Logic and Proofs

CSE203

Benjamin Werner — Pierre-Yves Strub

École polytechnique

Lecture 5

Inductive Properties

November 22rd 2022

Prop and bool

```
Inductive bool : Type :=
  | true : bool
  | false : bool.
```

which does not contain a variable

A *closed* term of type bool always computes to `true` or `false`

```
bool_ind : forall P : bool -> Prop,
              P true -> P false ->
                    forall b : bool, P b
```

When `P : nat -> bool`, then `P` is a <u>decidable</u> property:

we can check whether `(P x)` is `true` or `false`

Digression : arrives the indecidability

```
and : Prop -> Prop -> Prop

Definition andb b1 b2 : bool :=
 match b1, b2 with
| true, true => true
| _, _   => false
end.
```

`(andb P Q)` is still decidable

Same for or / ∨ and implication / →

But what about ∀ ?

Digression : arrives the indecidability (2)

`P : nat -> bool`

is `forall x, P x` true ?

We do not know !

This is where things become complicated !

Defining properties inductively

A way to define properties which are not necessarily decidable

Three ways to define even (1)

```
Fixpoint evenb n :=
  match n with
| 0 => true
| S n => negb (evenb n)
end.
```

```
Definition negb b:=
  match b with
| true => false
| false => true
end.
```

We can prove :

- `evenb 0`
- `forall n, evenb (S (S n))  = evenb n`
- `evenb 1 = false`

```
forall b, negb (negb b) = b
```

(2) with logic and functions

```
Definition evenl n :=
   exists p, n = p + p.
```

We can prove :

- `evenl 0`
- `forall n, evenl (S (S n))  <-> evenl n`
- `~(evenl 1)`
- `forall n, evenl n  <-> evenb n`

Alternative (3): inductive property

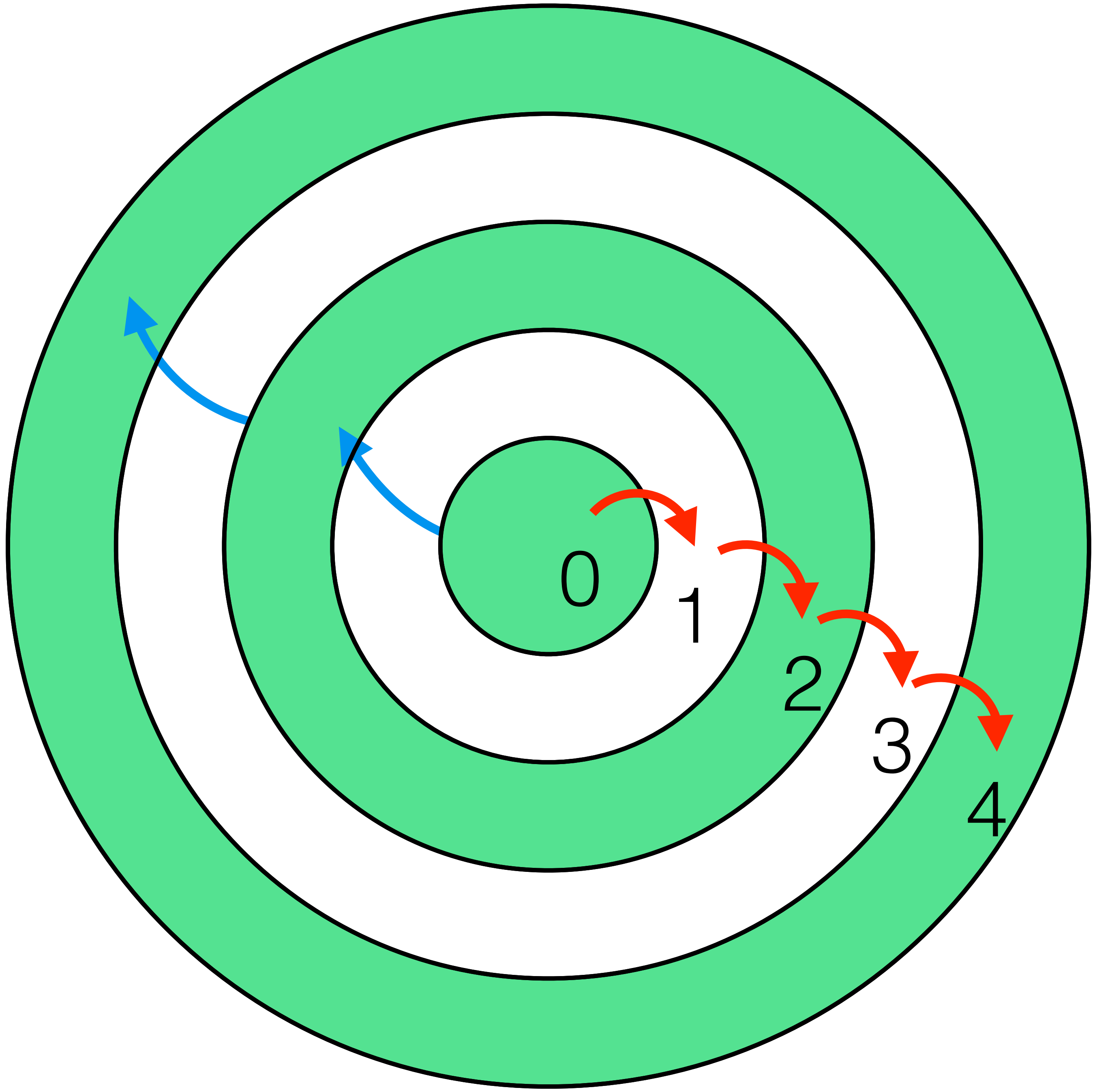
```
Inductive even : nat -> Prop :=
| even0 : even 0
| evenSS : forall n, even n -> even (S (S n)).
```

```
even 0
even (S (S 0))
even 4
even 6
…
```

