be exchanged as objects between people, and overall be checked by a proof checker (a specific software) that examines it in full detail. As building proofs is a tedious process, the software Coq offers tools to build them. Basically a Coq development is made of several not fully separated phases: the user defines data structures (in our case: games and strategies), then he defines predicates, relations and functions on those structures (in our case: conversion, utilities and equilibria) and finally he proves theorems about those structures and predicates (in our case: a theorem that says that subgame perfect equilibria are Nash equilibria). Typically a script presents a set of sections in which a subdevelopment is presented, this section may invoke other sections; it is a sequence of declaration of variables, axioms, or hypothesis[3], followed by definitions and theorems with their proofs. To check that a script is correct, it is highly recommended to run it on a Coq implementation.

3 Induction, Coinduction, and Fixed point

Induction is a tool to reason over infinite sets of objects, provided those objects are finitely based. On the opposite, *coinduction* [2] has been designed to reason on infinite objects, like games with infinite paths. However infinite objects may have parts that are finite like finite paths in infinite games and induction can also be used on infinite objects, specifically on their finite subparts. In particular, if the paths are finite, one can compute the utility. Moreover in infinite games we can define finite relations, called *convertibility*. Therefore finite concepts are interleaved with infinite ones.

Beside games, we can define other finite or infinite objects, further we consider the following *inductive* (finite) objects: 1) finite games, 2) finite strategies (or strategy profiles), 3) the predicate *eventually right* that says when applied to a path of a game that the path in question goes eventually to the right, 4) the two convertibility congruences among strategies, namely among finite strategies, but also among infinitely strategy (we indeed claim this latter congruence can be described finitely provided we restrict it to appropriate strategies). 5) Nash equilibrium on finite games, 6) backward induction predicate and 7) Nash equilibrium on infinite games are also finite inductive concepts.

In this paper the archetype of a *coinductive* object is an infinite game. Obviously the associated concept of infinite strategy is coinductive. The function *i2u* that associates a utility with a strategy is also coinductive since the infinite strategies are. The predicate *SGPE* which tells whether an infinite strategy is a *subgame perfect equilibrium* is a coinductive object since the strategy is. In what follows we will define predicates that say that a property is "always" satisfied along a path, those properties on infinite objects are coinductively defined.

Roughly speaking inductive definitions are like equations and correspond to define the concept as *least fixed points* and the properties of this concept are derived of this minimality. On another hand, coinductive definitions correspond to *greatest fixed points* and the properties of this concept are derived from the maximality. Like in the everyday live of a mathematician, handling infinite objects (actual infinity vs potential infinity) is tricky and the Coq user does not escape this rule. Notice that Coq offers tools to verify along a proof that the one one builds will be accepted by the checker.

4 Informal presentation of infinite games

In this paper, we analyze *games in extensive form*. Informally such games are presented as trees. Each node of the tree is a situation in the game where a player has to take a decision. For reason of simplicity, following Vestergaard [16], we studied *binary games* where the players have only two choices. This seems a reasonable design decision, since we can reduce a choice among n to a binary choice between the n^{th} and the

[3] Adding hypotheses and/or axioms is not recommended as it may break the consistence.

set $\{1, ..., n-1\}$ followed by the binary choice between the $(n-1)^{th}$ and the set $\{1, ..., n-2\}$ etc... We do not loose any generality in term of modeling, and in term of abstraction this is the same as n choices at each node. A reader interested by an implementation based on more than two choices, i.e., on polyadic trees, is advised to consult Stéphane Le Roux PhD [12]. By considering binary games, we hope to be more didactic.

First we will consider *finite binary games*. Those games are somewhat connected with games with *finite horizon*, but in addition to offer choices to a finite depth, they are also finitely branching. Such a game is either a leaf, that is a game which ends and attributes the utilities, or a node where an agent has to take a decision. This leads typically to an inductive description. Like a natural is either 0 or the successor of a natural, or a binary tree is either a leaf with a content or a node made of two binary subtrees, a finite game is either a leaf with a function that associates a utility to each agent or a node with an agent and two subgames, namely a left subgame and a right subgame. With a finite game, we associate strategies. A strategy has the same structure as the game it is associated with, except that to each (internal) node we give a direction *left* or *right*, which corresponds to the choice made by the agent in this particular situation. In other words, a strategy is either a leaf associated with a utility, exactly like for games, or a node with an agent, a choice (left or right), and two substrategies.

