Coinduction in proof assistants

Pierre Lescanne

ENS de Lyon

6 June 2009

Infinite objects

- An infinite list has infinite many items:
  - $0, 1, 2, ..., n, n + 1, \ldots$

Infinite objects

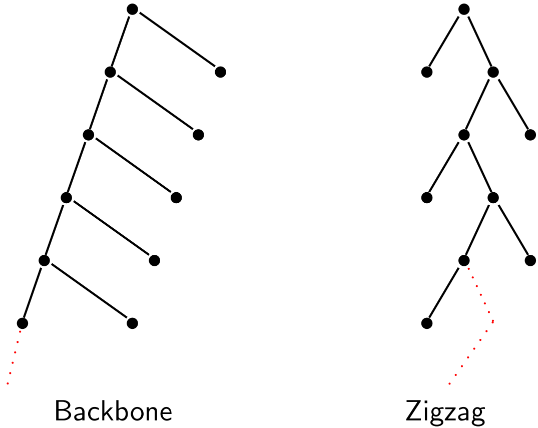
- An infinite list has infinite many items:
  - $0, 1, 2, ..., n, n + 1, \ldots$
  - $0, 0, 0, 1, ..., 1, 1, \ldots,$

Infinite objects

- An infinite list has infinite many items:
  - $0, 1, 2, ..., n, n+1, \ldots$
  - $0, 0, 0, 1, ..., 1, 1, \ldots,$ 1 eventually always

Infinite objects

- An infinite list has infinite many items:
  - $0, 1, 2, ..., n, n+1, \ldots$
  - $0, 0, 0, 1, ..., 1, 1, \ldots,$ 1 eventually always
- An infinite binary tree may have infinite branches:



What is coinduction?

CoInduction is a way to reason on an infinite set
which contains infinite objects.

- The *streams* or infinite lists

What is coinduction?

CoInduction is a way to reason on an infinite set which contains infinite objects.

- The *streams* or infinite lists
  - $[0, 0, 0, \ldots]$,
  - ...,

infinite lists

What is coinduction?

CoInduction is a way to reason on an infinite set
which contains infinite objects.

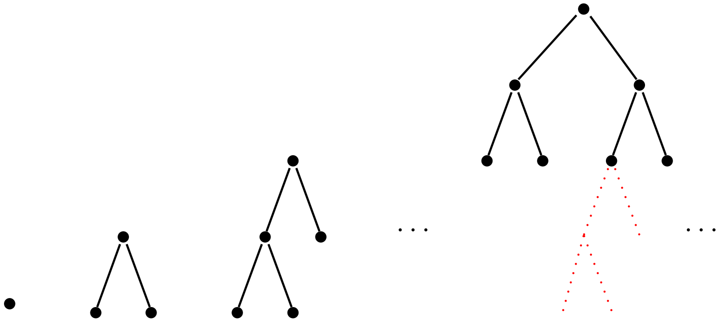
- The *streams* or infinite lists
  - $[0, 0, 0, \ldots]$,
  - ...,
  - $[0, 1, 2, \ldots]$,
  - ...,
  - $[2, 3, 5, 7, 11, \ldots]$,
  - ...,

infinite lists

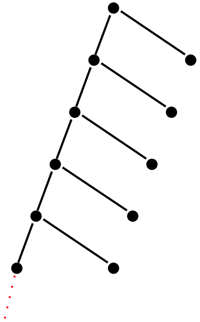
- The lazy lists or finite and infinite lists
  - [ ],
  - [0],
  - [1],
  - ...,
  - [0; 0],
  - [0; 1], ...,
  - [0; 0; 0],
  - [0; 0; 1],
  - ...

- The lazy lists or finite and infinite lists
  - $[\ ]$,
  - $[0]$,
  - $[1]$,
  - ...,
  - $[0; 0]$,
  - $[0; 1]$, ...,
  - $[0; 0; 0]$,
  - $[0; 0; 1]$,
  - ...
  - $[0, 0, 0, \ldots]$, infinite lists
  - ...,
  - $[0, 1, 2, \ldots]$,
  - ...,

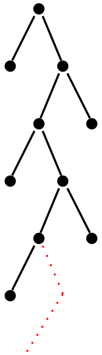
- The finite and infinite binary trees, aka lazy trees



. . .



