Modèles Evénementiels

Partie 3 : Non-déterminisme

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

© 2014-15 - Frédéric Peschanski

Non-déterminisme

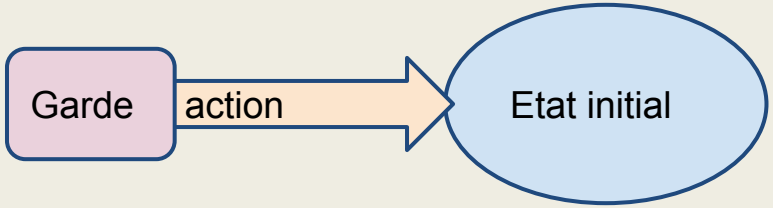
Modèles non-déterministes :

- **Abstraction**
  - on ne connait pas précisément les états possibles, uniquement certaines de leurs propriétés.

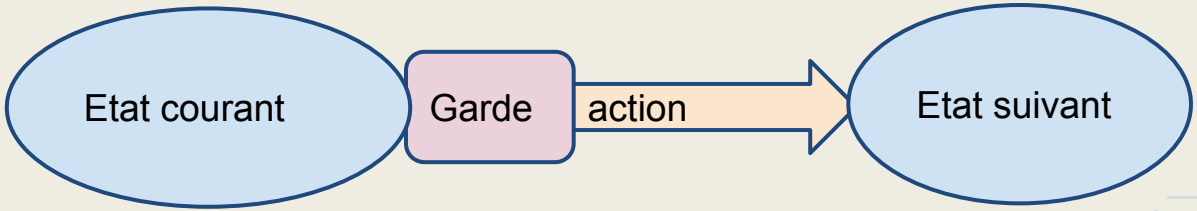
- **Concurrence**
  - les systèmes concurrents sont (souvent) de nature non-déterministe : plusieurs choses peuvent se produire “en même temps” à partir d’un même état, ce qui correspond à un ensemble d’états suivants possibles.