The main goal of the study presented in this paper is infinite games, more specifically we describe deductions and formal reasoning on those games. Again for reason of simplicity, we study infinite games that looks like "centipedes", in other words binary games with a unique infinite path. The basic concept of those games is a node made of three components: an agent, a left subgames which is itself an infinite subgame, and a right subgame which is a finite subgame. This means that we "recycle" the formal development made for finite subgames for the right branch. Since the game is infinite its formalization relies no more on induction, but on coinduction. Like for finite binary games, the attached concept of strategy relies on a node with four components: an agent, a choice (left or right), an infinite left substrategy, and a finite right strategy.

5 Equilibria on finite Games

As an introduction to the Coq development, let us study equilibria on *finite games*.

5.1 Agent and Utility

First we set two basic concepts namely *Agent* and *Utility*. In Coq, this is done by the following declaration:

Variable *Agent* : *Set*.
Variable *Utility*: *Set*.

In addition we set a *preference* as a binary relation on *Utility* and we make this *preference* a preorder, which we write $=<$, *Utility_fun* is the type of the *utility functions*, in other words the type of the objects belonging to $Agent \rightarrow Utility$.

5.2 Finite Games

A finite binary game, which we call a *FinGame*, in Coq, is built by induction, this means that this is either a *leaf* or this is a game with two subgames that are themselves binary games, we encapsulate these three items under the label *gNode*. In Coq, such a data structure will be written:

Inductive *FinGame* : *Set* :=
— *gLeaf* : *Utility_fun* $\rightarrow$ *FinGame*
— *gNode* : *Agent* $\rightarrow$ *FinGame* $\rightarrow$ *FinGame* $\rightarrow$ *FinGame*.

where Inductive is the key word to introduce any inductive definition. *FinGame* is a data structure, but elsewhere the same Inductive keyword will be used to define a predicate. An inductive definition creates a

deduction rule that allows us to reason by induction, namely in the case of *FinGame*, the term *FinGame_ind* is created by Coq:

$\forall\ P : FinGame \to Prop,$
$(\forall\ \mathrm{u} :\ Utility\_fun,\ P\ (gLeaf\ u)) \to$
$(\forall\ (a :\ Agent)\ (f0 :FinGame),\ P\ f0 \to \forall\ f1 :\ FinGame,\ P\ f1 \to P\ (gNode\ \mathrm{a}\ f0\ f1)\ )$
$\to \forall\ f1 :\ FinGame,\ P\ f1$

It can be written as the rule

$$\frac{\forall u : Utility\_fun, P(gLeaf\ u) \qquad [\forall f0 : FinGame, P\ f0 \wedge \forall f1 : FinGame, P\ f1] \to \forall a : Agent, P\ (gNode\ a\ f0\ f1)}{\forall f : FinGame, P\ f}$$

in other words, to prove that some properties holds for all the finite binary games, one has to prove it to hold for leaves and to prove that if it holds for two games then it holds for the game obtained by pairing those two games under the "control" of an agent.

5.3 Finite Strategies

As we said, a finite strategy[4] has a structure which is very similar to a finite game, the only difference is that we add a "choice" at each node.

Inductive *FinStrategy* : *Set* :=
— *sLeaf* : *Utility_fun* → *FinStrategy*
— *sNode* : *Agent* → *Choice* → *FinStrategy* → *FinStrategy* → *FinStrategy*.

With a strategy we provide a function that associates with every strategy a utility function. The Coq keyword to define such a function is Fixpoint. A syntactic construction called *match* allows using a pattern matching mechanism.

Fixpoint *f2u* (*s*:*FinStrategy*) : *Utility_fun* :=
match s with
— (*sLeaf uf*) ⇒ *uf*
— (*sNode a left sl sr*) ⇒ (*f2u sl*)
— (*sNode a right sl sr*) ⇒ (*f2u sr*)
end.

This reads as

- if the strategy *s* is a leaf, i.e., *sLeaf uf* (where *uf* is a utility function), then one returns *uf*,
- otherwise one returns the utility function associated with the left substrategy, if the choice is *left*, or the right substrategy if the choice is *right*.

Now we define on finite strategies a relation, which we call *a-convertibility*, and which we write *s<<− a−>>s'*. Labeled with *a*, it is associated with the agent *a* and says that

- Two leaves associated with the same utility function are *a*-convertible,
- Two *sNode*s, (*sNode a c s1 s2*) and (*sNode a c' s1' s2'*), associated with the same agent *a* are *a*-convertible if *s1 <<− a−>>s1'* and *s2 <<− a−>>s2'*. Notice that *c* and *c'* do not have to be the same,
- Two *sNode*s, (*sNode a' c s1 s2*) and (*sNode a' c s1' s2'*), associated with another agent *a'* are *a*-convertible if *s1 <<− a−>>s1'* and *s2 <<− a−>>s2'*. Notice that in this case *c* has to be the same.