. . .



. . .

Streams

A stream (or an infinite list) on A is

- of the form $a :: s$, where a is an element of A and s is a stream.

Lazy lists

A **coinductive** list (or a lazy list) on A is

- either the empty lazy list $[\,]$,
- or a lazy list of the form $a :: \ell$,
  where a is an element of A and ℓ is a lazy list.

Lazy binary trees

An **coinductive** binary tree (or lazy binary tree) is

- either the empty lazy binary tree

Lazy binary trees

An **coinductive** binary tree (or lazy binary tree) is

- either the empty lazy binary tree
- or a lazy binary tree made of two lazy binary trees.

CoInduction in COQ

```
CoInductive Stream (A:Set) :  Set :=
| SCons:  A -> Stream A -> Stream A.
```

CoInduction in COQ

```
CoInductive Stream (A:Set) :  Set :=
| SCons:  A -> Stream A -> Stream A.

CoInductive LList (A:Set) :  Set :=
| LNil:  LList A
| LCons:  A -> LList A -> LList A.
```

CoInduction in COQ

```
CoInductive Stream (A:Set) :  Set :=
| SCons:  A -> Stream A -> Stream A.

CoInductive LList (A:Set) :  Set :=
| LNil:  LList A
| LCons:  A -> LList A -> LList A.

CoInductive LBintree :  Set :=
| LLeaf:  LBintree
| LNode:  LBintree -> LBintree -> LBintree.
```

CoInduction in COQ

```
CoInductive Stream (A:Set) :  Set :=
| SCons:  A -> Stream A -> Stream A.

CoInductive LList (A:Set) :  Set :=
| LNil:  LList A
| LCons:  A -> LList A -> LList A.

CoInductive LBintree :  Set :=
| LLeaf:  LBintree
| LNode:  LBintree -> LBintree -> LBintree.
```

Coinductive predicates

One can also define coinductive predicates as predicates on infinite objects.

Coinductive predicates (Predicate *InfiniteBT*)

One can also define coinductive predicates as predicates on infinite objects.

```
CoInductive InfiniteLBT: LBintree -> Prop :=
| IBTLeft :  ∀ bl br,
        InfiniteLBT bl -> InfiniteLBT (LNode bl br)
| IBTRight :  ∀ bl br,
        InfiniteLBT br -> InfiniteLBT (LNode bl br).
```

Defining an infinite object

One defines infinite objects as fixpoints.

Defining an infinite object

One defines infinite objects as fixpoints.

```
CoFixpoint LBackBone:  LBintree := LNode LBackBone LLeaf.
```

Defining an infinite object

One defines infinite objects as fixpoints.

```
CoFixpoint LBackBone:  LBintree := LNode LBackBone LLeaf.
```

```
CoFixpoint Zig:  LBintree := LNode Zag LLeaf
with Zag:  LBintree := LNode LLeaf Zig.
```

Defining an infinite object

One defines infinite objects as fixpoints.

```
CoFixpoint LBackBone:  LBintree := LNode LBackBone LLeaf.
```

```
CoFixpoint Zig:  LBintree := LNode Zag LLeaf
with Zag:  LBintree := LNode LLeaf Zig.
```

One can prove three lemmas:

```
Lemma InfiniteBackbone:  InfiniteLBT LBackBone.
```

```
Lemma InfiniteZig:  InfiniteLBT Zig.
```

```
Lemma InfiniteZag:  InfiniteLBT Zag.
```

How does induction work?

An inductive definition yields the least fixpoint that satisfies the definition.
Inductive reasoning is a way to prove facts based on fixpoint properties.

How does induction work?

An inductive definition yields the least fixpoint that satisfies the definition.
Inductive reasoning is a way to prove facts based on fixpoint properties.

Usually to prove a property by induction,

How does induction work?

An inductive definition yields the least fixpoint that satisfies the definition. Inductive reasoning is a way to prove facts based on fixpoint properties.

Usually to prove a property by induction,

1. One proves the properties on basic objects.

How does induction work?

An inductive definition yields the least fixpoint that satisfies the definition.
Inductive reasoning is a way to prove facts based on fixpoint properties.