The smallest set of natural numbers such that:

- $0 \in Even$
- if $n \in Even$, then $n+2 \in Even$

```
Inductive nat : Type :=
  O : nat
  S : nat -> nat
```



"layer construction"

Inductive property: **`even`**

```
 evenO : even O
 evenS : forall n, even n -> even (S (S n))
```

Induction principle for **even**

States that even is indeed the smallest set which:

- contains 0
- is closed by +2

```
forall P : nat -> Prop,
   P 0 ->
   (forall m, P m -> P (S (S m))) ->
    forall n, even n -> P n
```

even ⊆ P

This principle is generated when **even** is inductively defined

`forall n, evenb n -> even n` by induction over n

`forall n, even n -> evenb n` by induction over even n

A proof by induction over `(even n)`

```
n : nat
h : even n
================
evenb n
```

```
induction h.
```

```
forall P : nat -> Prop,
   P 0 ->
   (forall m, P m -> P (S (S m))) ->
    forall n, even n -> P n
```

- `evenb 0`
- `forall m, evenb m -> evenb (S (S m))`

```
evenb 0 ->
 (forall m, evenb m -> evenb (S (S m))) ->
  forall n, even n -> evenb n
```

A more usual but more difficult induction

```
forall n, evenb n -> even n
```

- base case: `even 0` ok
- `evenb (S n) -> even (S n)`

  we are stuck

We need to *strengthen the induction hypothesis*

```
forall n,
     (evenb n -> even n)
   /\(evenb (pred n) -> even (pred n))
```

where `(pred (S n)) = n`

Order over natural numbers

```
Inductive le : nat -> nat -> Prop :=
| le_refl : forall n, le n n
| le_S : forall n m, le n m -> le n (S m).
```

Variant :

```
Inductive le (n:nat) : nat -> Prop :=
| le_refl : le n n
| le_S : forall m, le n m -> le n (S m).
```

Structure

`(le 6 6)`

just `le_refl`

`(le 6 10)`

How is it proved ?