Rappel : événéments

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

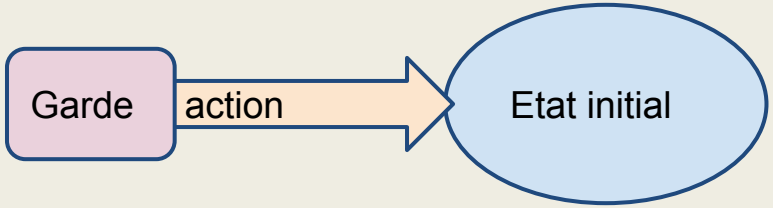


- soit un changement d'état
  ⇒ **événement de transition**

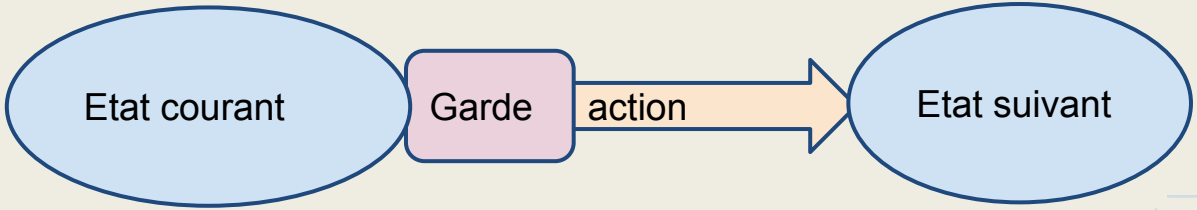


Rappel : événéments

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

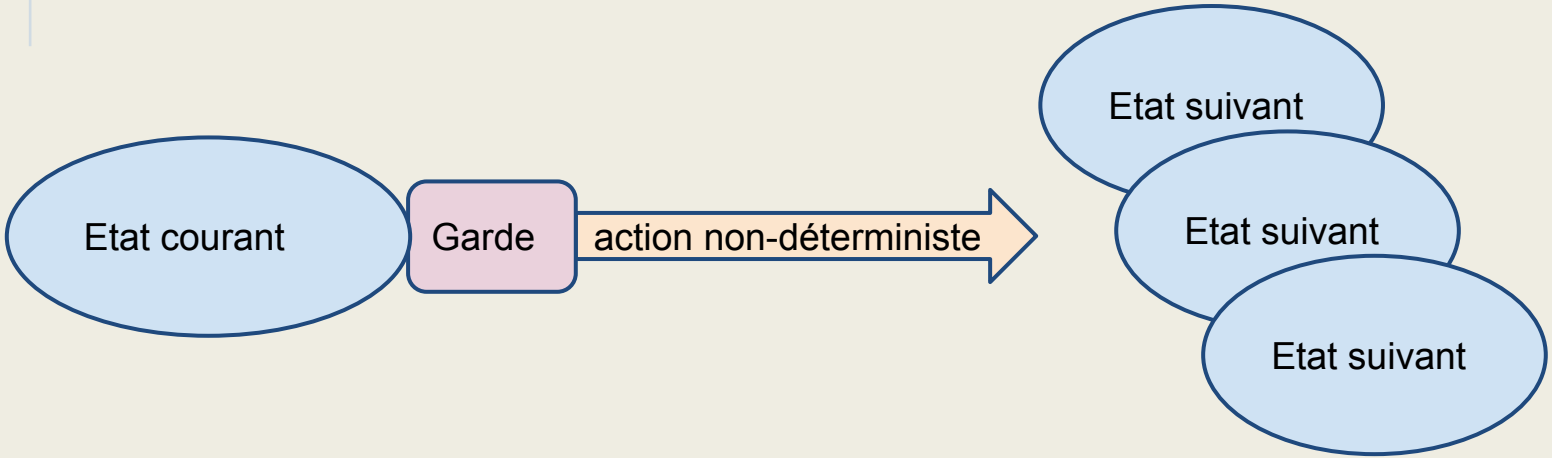


- soit un changement d'état
  ⇒ **événement de transition**



Evénements Déterministes

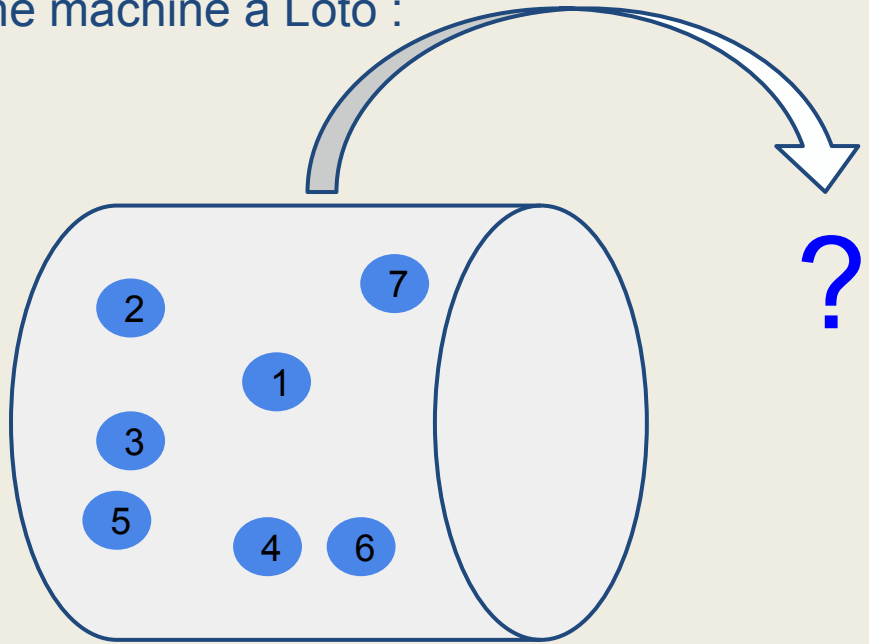
Evénéments non déterministes



Remarque : on étudie uniquement le cas des événements de transition, il est facile d'en déduire les initialisations non-déterministes.

Etude de cas

Une machine à Loto :



Evénéments déterministes en Coq

```
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>.
```

Evénéments non-déterministes

```
Module <Nom de l'événement>.

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

Definition action_Prop_<num> (S : State) <autres paramètres>
                             (S' : State) : Prop :=
      <relation entre l'état courant> S
      <et l'état suivant> S'.

<obligations de preuves (PO)>

End <nom de l'événement>.
```

Obligations de preuve

Evénements déterministes :

- PO Safety
  ⇒ l'état suivant vérifie les invariants
- PO convergence (optionnelle)
  ⇒ décroissance stricte d'un variant

Evénements non-déterministes :