Usually to prove a property by induction,

1. One proves the properties on basic objects.
2. One proves the preservation of the property, i. e.,

   If the property is true for the direct sub-objects of an object
   it is true for the whole object.

Induction on naturals

For the naturals this gives:

1. One proves the properties for `0`.

Induction on naturals

For the naturals this gives:

1. One proves the properties for `0`.
2. One proves that if the property is true for the list `n` then it is true for the list `S n`.

Induction on lists

For the lists this gives:

1. One proves the properties for `Nil`

Induction on lists

For the lists this gives:

1. One proves the properties for `Nil`
2. One proves that if the property is true for the list `l` then it is true for the list `Cons a l`.

Induction on binary trees

For the binary trees this gives:

1. One proves the properties for `LLeaf`

Induction on binary trees

For the binary trees this gives:

1. One proves the properties for `LLeaf`
2. One proves that if the property is true for the binary `b1` and for the binary tree `b2` then it is true for the list `LNode b1 b2`.

How does coinduction work?

Coinduction is about fixpoint, but coinduction defines the greatest fixpoint.

How does coinduction work?

Coinduction is about fixpoint, but coinduction defines the greatest fixpoint.

For instance

$\mathcal{L}_0$	The family of finite lists
$\mathcal{L}_1$	The family of finite lists or of lists which ends with an infinite sequence of 0's
$\mathcal{L}_2$	The family of finite lists or infinite lists which contains infinitely many 0's
$\mathcal{L}_\infty$	The family of lazy lists

How does coinduction work?

Coinduction is about fixpoint, but coinduction defines the greatest fixpoint.

For instance

$\mathcal{L}_0$	The family of finite lists
$\mathcal{L}_1$	The family of finite lists or of lists which ends with an infinite sequence of 0's
$\mathcal{L}_2$	The family of finite lists or infinite lists which contains infinitely many 0's
$\mathcal{L}_\infty$	The family of lazy lists

$\mathcal{L}_0 \subset \mathcal{L}_1 \subset \mathcal{L}_2 \subset \mathcal{L}_\infty$.

$A :: \mathcal{L} = \{\ell \in \mathcal{L}_\infty \mid \exists \ell' \in \mathcal{L},\ \exists n \in A, \ell = n :: \ell'\}.$

$\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_\infty$ are solutions of the fixpoint equation :

$$\mathcal{L} = \{[\,]\} \ \cup\ \mathbb{N} :: \mathcal{L}.$$

$$\begin{aligned}
\mathcal{L}_0 &= \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_0 \\
\mathcal{L}_1 &= \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_1 \\
\mathcal{L}_2 &= \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_2 \\
\mathcal{L}_\infty &= \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_\infty
\end{aligned}$$

$A :: \mathcal{L} = \{\ell \in \mathcal{L}_\infty \mid \exists \ell' \in \mathcal{L},\ \exists n \in A, \ell = n :: \ell'\}$.

$\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_\infty$ are solutions of the fixpoint equation :

$$\mathcal{L} = \{[\,]\} \ \cup \ \mathbb{N} :: \mathcal{L}.$$

$$\begin{array}{rcl} \mathcal{L}_0 & = & \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_0 \\ \mathcal{L}_1 & = & \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_1 \\ \mathcal{L}_2 & = & \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_2 \\ \mathcal{L}_\infty & = & \{[\,]\} \cup \mathbb{N} :: \mathcal{L}_\infty \end{array}$$

$\mathcal{L}_0$ is the least fixpoint

$\mathcal{L}_\infty$ is the greatest fixpoint.

The coinduction principle

Sketch: Assume one considers a coinductive object.

One proves that

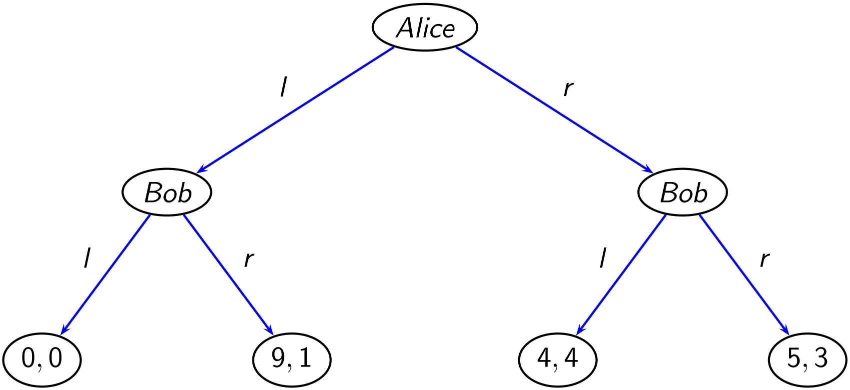
if a property which holds for a sub-object implies

that the property holds for the whole object,

then the property holds for the whole object.

