Modèles Evénementiels

Partie 2 : Raffinement

Spécification et Validation de Programmes (SVP) - M2 STL 2015

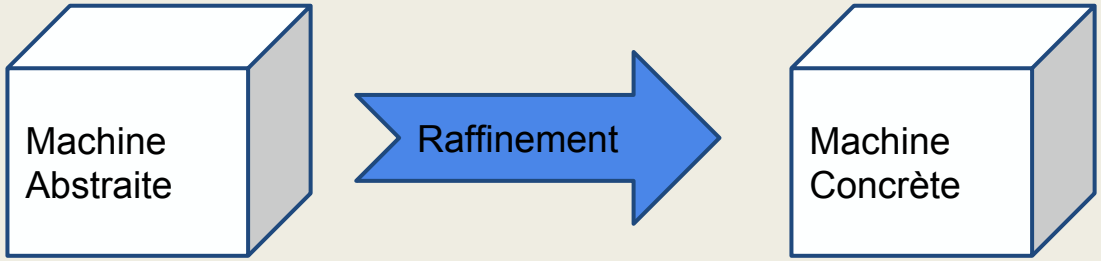
Raffinement : méthodologie

Du point de vue méthodologique le raffinement permet :

Une approche descendante itérative de la spécification :

- Spécification (la plus) abstraite
  => propriétés essentielles du système
- Spécification (plus) concrète
  => ajout de détails
- …
- Spécification (la plus) concrète
  => réalisation du système

Raffinement en B



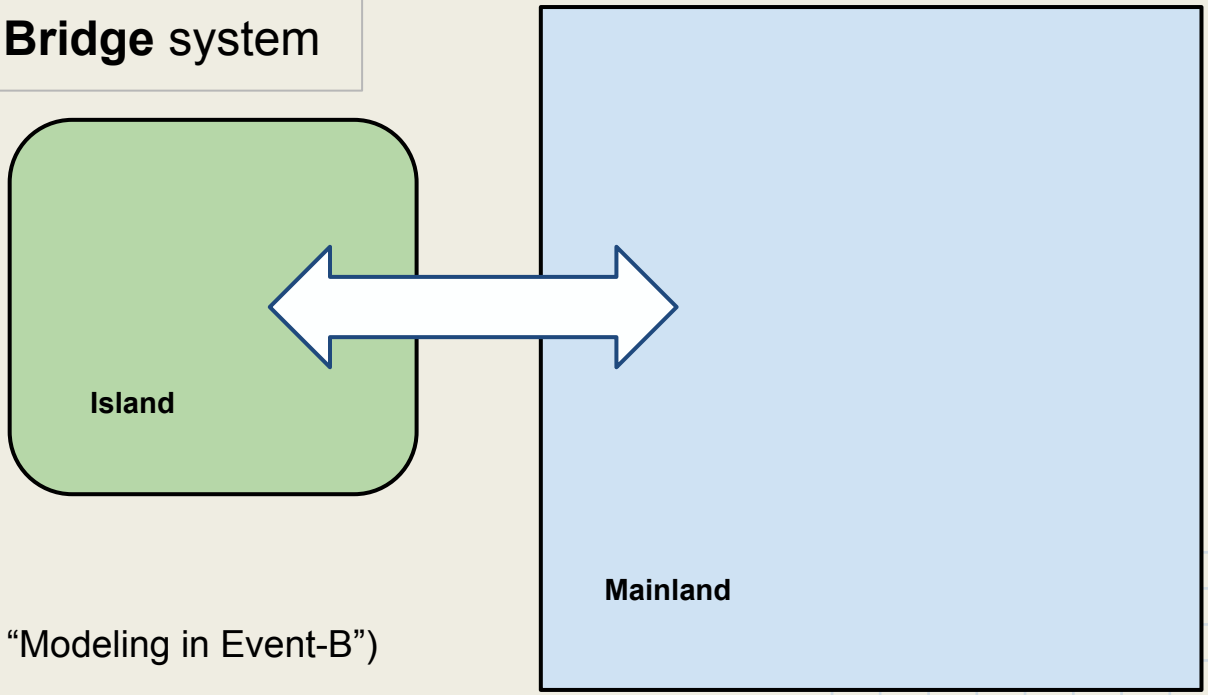
- Etat abstrait
- Invariants abstraits
- Evénements abstraits
  - PO safety
  - convergence (opt.)
- Deadlock freedom (opt.)

- Etat concret
- Invariants
  - invariants de glue
  - invariants concrets
- Evénements concrets
  - **événements raffinés**
    - **PO Strengthening + Simulation**
    - PO Safety, Convergence (optionnelle)
  - **événements concrets**
    - PO safety
    - **PO Simulation**
    - **PO Convergence** (obligatoire)
  - Relative deadlock freedom

Etude de cas : Machine abstraite

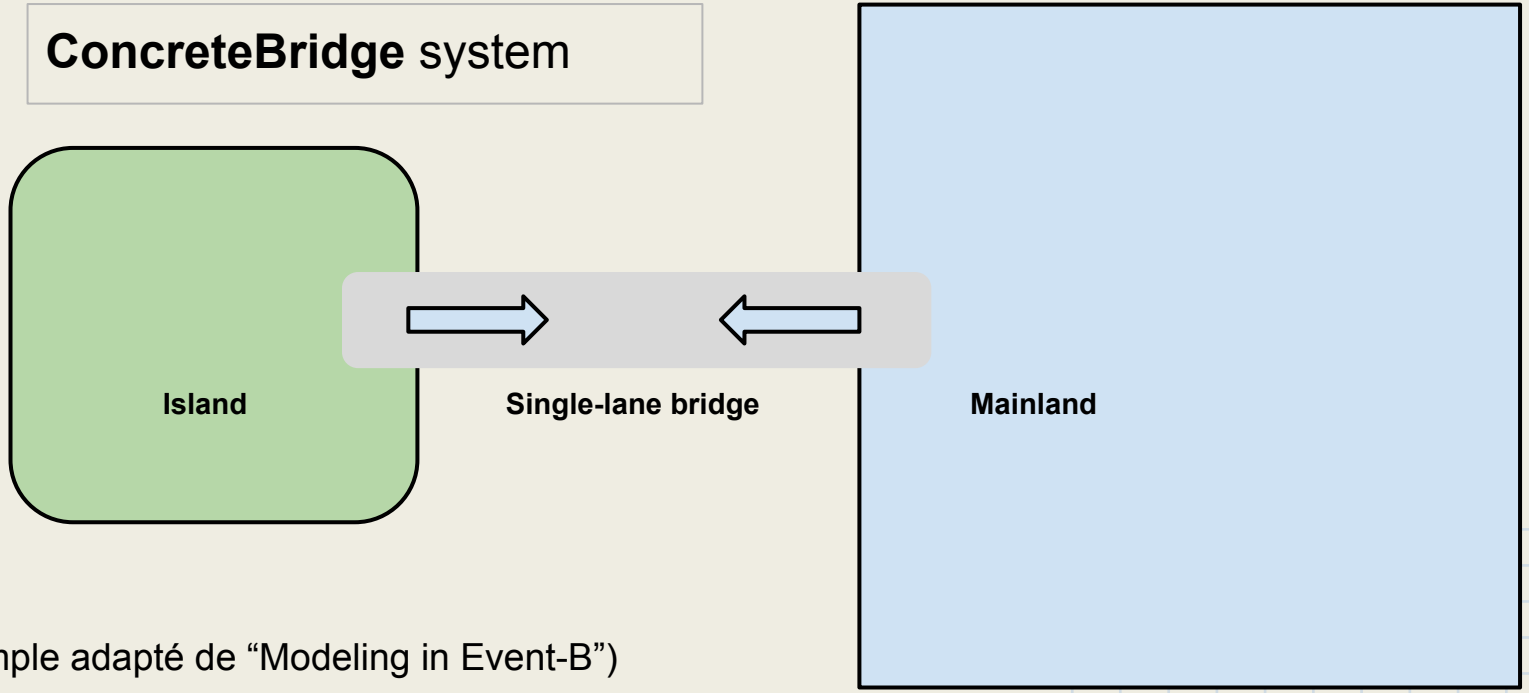
Exemple illustratif : le système *Bridge* avec raffinement