[4] Actually we should probably say a *strategy profile*, but, for convenience and conciseness, we call this data structure just a *strategy*.

We prove in Coq (a first interesting exercise) that the a-convertibility is an equivalence relation i.e., it is reflexive, symmetric and transitive. We are now equipped to define the predicate *Nash equilibrium* on *finite strategy* (in Coq the predicate *FinNashEq* on *FinStrategy*):

```
Inductive FinNashEq: FinStrategy → Prop :=
— NE : ∀ (s:FinStrategy),
  (∀ (a:Agent) (s':FinStrategy), s<<- a->>s' → (f2u s' a =< f2u s a)) →
  FinNashEq s.
```

It says that s is a Nash equilibrium if for all strategy s' that is a-convertible to s,

$$f2u\ s'\ a =< f2u\ s\ a,$$

in other words, the utility for a computed for s' is less than the utility for a computed for s. This is nothing more than the traditional definition of Nash equilibrium for extensive game, written in the formalism of Coq.

Beside the predicate *Nash equilibrium*, we define the predicate *BI* which says whether the strategy s can be obtained by the so-called *backward induction*. In Coq, it is written:

```
Inductive BI: FinStrategy → Prop :=
— BILeaf: ∀ uf:Utility_fun, BI (sLeaf uf)
— BINode_left: ∀ (a:Agent) (sl sr: FinStrategy),
    BI sl → BI sr → (f2u sr a =< f2u sl a) →
    BI (sNode a left sl sr)
— BINode_right: ∀ (a:Agent) (sl sr: FinStrategy),
    BI sl → BI sr → (f2u sl a =< f2u sr a) →
    BI (sNode a right sl sr).
```

In other words,

- *BI* holds for leaves,
- if *BI* holds for *sl* and *sr* and *f2u sl a =< f2u sr a* then *BI* holds for *sNode a left sl sr* (*BINode_left* principle),
- the *BINode_right* principle is symmetric, it says, that if *BI* holds for *sl* and for *sr* and *f2u sr a =< f2u sl a* then is *BI* holds for *sNode a right sl sr*.

This is the formalization of backward induction for finite horizon games as described in textbooks [9, 10, 7]. Then we are able to prove in Coq the theorem:

```
Theorem BI_is_FinNashEq : ∀ s, BI s → FinNashEq s.
```

The theorem relies on the following fact: the inductive definition of *BI* is a somewhat equational definition that says that *BI* is the least fixed point of that equation. Therefore if we can prove that if *FinNashEq* is another fixed point, this other fixed point is implied by *BI* and we are set. This can be done by three lemmas.

- *FinNashEq* satisfies the statement given by *BILeaf*:

```
Lemma FinNashEq Leaf : ∀ (uf:Utility_fun), FinNashEq (sLeaf uf).
```

- *FinNashEq* satisfies the statement given by *BINode_left*:

```
Lemma FinNashEq_fixpt_left : ∀ (a:Agent) (s1 s2: FinStrategy),
  FinNashEq s1 → FinNashEq s2 → (f2u s2 a =< f2u s1 a) →
  FinNashEq (sNode a left s1 s2).
```

- *FinNashEq* satisfies the statement given by *BINode_right*:

Lemma *FinNashEq_fixpt_right* : ∀ (*a*:*Agent*) (*s1 s2*: *FinStrategy*),
FinNashEq s1 → *FinNashEq s2* → (*f2u s1 a* =< *f2u s2 a*) →
FinNashEq (*sNode a right s1 s2*).

Then we conclude than *FinNashEq* satisfies the same equation as *BI*. Since *BI* is the least fixed point, *BI s* implies *FinNashEq s*.

6 Coinduction

Coinduction is the partner of induction, but whereas the induction defines objects or predicates as the least fixed point of some equations and therefore specifies finite objects and presents them from basic objects, coinduction defines infinite objects or infinite and finite objects as greatest fixed point of some equation. If one has in mind to define infinite objects only, there is no need to specify basic objects (i.e., objects meant to be the basis on which to define finite object). The typical infinite object is

CoInductive *InfGame* : *Set* :=
— *igNode* : *Agent* → *InfGame* → *FinGame* → *InfGame*.