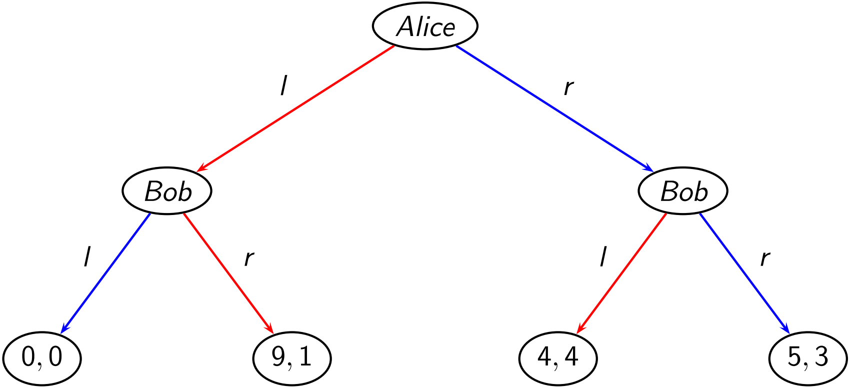
What is a sequential game?

A sequential game is described by a labeled tree



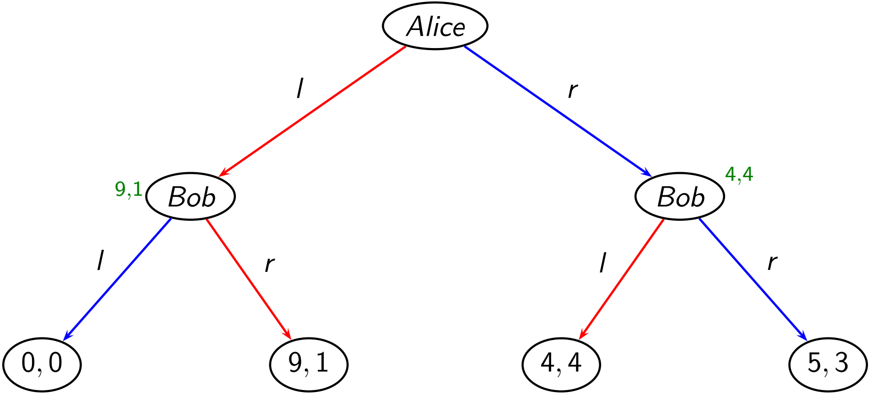
What is a sequential game?

A Nash equilibrium is a situation where if an agent changes alone his action he will get a utility which is not better.



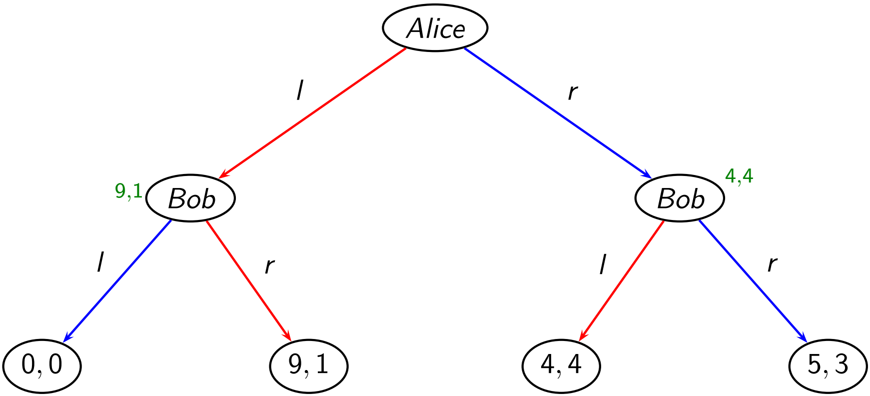
What is a sequential game?