Abstract Bridge system



(exemple adapté de “Modeling in Event-B”)

Etude de cas : Machine concrète



(exemple adapté de “Modeling in Event-B”)

Contexte abstrait

Contexte = partie statique du système

- **Constantes abstraites**
  - paramètre(s) du système
- **Axiomes abstraits**
  - propriétés statiques sur les constantes

Contexte abstrait en Coq

```
Module AbstractMachine.

Module Context.
(* constante *)
Parameter <nom de la constante> : <type>.
...
(* axiome *)
Axiom <nom de l'axiome> : <proposition>.
...
End Context.

...
```

Contexte concret

Contexte = partie statique du système

- **Glue** abstrait ⇝ concret
  - exemple : axiome abstrait ⇒ lemme concret
- **Constantes concrètes**
  - paramètre(s) du système raffiné
- **Axiomes concrets**
  - propriétés statiques sur les constantes concrètes

Structure d'une machine concrète

⇒ Une **machine concrète** B contient :

- un état concret
  - **variables** d'état concret
- des invariants de glue : liaison avec l'état abstrait
- des invariants concrets : portant sur l'état concret
- des événements :
  - raffinements d'événements abstraits
  - événements concrets
- des preuves de propriétés
  - guidées par des **obligations de preuves** (**PO**)

Invariants de Glue

Les **invariants de Glue** expliquent la relation entre :

- les variables de l'état abstrait
- les variables de l'état concret

En Coq :

```
Definition Glue_<id numérique>
      (<cvar>:State) (<avar>:AbstractMachine.State)
   : Prop :=
      <proposition invariante>.
```

Invariants concrets

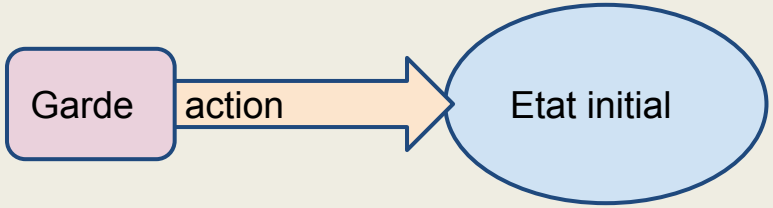
Un **invariant concret** exprime une propriété importante devant être toujours vérifiée par l'état concret de la machine.

En Coq :

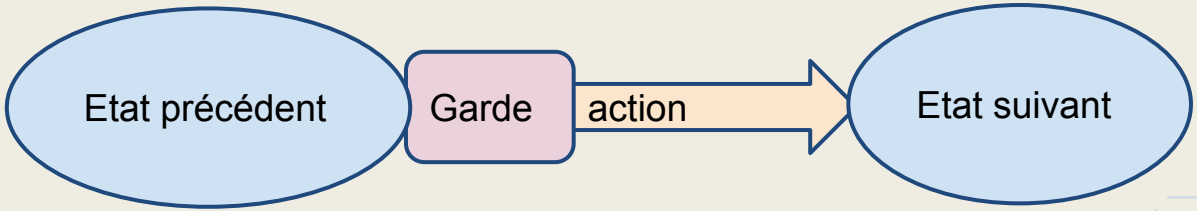
```
Definition Inv_<id numérique> (<variable>:State) : Prop :=
      <proposition invariante>.
```

Rappel : événéments

- Construction d'un état initial de la machine
  ⇒ **événement d'initialisation**



- soit un changement d'état
  ⇒ **événement de transition**

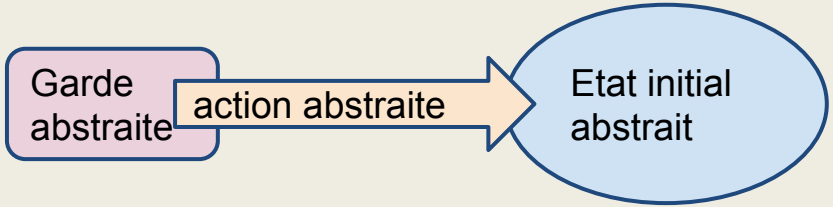


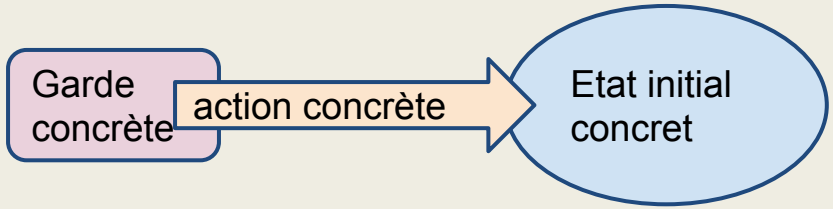
Evénements concrets et PO

1. **Evénements raffinés**
   - raffinement d'un événement abstrait
     - **PO Strengthening**
     - PO Safety
     - **PO Simulation**
     - PO Convergence
       (obligatoire si événement abstrait convergent)
2. **Evénements concrets**
     - PO Safety
     - **PO Simulation**
     - **PO Relative Convergence** (obligatoire)