- **PO Feasibility**
  ⇒ il existe au moins un état suivant possible
- PO Safety
  ⇒ les états suivants vérifient les invariants
- PO convergence (optionnelle)
  ⇒ décroissance stricte d'un variant pour tous les états suivants possibles

PO Feasibility en Coq

Principe : montrer qu'il existe au moins un état suivant possible.

```
Definition Guard (S:State) ... : Prop := <prop>.
Definition action_Prop_1 (S:State) ...
                         (S':State) : Prop := <prop>.
Definition action_Prop_N (S:State) ...
                         (S':State) : Prop := <prop>.
Theorem PO_Feasibility:
  (* état courant *)
  forall S : State,
    (* Invariants pre *)
    Inv_1 S  -> … ->  Inv_N S
    -> (* Garde *)
       Guard S
       -> (* états suivants *)
          exists S' : State,
          (* Propriétés de l'action *)
             action_Prop_1 S S'  /\ … /\ action_Prop_M S S'.
Proof.
```

Evénement témoin

Principe de preuve : caractériser un événement témoin

- Même garde que l'événement non-déterministe
- Action témoin déterministe

```
Definition Guard (S:State) ... : Prop := <prop>.

Definition action_Prop_1 (S:State) ...
                         (S':State) : Prop := <prop>.
...
Definition action_Prop_N (S:State) ...
                         (S':State) : Prop := <prop>.

Definition action_witness (S:State) ... : State := <expr>.
```

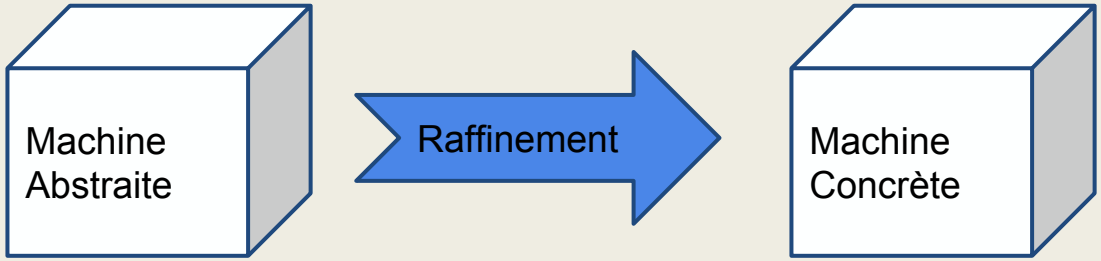
PO Feasibility : témoin

```
Definition action_witness (S:State) ... : State := <expr>.

Theorem PO_Feasibility:
  (* état courant *)
  forall S : State,
    (* Invariants pre *)
    Inv_1 S  -> … ->  Inv_N S
    -> (* Garde *)
       Guard S
       -> (* états suivants *)
          exists S’ : State,
          (* Propriétés de l’action *)
             action_Prop_1 S S’  /\ … /\ action_Prop_M S S’.
Proof.
 intros S HInv_1 … HInv_N HGuard.
 exists (action_witness  S).
 … fin de la preuve ...
```

Safety PO : non-determinisme

```
Definition Guard (S:State) : Prop := <prop>.
Definition action_Prop_1 (S:State) (S':State) : Prop := <expr>.
...
Definition action_Prop_M (S:State) (S':State) : Prop := <expr>.

Theorem PO_Safety:
  (* état courant *)
  forall S : State,
      (* pré-invariants *)
    Inv_1 S  -> … ->  Inv_N S
    -> (* Garde *)
       Guard S p1 … pN
       -> (* états suivants *)
          forall S':State,
            action_Prop_1 S S' -> … ->  action_Prop_M S S'
            (* post-invariants *)
            -> Inv_1 S'  /\ … /\ Inv_M S'.
Proof.
```