- 4 times `le_S`
- `le_refl` to finish

A proof of `(le n m)` is of size `m-n`

Example

```
forall n m, le n m -> exists p, m = n + p
```

We will prove it by induction over `(le n m)`

Permutations over lists

One possible definition :

```
Inductive permI  :list A ->list A ->Prop:=
|permI_refl:forall l, permI l l
|permI_cons:forall a l0 l1, permI l0 l1-> permI (cons a l0)(cons a l1)
|permI_end:forall a l, permI (cons a l) (app l (cons a nil))
|permI_trans:forall l1 l2 l3,
              permI l1 l2 -> permI l2 l3 -> permI l1 l3.
```

Many technical lemmas needed

An old friend

```
Inductive myst (a : nat) : nat -> Prop :=
| R : myst a.
```

What is this ?

```
myst_ind : forall P : nat -> Prop,
    (P a) ->
       forall x, myst x -> P x.
```

It is equality !
more precisely, "being equal to a"

An old friend

We have: `a : nat`

```
Inductive myst : nat -> Prop :=
| R : myst a.
```

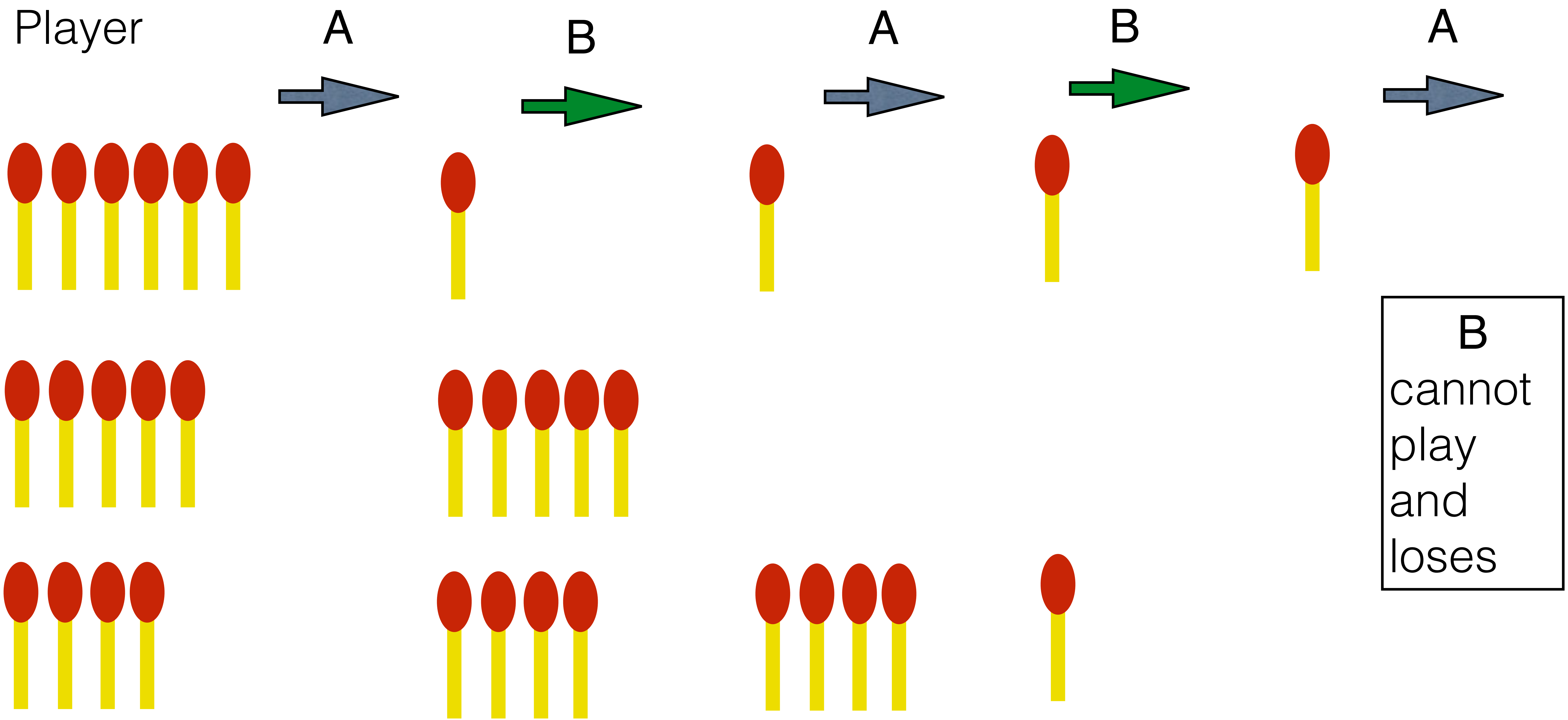
What is this ?
The property only verified by a
"being equal to a"
We have defined equality !

```
myst_ind : forall P : nat -> Prop,
    (P a) ->
       forall x, myst x -> P x.
```