\+ PO Relative Deadlock Freedom
(obligatoire si Deadlock Freedom au niveau abstrait)

Initialisation concrète



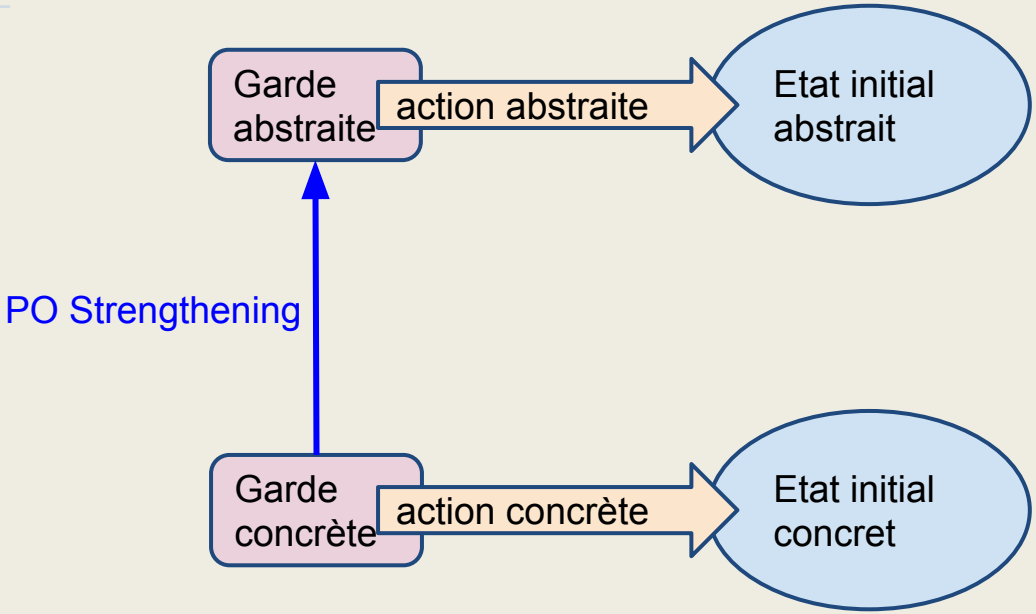


Initialisations concrètes en Coq

Evénement d'initialisation : description d'un état initial

```
Module <Nom de l'événement>.   (* généralement : Init *)

Definition Guard <paramètres> : Prop :=
      <proposition de garde>.

Definition action <paramètres> : State :=
      <définition de l'action : construction de l'état initial>.

<obligations de preuves (PO)>

End <nom de l'événement>.
```

PO Strengthening (initialisation)

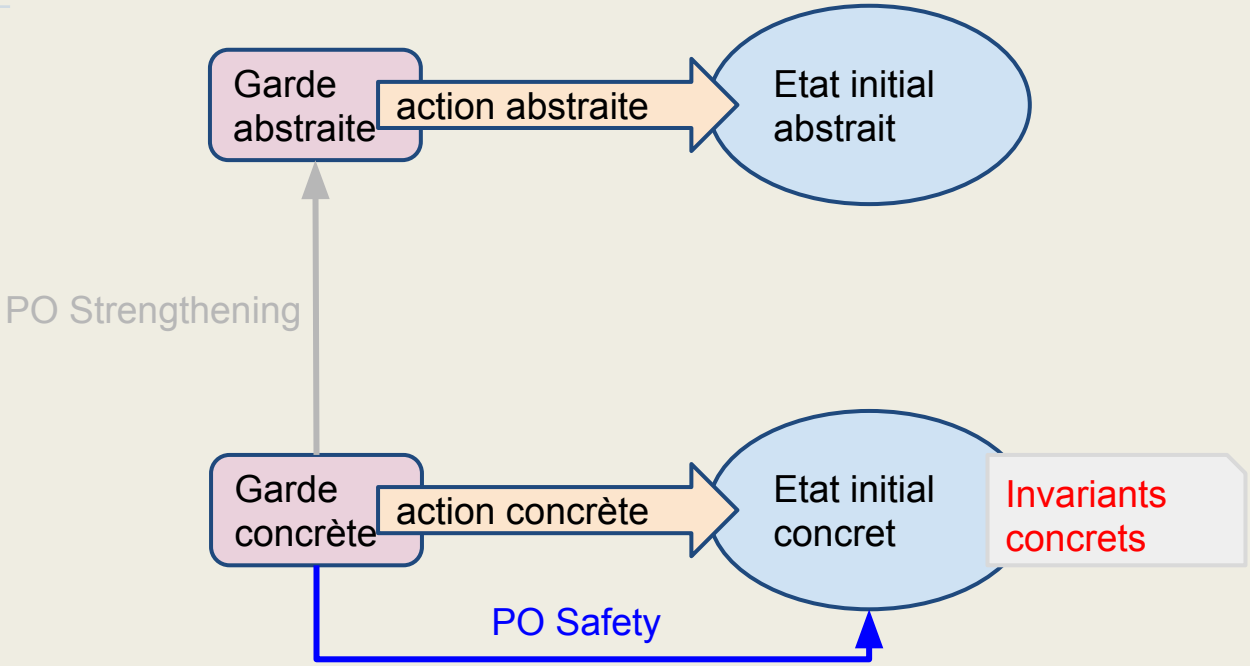


Strengthening : l'initialisation concrète ne contredit pas l'initialisation abstraite

PO Strengthening en Coq

```
Definition Guard (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (p1:T1) … (pN:TN) : State := <expr>.

Theorem PO_Strengthening:
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Garde concrète *)
    Guard p1 … pN
    -> (* Garde abstraite *)
       AbstractMachine.Init.Guard e1 … eM.
Proof.
```