Rappel : raffinement



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

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

Raffinements d'événements non-déterministes

Evénement abstrait non-déterministe

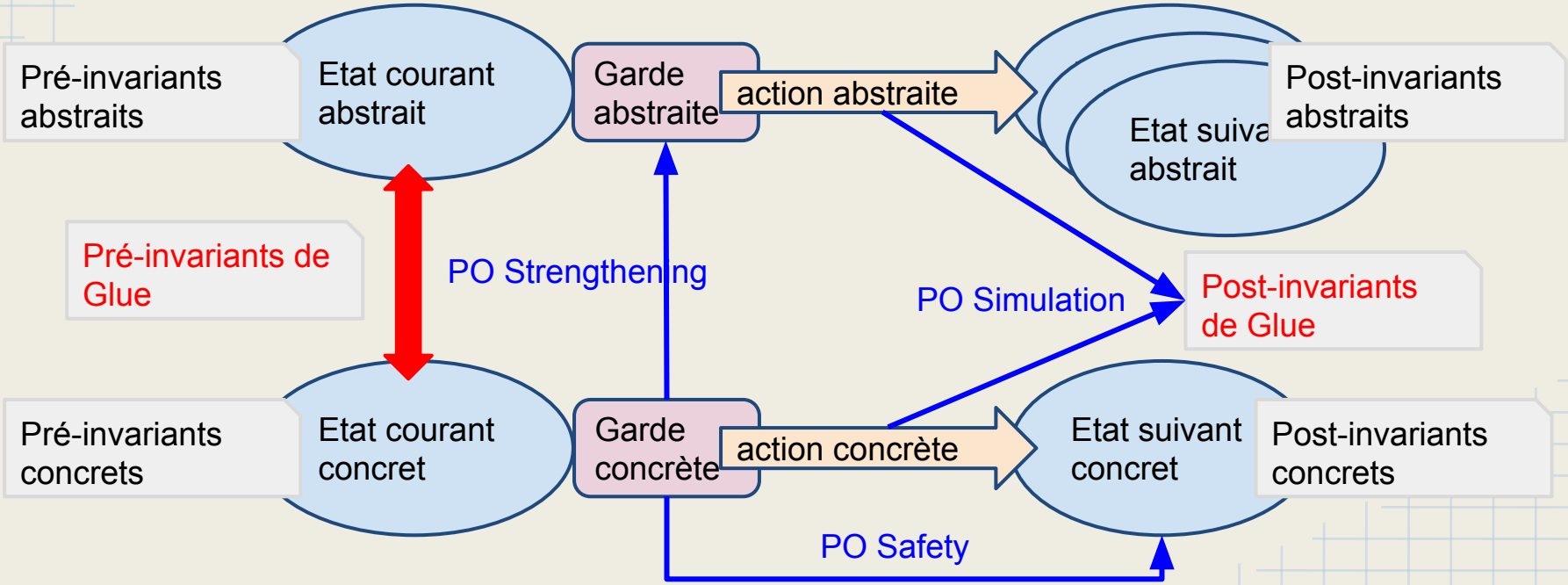
Raffinements possibles :

1. **Evénement concret non-déterministe**
   ⇒ PO Strengthening + Feasibility + Safety + Simulation (+ Convergence)

2. **Evénement concret déterministe**
   ⇒ PO Strengthening + Safety + Simulation (+ Convergence)

Remarque : on n'étudie que le cas déterministe, le cas non-déterministe est laissé en exercice.

Rappel : PO de raffinement



Transitions concrètes déterministes

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>.
```

PO Strengthening

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

Theorem PO_Strengthening:
  (* états courants (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
    -> (* Garde concrète *)
       Guard S
       -> (* ====> Garde abstraite *)
       AbstractMachine.Event.Guard AS.
Proof.
```

Remarque : pas de différence déterministe/non-déterministe.

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 courants (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
    (* Garde concrète *)
    -> Guard S
       (* état suivant *)
          let S' := action S
          in (* ====> Post-invariants concrets *)
             Inv_1 S' /\ … /\ Inv_Q S'
Proof.
```

Remarque : pas de différence déterministe/non-déterministe.

PO Simulation en Coq

```
Theorem PO_Simulation:
  (* états courants (concret/abstrait) *)
  forall S : State, forall AS : AbstractMachine.State,
    (* Pré-invariants abstraits *)
    AbstractMachine.Inv_1 AS  -> … ->  AbstractMachine.Inv_N 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
       in exists AS' : AbstractMachine.State,
          AbstractMachine.Event.action_Prop_1 AS AS'
          /\ … /\ AbstractMachine.Event.action_Prop_M AS AS'
          (* ====> Post-invariants de glue *)
          /\ Glue_1 S' AS' /\ … /\ Glue_Q S' AS'
Proof.
```

Autres obligations

- PO Convergence
  - mêmes principes pour prendre en compte le non-déterminisme.

- PO Deadlock freedom
  - pas de changement (propriété des gardes)

Fin

Résumé :

- Modélisation de machines B
- Raffinement
- Evénements non-déterministes