In other words, an infinite binary game is made of an agent, an infinite binary subgame, and a finite binary subgame. On the same model as for finite binary games, we define infinite strategies the same way, i.e., as a coinductive:

CoInductive *InfStrategy* : *Set* :=
— *iNode* : *Agent* → *Choice* → *InfStrategy* → *FinStrategy* → *InfStrategy*.

7 Equilibria on Infinite Games

7.1 Decomposing Infinite Games

Since the strategies are infinite we can define total functions *iAgent* (which gives the agent at the root of the game), *SubGameLeft* (the infinite game on the left) and *SubGameRight* (the finite game on the right) with the lemma:

Lemma *InfGame_Decomposition*: ∀ (*g*:*InfGame*),
igNode (*iAgent g*) (*SubGameLeft g*) (*SubGameRight g*) = *g*.

that says that a game can be uniquely decomposed into an agent, a left subgame and a right subgame.

7.2 Generalizing the notion of horizon

In classical extensive game theory, the concept of backward induction relies on this of *finite horizon*, our experience in mechanizing the proof has shown us that finite horizon is not exactly the right notion, we prefer the notion of *limited horizon*, which means that the horizon of the agents is not bounded by a number, but that however it cannot go to infinity. In the frame of binary games, this means that we forbid paths that go always to the left, we want to consider the paths that eventually go to the right so that one can compute the utility of each agent.

Since the games are now infinite, a *total function* that associates, by computation, a utility to a strategy can no more be defined, one can only define a *relation* between a strategy and a utility. If the path of the choice goes always to the left (on the backbone) the utility cannot be defined, but if the path goes *"eventually to the right"*, the utility of a strategy makes sense. We therefore define a predicate *EvtRight* which says whether the path of the given strategy goes eventually to the right. The basic case for this predicate is that it holds for the game *iNode a right sl sr*. The induction case says that if *EvtRight sl* then *EvtRight* (*iNode a left sl sr*). The predicate *EvtRight* is typical of an inductive predicate. We get two lemmas which show the existence and the uniqueness of a utility for a strategy that goes eventually to the right, therefore in

that case, the association of the utility to the strategy is functional. Since most of the time we want this functional association, we require the predicate "eventually right" to hold.

Lemma *Existence_i2u*: ∀ (*a*:*Agent*) (*s*:*InfStrategy*),
EvtRight s → ∃ *u*:*Utility*, *i2u a u s*.

Lemma *Uniqueness_i2u*: ∀ (*a*:*Agent*) (*u v*:*Utility*) (*s*:*InfStrategy*),
EvtRight s → *i2u a u s* → *i2u a v s* → *u*=*v*.

Going eventually to the right is not enough. In some cases, when dealing with subgames, more specifically with subgame perfect equilibria, one wants to be able to compute utilities in subgames. Therefore one wants to be sure that even further in subgames one will go eventually to the right. Hence we define another predicate that ensures that we will always go eventually to the right. This predicate *AlwEvtRight* reminds us the same kind of predicate defined in the frame of temporal logic (see [3] chap.13 and [5]). Since *AlwEvtRight* has to traverse the whole infinite game, it has to be coinductive. It says that *AlwEvtRight* (*iNode a c sl sr*) holds, if *AlwEvtRight sl* and *EvtRight sr* hold, in other words, a strategy goes always eventually to the right if its left substrategy goes always eventually to the right and if its right substrategy (which is finite) goes eventually to the right.

7.3 Convertibility of infinite strategies

The definition of the convertibility *s* < −−*a* −−>*s'* of two strategies *s* and *s'* is inductive. Its base case is the reflexivity of the convertibility. In other words, *s* < −−*a* −−>*s'* if:

- *s* = *s'*, or
- if *s1* < −−*a* −−>*s1'* and *s2*<<− *a*−>>*s2'* and *s* is *iNode a c s1 s2* and *s'* is *iNode a c' s1' s2'* (*c* may be different, but *a* has to be the same), or
- if *s1* < −−*a* −−>*s1'* and *s2*<<− *a*−>>*s2'* and *s* is *iNode a' c s1 s2* and *s'* is *iNode a' c s1' s2'* (different agents, but same *c*).

7.4 Equilibria

To define Nash Equilibria on infinite games, one must be able to compare utilities. For that, one must be able to compute those utilities. Hence one restricts to strategies that go eventually to the right.