A Nash equilibrium is a situation where if an agent changes alone his action he will get a utility which is not better.



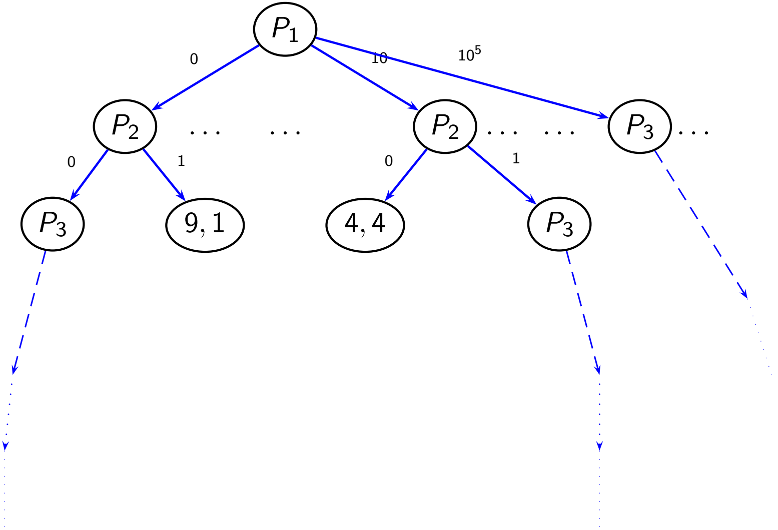
What is a sequential game?

A Nash equilibrium is a situation where if an agent changes alone his action he will get a utility which is not better.



This method for computing a Nash equilibrium is called backward induction.

A sequential game can be infinite



What is an **infinite** sequential game?

What does "infinite" mean?

- The length of the game?
- The number of players?
- The number of choices of actions a player can perform at each decision node?

What is an **infinite** sequential game?

What does "infinite" mean?

- The length of the game?
- The number of players?
- The number of choices of actions a player can perform at each decision node?

Definition [of extensive game] allows terminal histories to be infinitely long. Thus we can use the model of an extensive game to study situations in which the participants do not consider any particular fixed horizon when making decisions. If the length of the longest terminal history is in fact finite, we say that the game has a **finite horizon**.

Even a game with a finite horizon may have infinitely many terminal histories, because some player has infinitely many actions after some history. If a game has a finite horizon and finitely many terminal histories we say it is **finite**.

Martin Osborne,
An Introduction to Game Theory
Oxford U. Press, (2004), p. 137

Definition [of extensive game] allows terminal histories to be infinitely long. Thus we can use the model of an extensive game to study situations in which the participants do not consider any particular fixed horizon when making decisions. If the length of the longest terminal history is in fact finite, we say that the game has a ***finite horizon****.*

Even a game with a finite horizon may have infinitely many terminal histories, because some player has infinitely many actions after some history. If a game has a finite horizon and finitely many terminal histories we say it is ***finite****.*

Martin Osborne,
An Introduction to Game Theory
Oxford U. Press, (2004), p. 137

A typical example: the illogic escalation

In 1971, Martin Shubik described an infinite game, he calls

The Dollar Auction Game,

in which players bid forever.

Finite sequential games as inductive objects

A finite sequential games is described by induction from its subgames.

Without loss of generality, I restrict to binary sequential games.

A *binary finite sequential game* is

- either a **node**, assigned to a **player**, with **two subgames**,
- or a **leaf**.

Finite sequential games as inductive objects

A finite sequential games is described by induction from its subgames.

Without loss of generality, I restrict to binary sequential games.

A *binary finite sequential game* is

- either a **node**, assigned to a **player**, with **two subgames**,
- or a **leaf**.

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

Utility and utility functions

Utility is given.