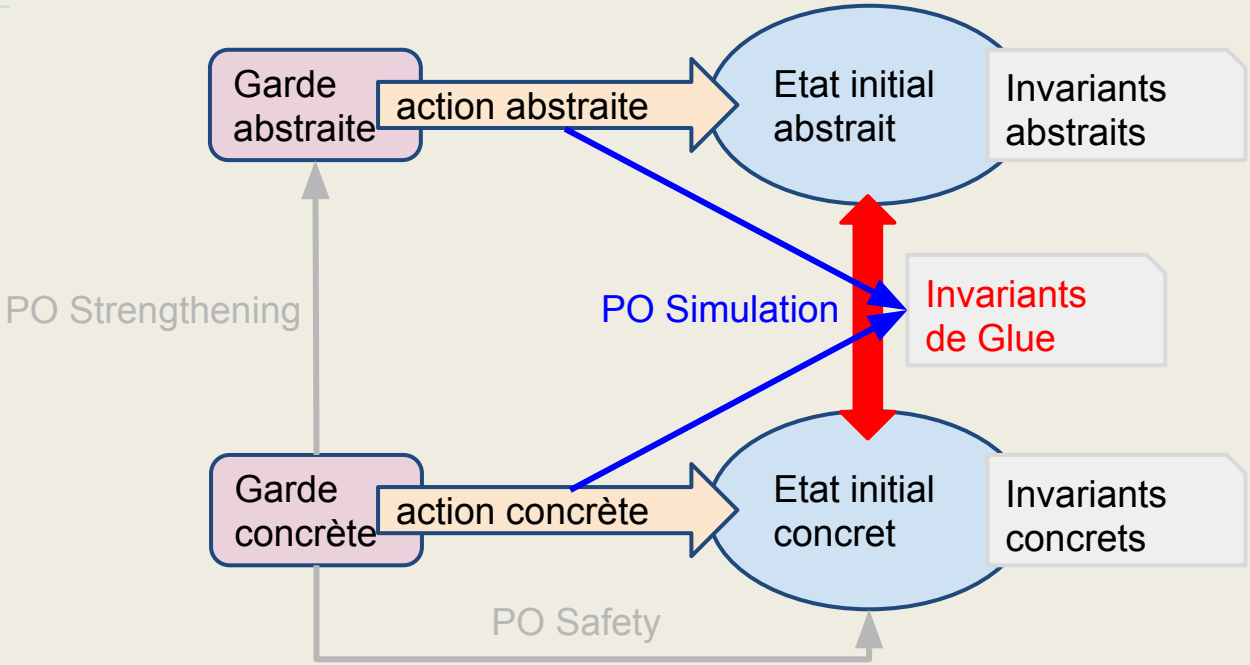
PO Safety (initialisation)



PO Safety en Coq

```
Definition Guard (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (p1:T1) … (pN:TN) : State := <expr>.

Theorem PO_Safety:
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Garde concrète *)
    Guard p1 … pN
    -> (* état initial concret *)
       let S := action p1 … pN
       in (* Invariants concrets *)
          Inv_1 S  /\ … /\ Inv_M S.
Proof.
```

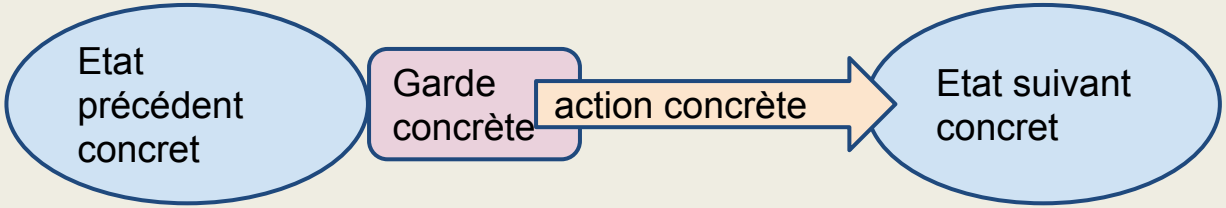
PO Simulation (initialisation)



PO Simulation en Coq

```
Definition Guard (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (p1:T1) … (pN:TN) : State := <expr>.

Theorem PO_Simulation:
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Garde concrète *)
    Guard p1 … pN
    -> (* état initial *)
       let S := action p1 … pN in
       let AS := AbstractMachine.Init.action e1 … eM in
       in (* Invariants de glue *)
          Glue_1 S AS  /\ … /\ Glue_P S AS.
Proof.
```

Transitions concrètes



Deux types de transition concrète :

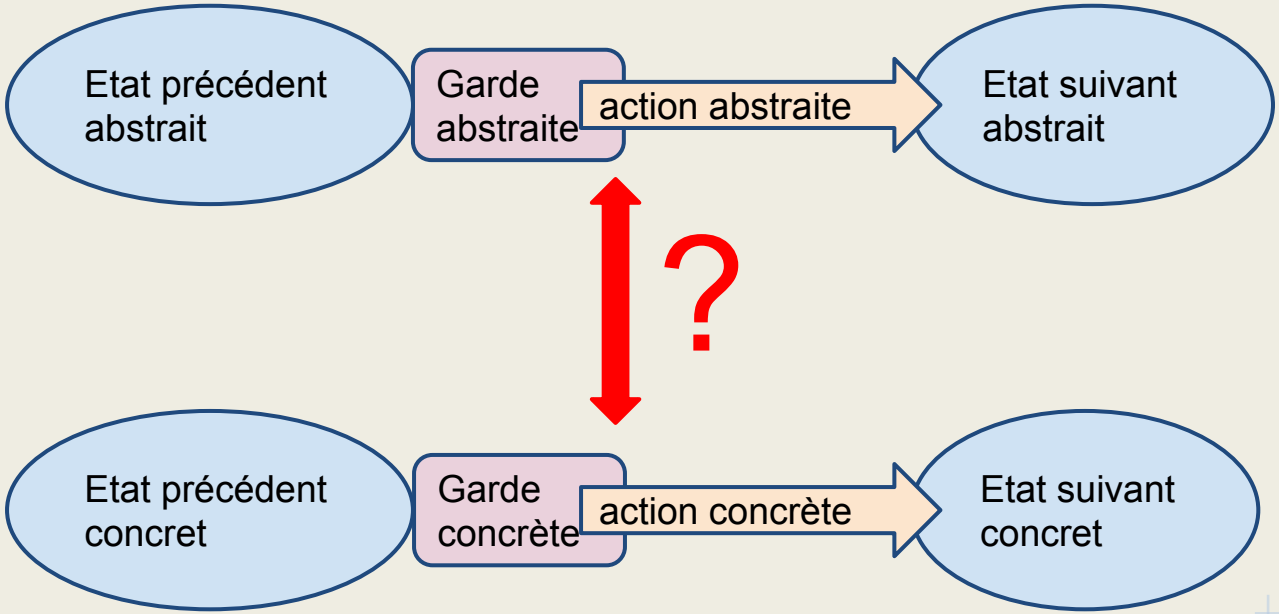
1. **transition de raffinement**
   - raffinement d'une transition abstraite
2. **transition concrète**
   - nouvelle transition