Inductive *InfNashEq*: *InfStrategy* → *Prop* :=
— *INE* : ∀ (*s*: *InfStrategy*),
EvtRight s →
(∀ (*a*:*Agent*) (*s'*:*InfStrategy*) (*u u'*: *Utility*),
EvtRight s' → *s'*< −−*a* −−>*s* → (*i2u a u' s'*) → (*i2u a u s*) → (*u'* =< *u*)) →
InfNashEq s.

On infinite strategies one defines a predicate *Subgame Perfect Equilibrium*. Because one has to be able to compute utilities on subgames, the Subgame Perfect Equilibrium predicate, written *SGPE*, is defined on strategies which go always eventually to the right. Notice that Subgame Perfect Equilibria are defined on a smaller class of strategies than Nash Equilibria, but also that the definition of *SGPE* is coinductive, since its definition requires to traverse the whole infinite subgame.

CoInductive *SGPE*: *InfStrategy* → *Prop* :=
— *SGPEnode_left*: ∀ (*a*:*Agent*)(*u*:*Utility*) (*sl*: *InfStrategy*) (*sr*: *FinStrategy*),
AlwEvtRight sl → *SGPE sl* → *BI sr* → *i2u a u sl* → (*f2u sr a* =< *u*) →
SGPE (*iNode a left sl sr*)
— *SGPEnode_right*: ∀ (*a*:*Agent*) (*u*:*Utility*) (*sl*: *InfStrategy*) (*sr*: *FinStrategy*),

AlwEvtRight sl → SGPE sl → BI sr → i2u a u sl → (u =< f2u sr a) →
SGPE (iNode a right sl sr).

One can show that *InfNashEq* is a fixed point of this definition. To prove the lemma *SGPE s → InfNashEq s*, one needs to perform a proof by induction, this cannot be an induction on the definition *SGPE* which is coinductive, therefore we impose an additional requirement, namely that somewhere in the game a maximal utility is reached for all agents. This means that there are substrategies, agents cannot consider in increasing their utility. We give a statement of this requirement through the following predicate *EvtMaxU*:

Inductive *EvtMaxU*: *InfStrategy → Prop* :=
— *EUM_basis*: ∀ (*s*:*InfStrategy*),
(∀ (*a*:*Agent*)(*u u'*:*Utility*)(*s'*:*InfStrategy*), *s*< −−*a* −−>*s'* → *i2u a u' s'* → *i2u a u s* → (*u'* =< *u*)) →
EvtMaxU s
— *EUM_gen*: ∀ (*a*:*Agent*)(*c*:*Choice*)(*sl*:*InfStrategy*)(*sr*:*FinStrategy*),
EvtMaxU sl → *EvtMaxU* (*iNode a c sl sr*).

It has two parts.

- The game fulfills the maximal utility requirement for every agent,
- One of its subgames does.

Then we can set and prove the main theorem:

Theorem *SGPE_is_InfNashEq* : ∀ *s*:*InfStrategy*, *EvtMaxU s* → *SGPE s* → *InfNashEq s*.

If every agent reaches a maximal utility somewhere in the game, then a subgame perfect equilibrium is a Nash equilibrium.

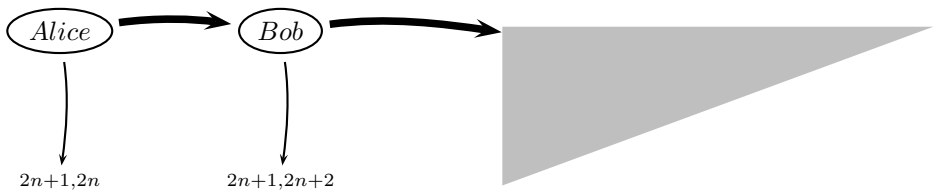
8 The "Illogic of Conflict Escalation" revisited

Now we may want to apply those general results on a specific infinite subgame. Consider the following game proposed by Shubik [13]. Recall its principle. Two agents *Alice* and *Bob* compete in an auction for an object of a value, say 2 ¢, in this statement. The two agents bid 2 ¢, one after the other. If one agent gives up, the highest bidder gets the object, but the second bidder pays also for his bid. As Shubik noted this game may never cease.

Let us take as the utility ordering, the order $\geq$ on the *nat* (the natural numbers or non negative integers), in other words, if u and v are two utilities, u and v are of type *nat* and $u =< v$ means $u \geq v$, i.e., the larger the bid, the smaller the utility.