Utility_fun is a function which associates a utility with an agent:

Definition $\mathit{Utility\_fun} := \mathit{Agent} \rightarrow \mathit{Utility}.$

A “finite” strategy is also an inductive

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

From finite strategy to utility function

```
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.
```

a-convertibility

$s \vdash^{a} \dashv s'$ is a relation between strategies.

s is a-convertible to s'

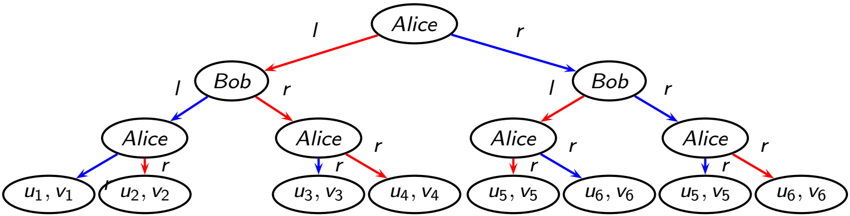
if one goes from s to s' by changing the choices of agent a.

a-convertibility

$s \vdash^{a} \dashv s'$ is a relation between strategies.

s is *a*-convertible to s'

if one goes from s to s' by changing the choices of agent a.

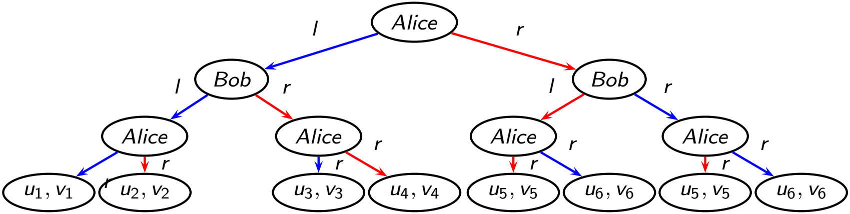


a-convertibility

$s \vdash^{a} \dashv s'$ is a relation between strategies.

s is a-convertible to s'

if one goes from s to s' by changing the choices of agent a.



a-convertibility

$s \vdash a \dashv s'$ is a relation between strategies.

s is a-convertible to s'
if one goes from s to s' by changing the choices of agent a.

$\vdash a \dashv$ is defined as an inductive.

- $sLeaf\ uf \vdash a \dashv sLeaf\ uf$.
- $(sNode\ a\ c\ s1\ s2) \vdash a \dashv (sNode\ a\ c'\ s1'\ s2')$
  if $s1 \vdash a \dashv s1'$ and $s2 \vdash a \dashv s2'$.
  a is the same
  c and c' do not have to be the same,
- $(sNode\ a'\ c\ s1\ s2) \vdash a \dashv (sNode\ a'\ c\ s1'\ s2')$
  if $s1 \vdash a \dashv s1'$ and $s2 \vdash a \dashv s2'$.
  a and a' do not both have to be the same,
  c has to be the same.

a-convertibility

I proved in COQ that the $\vdash a \dashv$ is an equivalence relation.

a-convertibility

I proved in Coq that the $\vdash a \dashv$ is an equivalence relation.

We are now equipped to define the predicate Nash equilibrium on finite strategies.

Nash equilibrium

Definition *FinNashEq* (s:FinStrategy): *Prop* :=

$\forall$ (*a*:*Agent*) (*s*':*FinStrategy*), $s \vdash^{a} \dashv s' \rightarrow (f2u\ s'\ a \preceq f2u\ s\ a)$.

Backward induction

On finite strategies.

Inductive BI: $FinStrategy \rightarrow Prop :=$

$|\ BILeaf: \forall\ uf{:}Utility\_fun,\ BI\ (sLeaf\ \ uf)$

$|\ BINode\_left: \forall\ (a{:}Agent)\ (sl\ sr{:}\ FinStrategy),$
$\quad BI\ sl \rightarrow BI\ sr \rightarrow (f2u\ sr\ a \preceq f2u\ sl\ a) \rightarrow BI\ (sNode\ a\ left\ sl\ sr)$

$|\ BINode\_right: \forall\ (a{:}Agent)\ (sl\ sr{:}\ FinStrategy),$
$\quad BI\ sl \rightarrow BI\ sr \rightarrow (f2u\ sl\ a \preceq f2u\ sr\ a) \rightarrow BI\ (sNode\ a\ right\ sl\ sr).$

Backward induction

On finite strategies.