Transitions concrètes en Coq

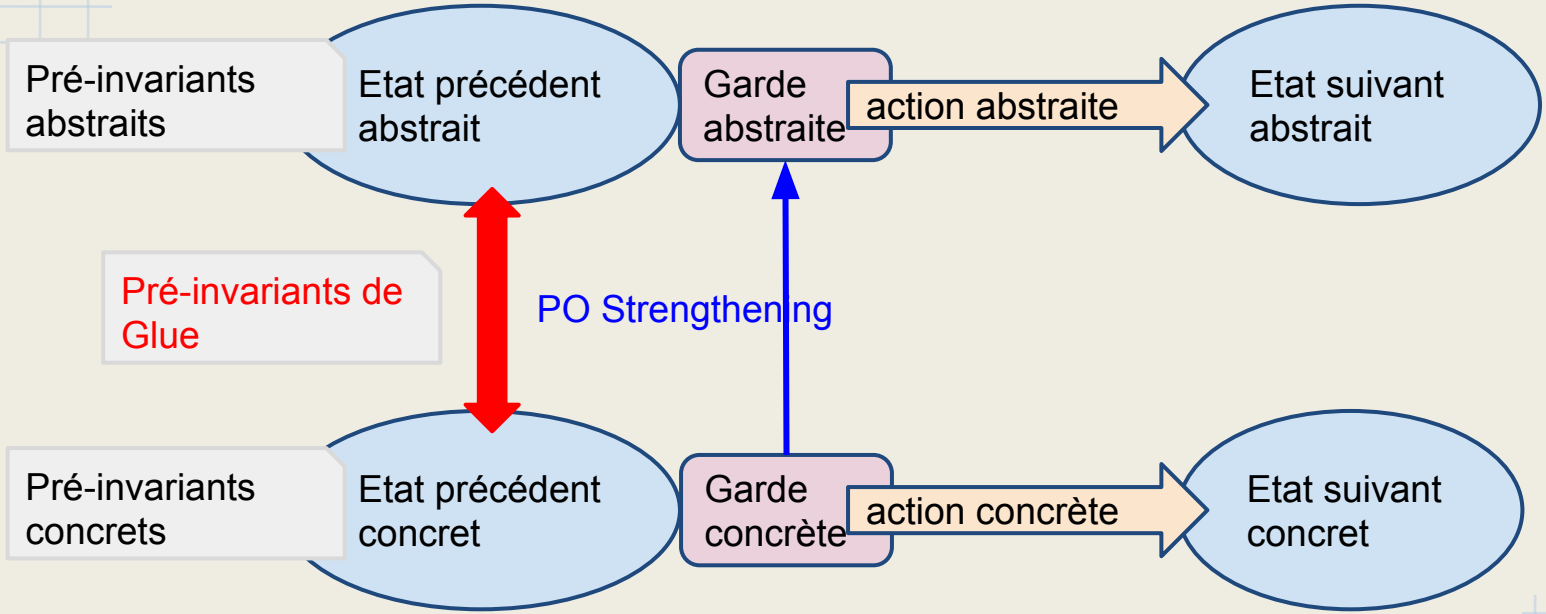
Evénement de transition : description d'un changement d'état atomique

```
Module <Nom de l'événement>.

Definition Guard (<variable d'état> : State) <autres paramètres> : Prop :=
      <proposition de garde>.

Definition action (<variable d'état> : State) <autres paramètres> : State :=
      <définition de l'action : construction de l'état suivant>.

<obligations de preuves (PO)>

End <nom de l'événement>.
```

Transitions concrètes : raffinement



PO Strengthening (transition raffinée)

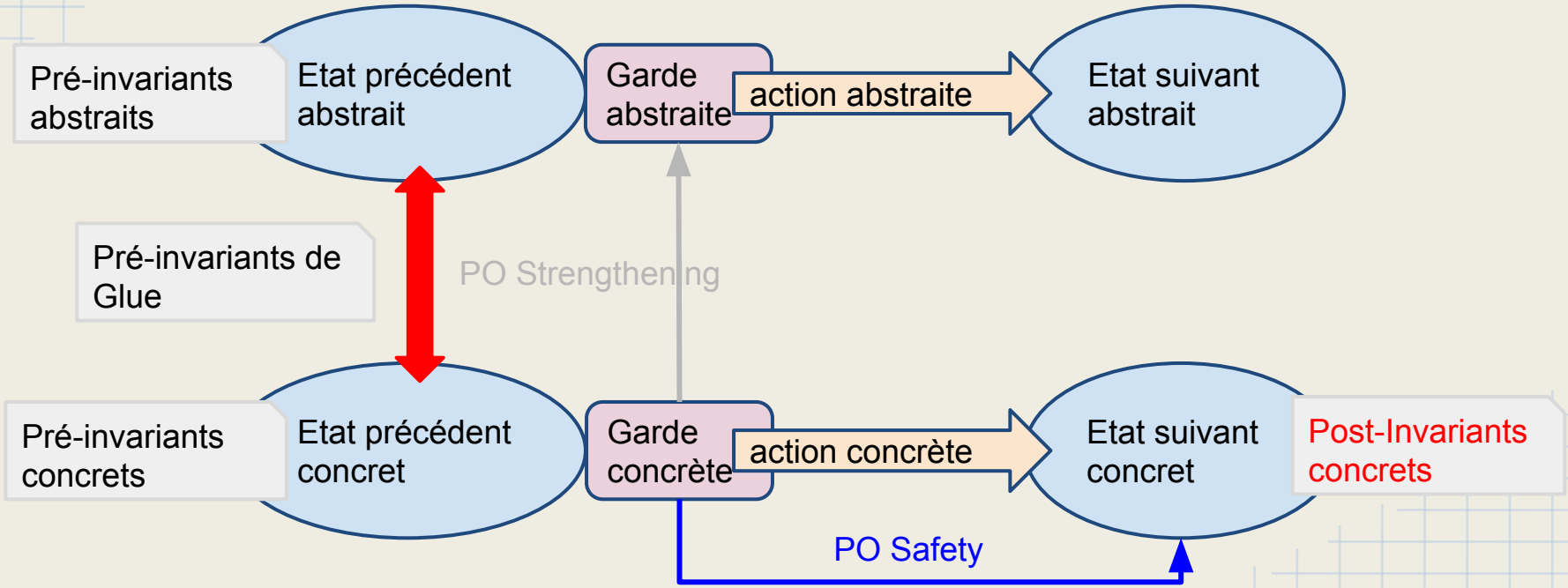


Strengthening : la transition concrète ne contredit pas l'initialisation abstraite

PO Strengthening en Coq

```
Definition Guard (S:State) (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (S:State) (p1:T1) … (pN:TN) : State := <expr>.

Theorem PO_Strengthening:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    -> (* Garde concrète *)
       Guard S p1 … pN
       -> (* ====> Garde abstraite *)
       AbstractMachine.Event.Guard AS.
Proof.
```