8.1 Strategy *Never give up*

Let us consider the function *enlarge left left n* on strategies, that takes a strategy and returns another one and that can be described by the following picture, where the thick arrows give the choice at each node.



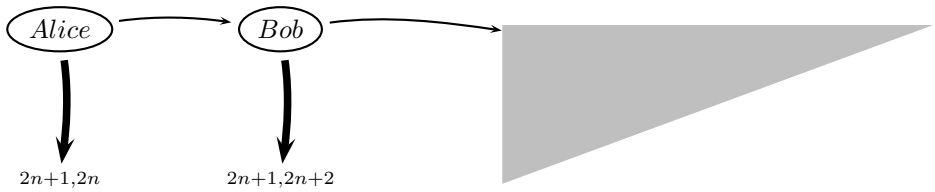
We define the strategy *never give up* (in short *ngu*) as the strategy where the agents always keep bidding. It requires to define first a strategy that starts with the value n as the solution of

ngu n = enlarge left left (ngu (S n)).

ngu n is an infinite strategy and we can prove that for *n*, *(ngu n)* is not a Nash equilibrium. In this proof, reasoning has a flavor of *temporal logic*.

8.2 Strategy *Always give up*

Here we consider the function *enlarge right right n* on strategies.



Again *agu n* is an infinite strategy and we can prove the lemma:

Lemma *SGPEAGU*: ∀ (*n*:*nat*), *SGPE Ag12 nat ge* (*agu n*).

which says that that this strategy is a *Subgame Perfect Equilibrium.*

9 Conclusion

The experiment presented in this paper is in a very stage. Its goal is to make clear that, in games, especially in infinite games, the reasoning can be mechanized despite it is not obvious; it evidences with no surprise some subtlety. To mimic it, humans elaborate rather complex deductions. One of the main statement we can make is that true *classical logic is never used*. More specifically, we noticed no use of the excluded middle or proofs by double negations. We hope this experience opens a discussion and shows how far humans are from the somewhat ideal mechanized reasoning. We wish to pursue this research by trying other modelings and concepts and going further in the proofs, for instance finding other conditions for *SGPE s* → *InfNashEq s* and deeper results on the example of "Illogic Conflict of Escalation".

References

[1] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Yann Coscoy, David Delahaye, Daniel de Rauglaudre, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, Gérard Huet, Henri Laulhère, César Muñoz, Chetan Murthy, Catherine Parent-Vigouroux, Patrick Loiseleur, Christine Paulin-Mohring, Amokrane Saïbi, and Benjamin Werner. *The Coq Proof Assistant Reference Manual*. INRIA, version 6.3.11 edition, May 2000.

[2] Y. Bertot. Coinduction in Coq. Open archives, March 2006. http://cel.archives-ouvertes.fr/inria-00001174/fr/.

[3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Springer-Verlag, 2004.

[4] A.M. Colman. Depth of strategic reasoning in games. *Trends Cogn. Sci.*, 7:1–2, 2003.

[5] Solange Coupet-Grimal. An axiomatization of linear temporal logic in the calculus of inductive constructions. *J Logic Computation*, 13(6):801–813, 2003.

[6] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[7] Herbert Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*. Princeton University Press, 2000.

[8] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoret Computer Science*. Cambridge University Press, 1989.

[9] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, Massachusetts, 1994.

[10] Martin J. Osborne. *An Introduction to Game Theory*. Oxford, 2004.

[11] Dag Prawitz. *Natural Deduction: A Proof Theoretical Study*. Almqvist and Wiksell, Stockholm, 1965.

[12] Stéphane Le Roux. *Abstraction and Formalization in Game Theory*. PhD thesis, École normale supérieure de Lyon (France), January 2008.

[13] Martin Shubik. The dollar auction game: A paradox in noncooperative behavior and escalation. *Journal of Conflict Resolution*, 15(1):109–111, 1971.

[14] D. Stahl and P Wilson. On players models of other players: theory and experimental evidence. *Games and Economic Behavior*, 10:218–254, 1995.

[15] D. van Dalen. *Logic and Structure*. Springer, Berlin, Heidelberg, 3 edition, 1997.

[16] René Vestergaard. A constructive approcah to sequential Nash equilibria. *Inf. Process. Lett.*, 97:46–51, 2006.

[17] Jun Zhang and Trey Hedden. Two paradigms for depth of strategic reasoning in games response to Colman. *Trends Cogn. Sci.*, 7(1):4–5, January 2003.