Inductive *BI*: *FinStrategy* $\rightarrow$ *Prop* :=

| *BILeaf*: $\forall$ *uf*:*Utility_fun*, *BI* (*sLeaf uf*)

| *BINode_left*: $\forall$ (*a*:*Agent*) (*sl sr*: *FinStrategy*),
BI sl $\rightarrow$ *BI sr* $\rightarrow$ (*f2u sr a* $\preceq$ *f2u sl a*) $\rightarrow$ *BI* (*sNode a left sl sr*)

| *BINode_right*: $\forall$ (*a*:*Agent*) (*sl sr*: *FinStrategy*),
BI sl $\rightarrow$ *BI sr* $\rightarrow$ (*f2u sl a* $\preceq$ *f2u sr a*) $\rightarrow$ *BI* (*sNode a right sl sr*).

If *s* is *BI* then *s* is a Nash equilibrium.

Backward induction

On finite strategies.

Inductive *BI*: *FinStrategy* $\rightarrow$ *Prop* :=

| *BILeaf*: $\forall$ *uf*:*Utility_fun*, *BI* (*sLeaf uf*)

| *BINode_left*: $\forall$ (*a*:*Agent*) (*sl sr*: *FinStrategy*),
BI sl $\rightarrow$ *BI sr* $\rightarrow$ (*f2u sr a* $\preceq$ *f2u sl a*) $\rightarrow$ *BI* (*sNode a left sl sr*)

| *BINode_right*: $\forall$ (*a*:*Agent*) (*sl sr*: *FinStrategy*),
BI sl $\rightarrow$ *BI sr* $\rightarrow$ (*f2u sl a* $\preceq$ *f2u sr a*) $\rightarrow$ *BI* (*sNode a right sl sr*).

If *s* is *BI* then *s* is a Nash equilibrium.

Theorem *BI_is_FinNashEq* : $\forall$ *s*, *BI s* $\rightarrow$ *FinNashEq s*.

Coinductive *Games*

CoInductive *Game* : *Set* :=
| *gLeaf* : *Utility_fun* → *Game*
| *gNode* : *Agent* → *Game* → *Game* → *Game*.

Coinductive *Games*

CoInductive *Game* : *Set* :=
| *gLeaf* : *Utility_fun* → *Game*
| *gNode* : *Agent* → *Game* → *Game* → *Game*.

Either a leaf or a triple with an agent and two subgames that are infinite

Coinductive *Games*

CoInductive *Game* : *Set* :=
| *gLeaf* : *Utility_fun* → *Game*
| *gNode* : *Agent* → *Game* → *Game* → *Game*.

Either a leaf or a triple with an agent and two subgames that are infinite

The concept of infinite strategy is also defined as a coinductive:

CoInductive *Strategy* : *Set* :=
| *sLeaf* : *Utility_fun* → *Strategy*
| *sNode* : *Agent* → *Choice* → *Strategy* → *Strategy* → *Strategy*.

From infinite strategy to utility function

The utility function *i2u* is no more a function, but a relation, since it is no more total.

From infinite strategy to utility function

The utility function *i2u* is no more a function, but a relation,
since it is no more total.

It returns a value only on strategies which go eventually to a leaf.

The predicate eventually to the right

I introduce a predicate on strategies,

- called eventually to a leaf and
- written *LeadsToLeaf* .

The predicate eventually to the right

I introduce a predicate on strategies,

- called eventually to a leaf and
- written *LeadsToLeaf*.

On strategies that go eventually to a leaf, one gets
existence and uniqueness of the utility
associated with each agent.

The predicate eventually to the right

I introduce a predicate on strategies,

- called eventually to a leaf and
- written *LeadsToLeaf*.

On strategies that go eventually to a leaf, one gets
existence and uniqueness of the utility
associated with each agent.

Lemma *Existence_i2u*: $\forall$ (*a*:*Agent*) (*s*:*Strategy*),
LeadsToLeaf s $\rightarrow$ $\exists$ *u*:*Utility*, *i2u a u s*.

Lemma *Uniqueness_i2u*: $\forall$ (*a*:*Agent*) (*u v*:*Utility*) (*s*:*Strategy*),
LeadsToLeaf s $\rightarrow$ *i2u a u s* $\rightarrow$ *i2u a v s* $\rightarrow$ *u*=*v*.