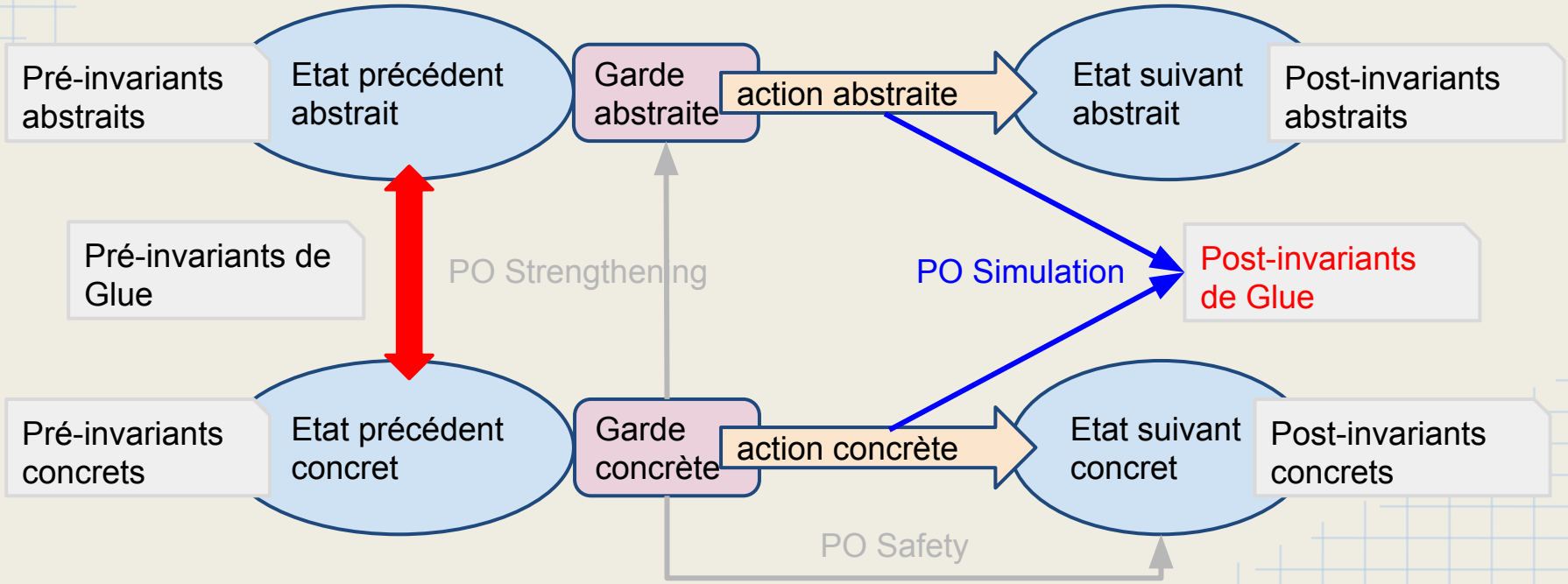
PO Safety (transition raffinée)



PO Safety en Coq

```
Definition Guard (S:State) (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (S:State) (p1:T1) … (pN:TN) : State := <expr>.

Theorem PO_Safety:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* Garde concrète *)
    -> Guard S p1 … pN
       (* état suivant *)
          let S' := action S p1 … pN
          in (* ====> Post-invariants concrets *)
             Inv_1 S' /\ … /\ Inv_Q S'
Proof.
```

PO Simulation (transition raffinée)



PO Simulation en Coq

```
Theorem PO_Simulation:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres concrets *)
  forall p1 : T1, …, forall pN : TN,
  (* paramètres abstraits *)
  forall q1 : U1, …, forall qL : UL,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* Garde concrète *)
    -> Guard S p1 … pN
       (* états suivants (concret/abstrait) *)
          let S' := action S p1 … pN
          in let S' := AbstractMachine.Event.action AS q1 … qL
             in(* ====> Post-invariants de glue *)
               Glue_1 S' AS' /\ … /\ Glue_Q S' AS'
Proof.
```

PO Convergence (raffinement)

Convergence d'un événement de transition :

- l'événement ne peut s'appliquer indéfiniment dans un état donné
- Optionelle sauf si l'événement raffiné est convergent.

Preuve de convergence :

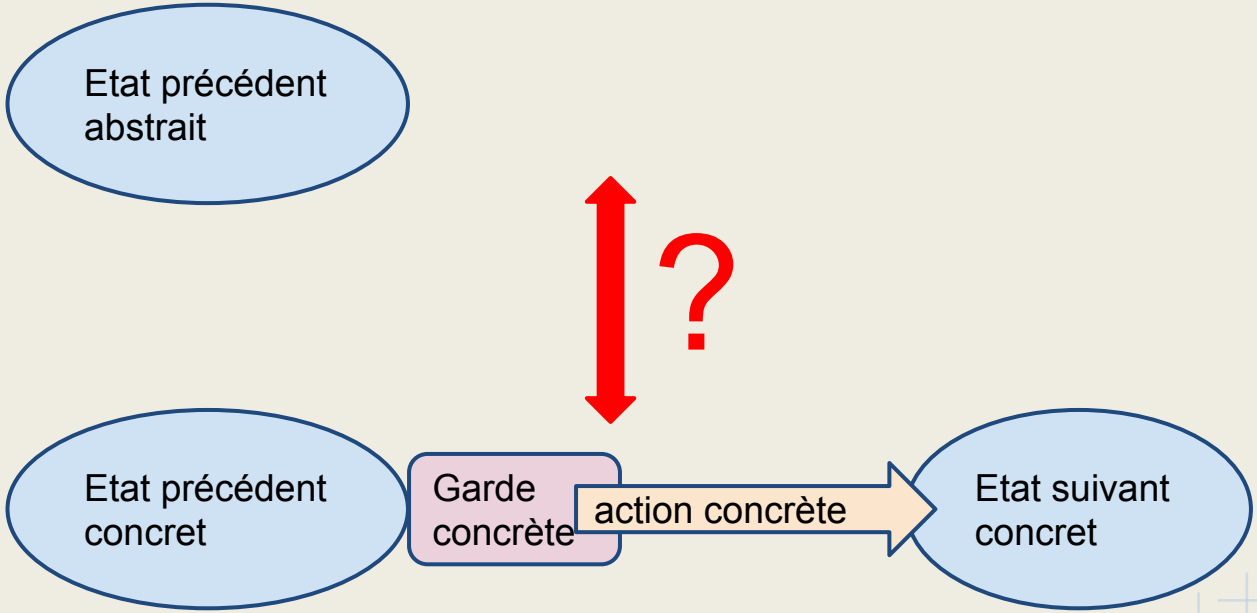
- Définition d'un **variant** d'état pour la transition
  - Mesure définie sur un ordre strict bien fondé
    ⇒ exemple : (nat, <)
  - Synthétisé à partir d'un état de la machine