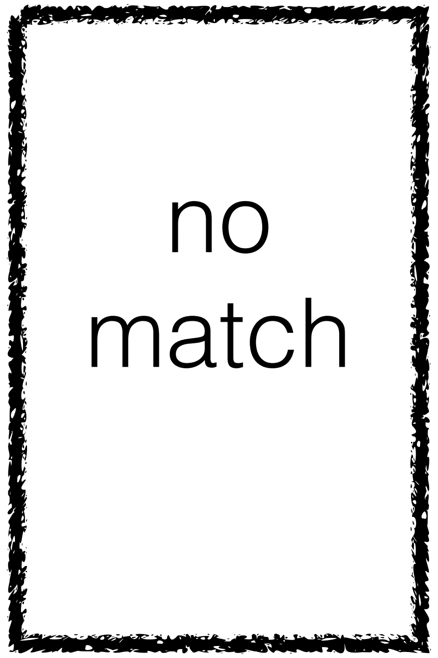
This year's project: the Game of Nim

L'année dernière à Marienbad

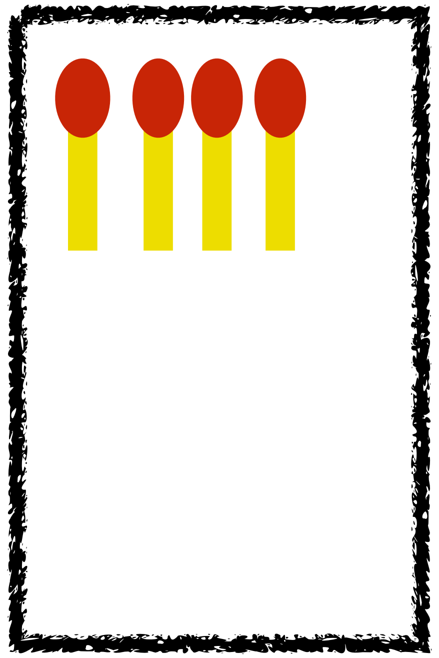
The Game of Nim



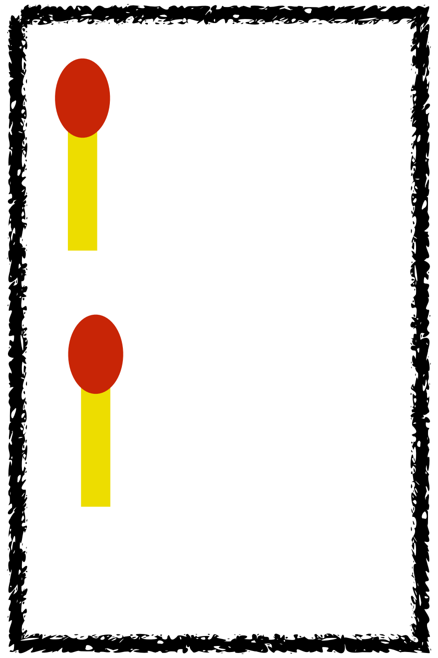
Who wins ?