Nash equilibria

Definition *NashEq* (s: Strategy): Prop :=
$\forall$ a s' u u',
$s' \vdash^{a} \dashv s \rightarrow$
$LeadsToLeaf\ s' \rightarrow (s2u\ s'\ a\ u') \rightarrow$
$LeadsToLeaf\ s \rightarrow (s2u\ s\ a\ u) \rightarrow (u' \preceq u).$

Nash equilibria

Definition *NashEq* (s: Strategy): Prop :=
$\forall$ a s' u u',
 s' $\vdash^{a}\dashv$ s $\rightarrow$
 LeadsToLeaf s' $\rightarrow$ (s2u s' a u') $\rightarrow$
 LeadsToLeaf s $\rightarrow$ (s2u s a u) $\rightarrow$ (u' $\preceq$ u).

Sub Game Perfect Equilibria

CoInductive *SGPE*: *Strategy* → *Prop* :=

| *SGPEnode_left*: ∀ (*a*:*Agent*)(*u*:*Utility*) (*sl*: *Strategy*) (*sr*: *Strategy*),
AlwLeadsToLeaf sl → *SGPE sl* → *BI sr* → *i2u a u sl* → (*f2u sr a* $\preceq$ *u*) →
SGPE (*sNode a left sl sr*)

| *SGPEnode_right*: ∀ (*a*:*Agent*) (*u*:*Utility*) (*sl*: *Strategy*) (*sr*: *Strategy*),
AlwLeadsToLeaf sl → *SGPE sl* → *BI sr* → *i2u a u sl* → (*u* $\preceq$ *f2u sr a*) →
SGPE (*sNode a right sl sr*).

Sub Game Perfect Equilibria

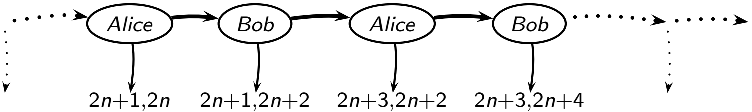
CoInductive *SGPE*: *Strategy* $\rightarrow$ *Prop* :=

| *SGPEnode_left*: $\forall$ (*a*:*Agent*)(*u*:*Utility*) (*sl*: *Strategy*) (*sr*: *Strategy*),
AlwLeadsToLeaf sl $\rightarrow$ *SGPE sl* $\rightarrow$ *BI sr* $\rightarrow$ *i2u a u sl* $\rightarrow$ (*f2u sr a* $\preceq$ *u*) $\rightarrow$
SGPE (*sNode a left sl sr*)

| *SGPEnode_right*: $\forall$ (*a*:*Agent*) (*u*:*Utility*) (*sl*: *Strategy*) (*sr*: *Strategy*),
AlwLeadsToLeaf sl $\rightarrow$ *SGPE sl* $\rightarrow$ *BI sr* $\rightarrow$ *i2u a u sl* $\rightarrow$ (*u* $\preceq$ *f2u sr a*) $\rightarrow$
SGPE (*sNode a right sl sr*).

Never give up

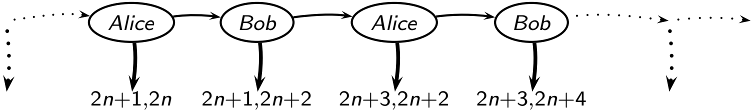
In Shubik's game, we can proof that the strategy never give up



is a Nash equilibrium.

Always give up

The strategy always give up



is a SubGame Perfect Equilibrium and a Nash equilibrium.

Conclusion

- Reasoning on infinite sequential games is subtle,

Conclusion

- Reasoning on infinite sequential games is subtle,
- Reductio ad absurdum is not needed ,

Conclusion

- Reasoning on infinite sequential games is subtle,
- Reductio ad absurdum is not needed nor non monotonic logic,

Conclusion

- Reasoning on infinite sequential games is subtle,
- Reductio ad absurdum is not needed ,
- I was able to explain the so-called irrationality of *escalation*

Conclusion

- Reasoning on infinite sequential games is subtle,
- Reductio ad absurdum is not needed ,
- I was able to explain the so-called irrationality of *escalation*
- Kim Jong Ill is rational.

Conclusion

- Reasoning on infinite sequential games is subtle,
- Reductio ad absurdum is not needed ,
- I was able to explain the so-called irrationality of *escalation*

The End!