- Preuve de **décroissance stricte du variant** durant la transition
  ⇒ variant *<état suivant>* < variant *<état précédent>*

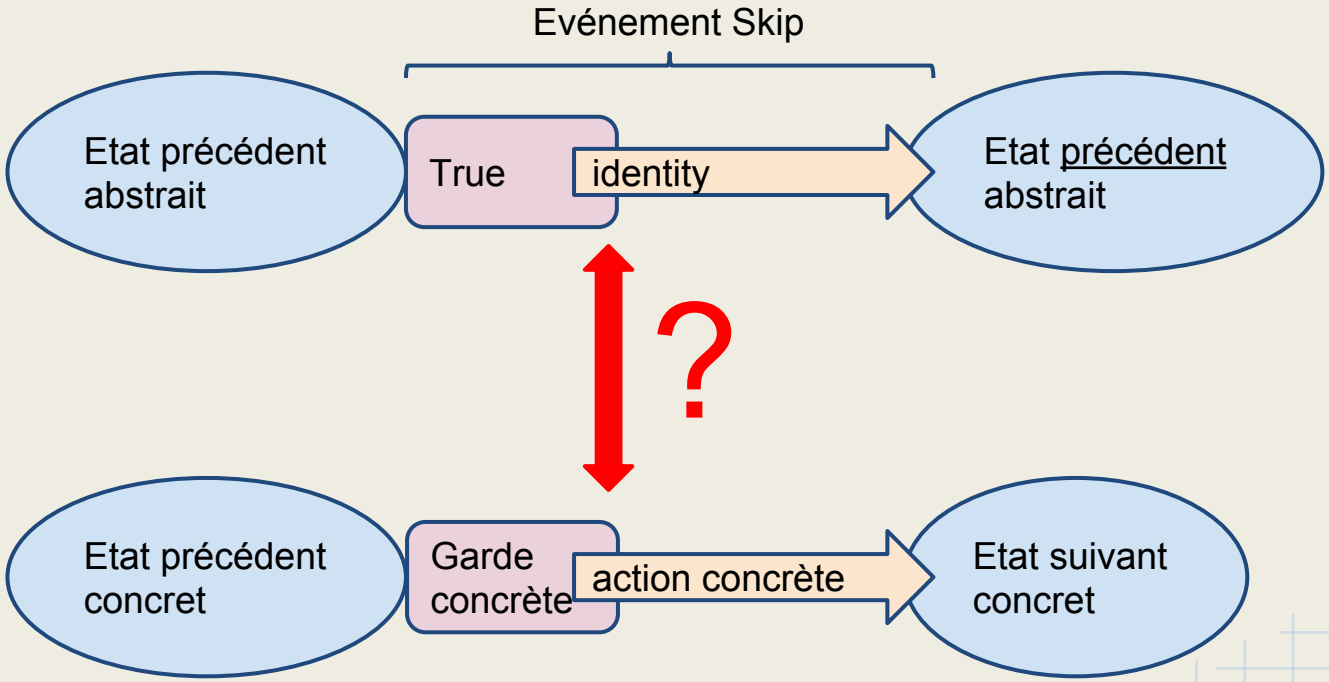
Convergence PO en Coq

```
Definition Guard (S:State) (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (S:State) (p1:T1) … (pN:TN) : State := <expr>.
Definition variant (S:State) : nat := <expr>.
Theorem PO_Convergence:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* Garde concrète *)
    -> Guard S p1 … pN
       (* état suivant *)
          let S' := action S p1 … pN
          in (* ====> Décroissance stricte du variant *)
              variant S' < variant S.
Proof.
```

Transitions concrètes

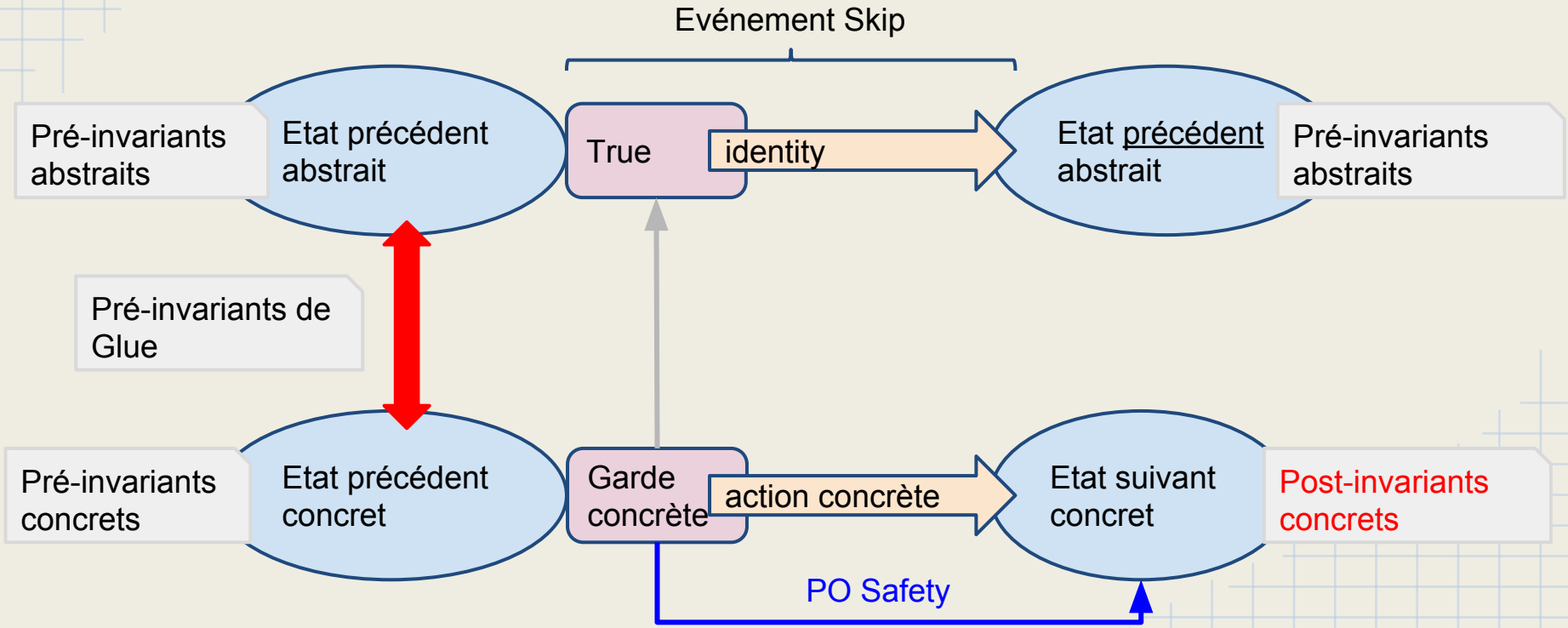


Transitions concrètes



Remarque : pas de *strengthening*

PO Safety (transition extension)

