Chapter 1

LotoR

1.1 Utilitaires

```
Lemma neq_beq_false:
  ∀ n m : nat,
    n ≠ m → beq_nat n m = false.
Fixpoint frequence (n:nat) (l:list nat) : nat :=
  match l with
    | nil ⇒ 0
    | m::l' ⇒ if (beq_nat n m) then
                  S (frequence n l')
                else frequence n l'
end.
```

1.2 Machine abstraite

```
Module AbstractLoto.
Module Context.
End Context.
```

1.2.1 Etat et invariant

```
Record State : Set := mkState {
  bouboules : list nat
}.
Definition Inv_1 (B : State) : Prop :=
  ∀ n : nat,
    frequence n B.(bouboules) ≤ 1.
```

1.2.2 Evénement : initialisation

```
Module INIT.

Definition Guard (bs:list nat) : Prop :=
  ∀ n : nat, frequence n bs ≤ 1.

Definition action (bs:list nat) : State :=
  mkState bs.

Theorem PO_Safety:
  ∀ bs : list nat,

    Guard bs

    → let B := action bs
      in Inv_1 B.

End INIT.
```

1.2.3 Evénement : tirage d'une boule

```
Module PICK.

Definition Guard (B:State) : Prop :=
  0 < length B.(bouboules).

Definition action_Prop_1 (B B': State) : Prop :=
  ∀ n : nat, frequence n B'.(bouboules) ≤ frequence n B.(bouboules).

Definition action_Prop_2 (B B': State) : Prop :=
  length B.(bouboules) = S (length B'.(bouboules)).

Definition action_witness (B : State) : State :=
  match B.(bouboules) with
    | nil ⇒ B
    | n::bs ⇒ mkState bs
  end.

Lemma PO_Feasibility_1:
  ∀ B : State,

    Inv_1 B

    → Guard B

    → let B' := action_witness B
      in
      action_Prop_1 B B'.
```

```
Lemma PO_Feasibility_2:
  ∀ B : State,

     Inv_1 B

     → Guard B

     → let B' := action_witness B
        in
        action_Prop_2 B B'.
Theorem PO_Feasibility:
  ∀ B:State,

     Inv_1 B

     → Guard B

     → ∃ B' : State,
          action_Prop_1 B B'
          ∧ action_Prop_2 B B'.
Theorem PO_Safety:
  ∀ B:State,

     Inv_1 B

     → Guard B

     → ∀ B' : State,
          action_Prop_1 B B'
          → action_Prop_2 B B'

          → Inv_1 B'.
End PICK.
```

1.2.4 Exercice

Question 1

Définir un evénement non-déterministe *Poke* qui ajoute une boule dans l'état courant, sans paramètre.

Question 2

Utiliser comme témoin la valeur maximale des boules présentes plus 1.

Question 3

Montrer les obligations de preuves *PO_Feasibility* et *PO_Safety*.

`End` AbstractLoto.

1.3 Machine concrète

`Module` ConcreteLoto.

`Module` Context.

`End` Context.

1.3.1 Etat et invariant

```
Record State : Set := mkState {
  bouboules : list nat
}.
Definition Glue_1 (B : State) (AB : AbstractLoto.State) : Prop :=
    B.(bouboules) = AbstractLoto.bouboules AB.
```

1.3.2 Evénement : initialisation

`Module` Init.

```
Definition Guard (bs:list nat) : Prop :=
  ∀ n : nat, frequence n bs ≤ 1.
Definition action (bs:list nat) : State :=
  mkState bs.
Theorem PO_Strengthening:

  ∀ bs : list nat,

    Guard bs
    →

    AbstractLoto.Init.Guard bs.

Theorem PO_Simulation:
```

```
  ∀ bs : list nat,

    Guard bs
    → let B := action bs
      in
      let AB := AbstractLoto.Init.action bs
      in

      Glue_1 B AB.
End INIT.
```

1.3.3 Evénement raffiné : tirage d'une boule

```
Module PICK.
Definition Guard (B:State) : Prop :=
  0 < length B.(bouboules).
Fixpoint min (l: list nat) : nat :=
  match l with
    | nil ⇒ 0
    | n::l' ⇒ let m := min l'
              in if leb m n then n else m
  end.
Fixpoint remove_elem (n:nat) (l : list nat) : list nat :=
  match l with
    | nil ⇒ l
    | m::l' ⇒ if beq_nat n m then l'
              else m::(remove_elem n l')
  end.
Definition action (B:State) : State :=
  mkState
    (remove_elem (min B.(bouboules)) B.(bouboules)).
Theorem PO_Strengthening:
  ∀ B : State, ∀ AB : AbstractLoto.State,

    AbstractLoto.Inv_1 AB

    → Glue_1 B AB



    → Guard B
```

```
    → AbstractLoto.Pick.Guard AB.

Lemma lt_diff:
  ∀ n m : nat, n < m → n ≠ m.

Lemma frequence_min_remove:
  ∀ n : nat, ∀ l : list nat,
    frequence n (remove_elem (min l) l) ≤
    frequence n l.

Lemma length_remove_min:
  ∀ l : list nat,
    0 < length l → length l = S (length (remove_elem (min l) l)).
Theorem PO_Simulation:
  ∀ (B:State), ∀ (AB:AbstractLoto.State),

    AbstractLoto.Inv_1 AB

    → Glue_1 B AB



    → Guard B

    → let B' := action B
      in ∃ AB' : AbstractLoto.State,
           AbstractLoto.Pick.action_Prop_1 AB AB'
           ∧ AbstractLoto.Pick.action_Prop_2 AB AB'

           ∧ Glue_1 B' AB'.
End PICK.
```

1.3.4 Exercice

Question 1

Proposer un raffinement de l'opération abstraite *Poke* telle que la boule ajoutée est le premier "trou" rencontré (boule dont le numéro non-présent est minimal).

Question 2

Montrer les obligations de preuve *PO_Strengthening* et *PO_Simulation*.

```
End CONCRETELOTO.
```