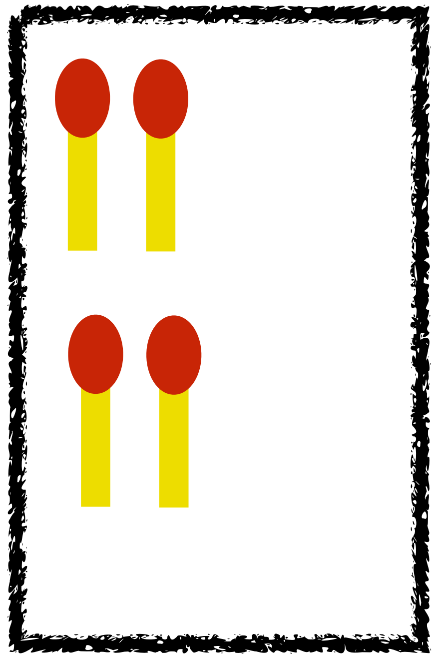
losing position
(actually lost)



winning
position



losing position



losing position

Defining who wins

An inductive definition :

1. no matches is a losing situation
2. for any situation x, if there exists a losing situation y, s.t. x -> y, then x is a winning situation
3. for any situation x, if for all y s.t. x->y, y is a winning situation, then x is a losing situation

winning situation = there exists a winning strategy
losing situation = the player cannot be sure to win (whatever he/she plays)

Question: can we determine whether a given situation is winning ?

Brutal

It should be decidable :

- finite number of possible moves
- games have finite number of moves

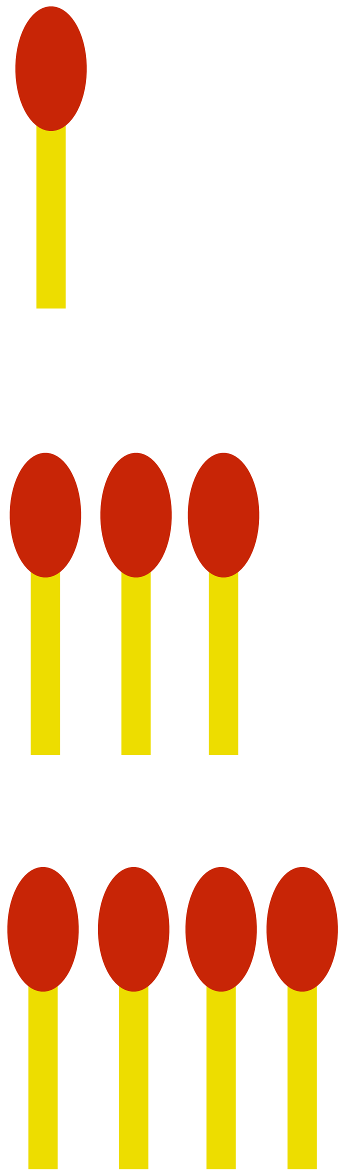
=> One can explore the whole tree

How would you do that ?

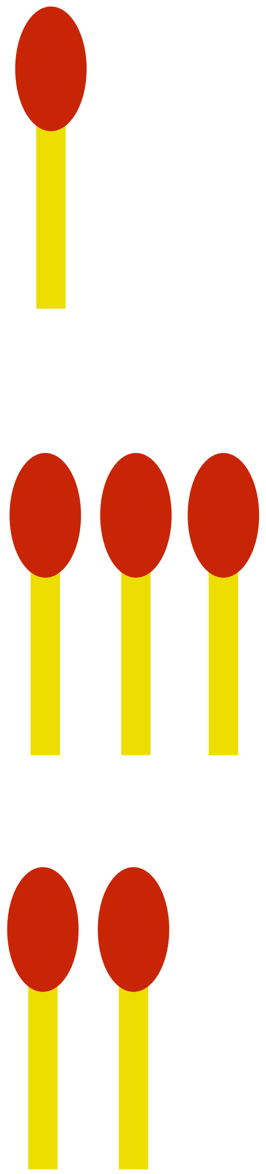
You can do it by dynamic programming !

(but it is not what we interested in here)

Clever: finding the invariant



1	001	
3	011	XOR
4	100	
	110	



1	001
3	011
2	010
	000

claim: if this is 0, then it is a losing situation

How is it formalized ?

- We will give you the outline
- One needs results about sequences of bits (we will give you most)
- Then show the main results
  - From any non-zero position, one can go to a zero-position in one move
  - A position is winning if and only if it is non-zero
  - (or losing if and only if it is zero)