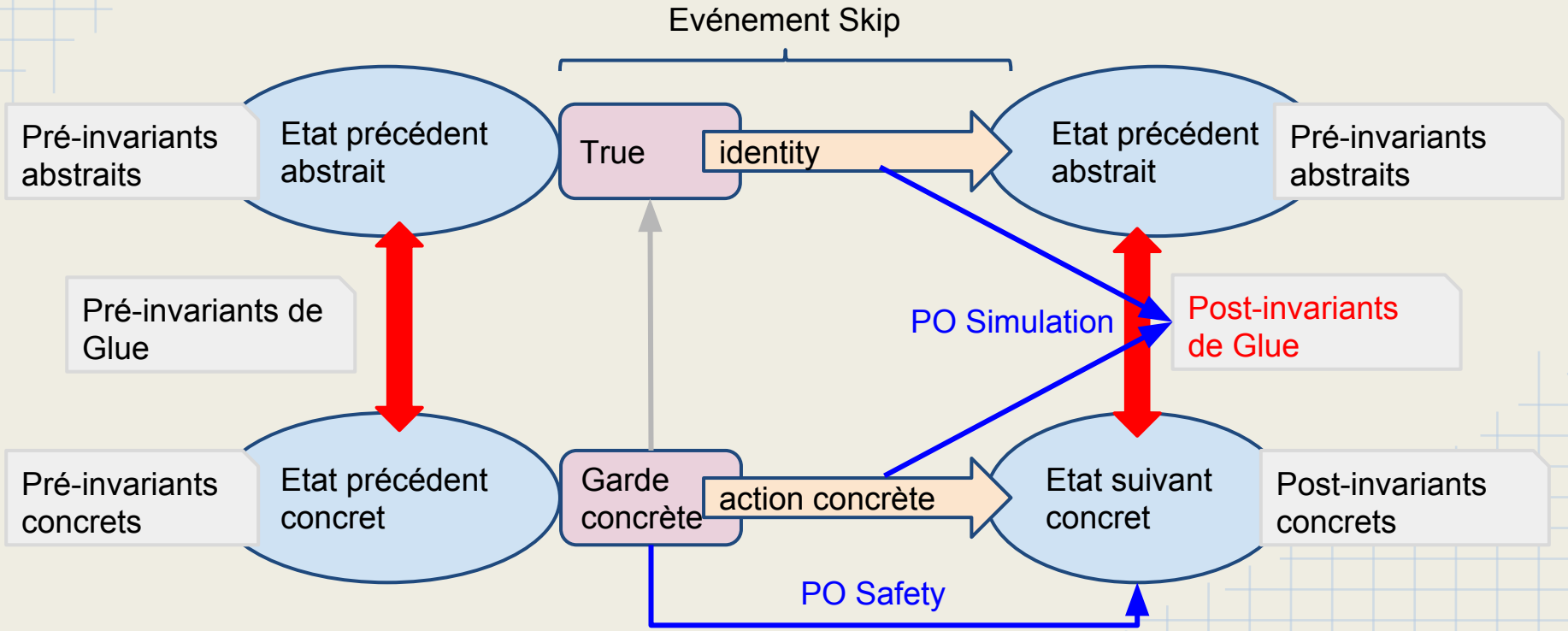


PO Safety en Coq

```
Definition Guard (S:State) (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (S:State) (p1:T1) … (pN:TN) : State := <expr>.

Theorem PO_Safety:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* Garde concrète *)
    -> Guard S p1 … pN
       (* état suivant *)
          let S' := action S p1 … pN
          in (* ====> Post-invariants concrets *)
             Inv_1 S' /\ … /\ Inv_Q S'
Proof.
```

Remarque : pas de changement par rapport aux transitions raffinées.

PO Simulation (transition extension)



PO Simulation en Coq

```
Theorem PO_Simulation:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres concrets *)
  forall p1 : T1, …, forall pN : TN,
  (* paramètres abstraits *)
  forall q1 : U1, …, forall qL : UL,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* Garde concrète *)
    -> Guard S p1 … pN
       (* états suivants (concret/abstrait) *)
       let S' := action S p1 … pN
       in (* ====> Post-invariants de glue *)
            Glue_1 S' AS /\ … /\ Glue_Q S' AS   (* ⇐  remarque : état abstrait inchangé *)
Proof.
```

PO Convergence (extension)

Convergence d'un événement de transition :

- l'événement ne peut s'appliquer indéfiniment dans un état donné
- **Obligatoire** pour garantir l'existence de l'état abstrait.

Preuve de convergence :

- Définition d'un **variant** d'état pour la transition
  - Mesure définie sur un ordre strict bien fondé
    ⇒ exemple : (nat, <)
  - Synthétisé à partir d'un état de la machine

- Preuve de **décroissance stricte du variant** durant la transition
  ⇒ variant *<état suivant>* < variant *<état précédent>*

Convergence PO en Coq

```
Definition Guard (S:State) (p1:T1) … (pN:TN) : Prop := <prop>.
Definition action (S:State) (p1:T1) … (pN:TN) : State := <expr>.
Definition variant (S:State) : nat := <expr>.
Theorem PO_Convergence:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
  (* paramètres *)
  forall p1 : T1, …, forall pN : TN,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* Garde concrète *)
    -> Guard S p1 … pN
       (* état suivant *)
          let S' := action S p1 … pN
          in (* ====> Décroissance stricte du variant *)
              variant S' < variant S.
Proof.
```

Relative Deadlock Freedom

PO optionnelle de la machine :
⇒ dans tout état au moins une transition est possible.

Preuve de Deadlock Freedom:

- Soit un état S quelconque,
  - En supposant les invariants d'état de la machine (et implicitement les axiomes)

  ⇒ montrer que la disjonction des gardes de tous les événements est vraie.

Deadlock Freedom en Coq

```
Theorem PO_Deadlock_Freedom:
  (* états précédents (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_M AS
    (* Pré-invariants de glue *)
    -> Glue_1 S AS  -> … ->  Glue_P S AS
    (* Pré-invariants concrets *)
    -> Inv_1 S  -> … ->  Inv_Q S
    (* ===> Disjonction des garde concrètes *)
    -> Event1.Guard S \/ … \/ EventN.Guard S
Proof.
```

Au prochain numéro ...

1. Actions non-déterministes
   ⇒ plusieurs états suivants possibles pour une même transition

2. Raffinement d'actions non-déterministes
   ⇒ abstrait ≈ - déterminisite
   concret ≈ + déterministe