Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

Djamila Baroudi (1), Philippe Dhaussy (2),

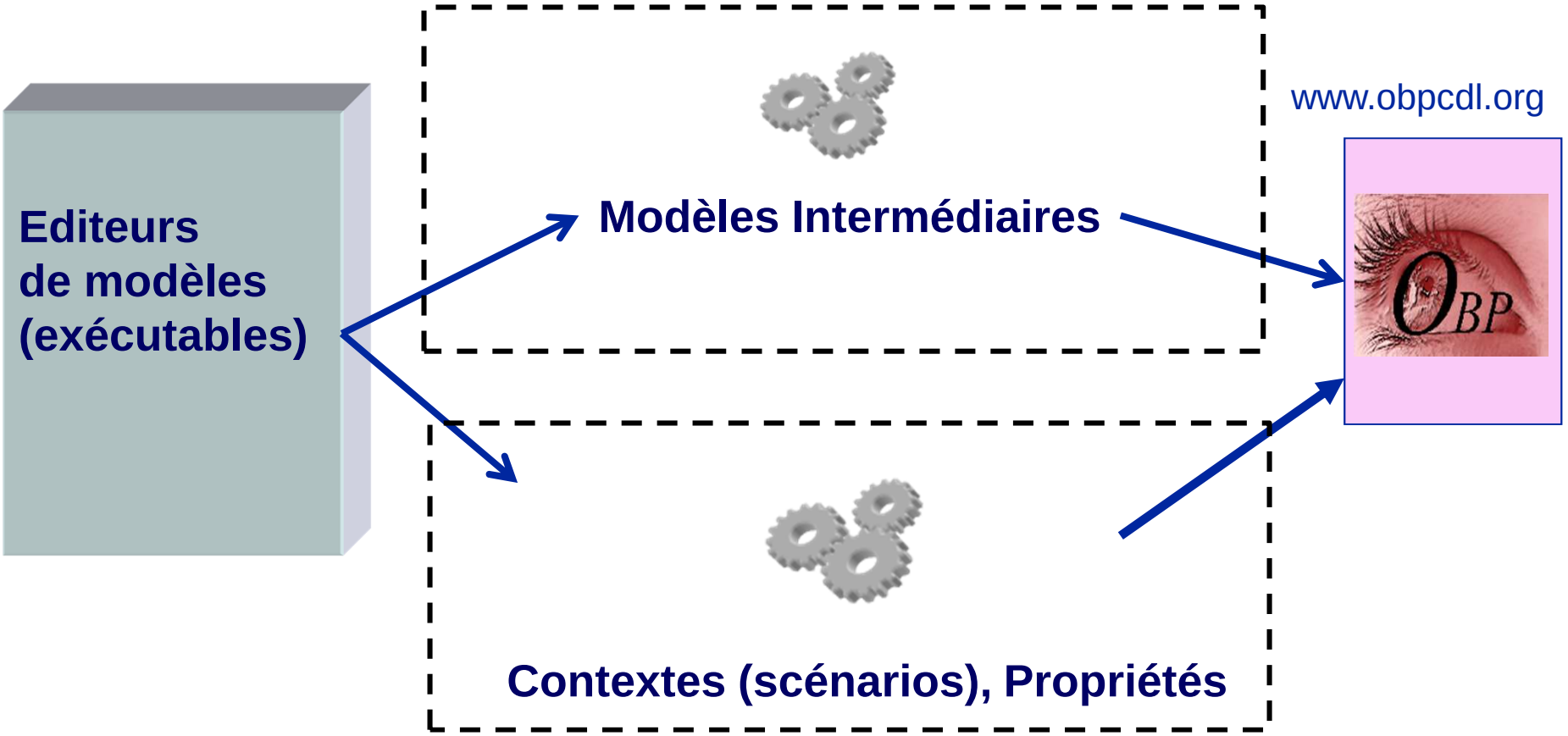
Luka Leroux (2), Nait Bahloul Safia (3)

(1) UMAB, Université de Mostaganem, Algérie, baroudi.djamila@univ-mosta.dz

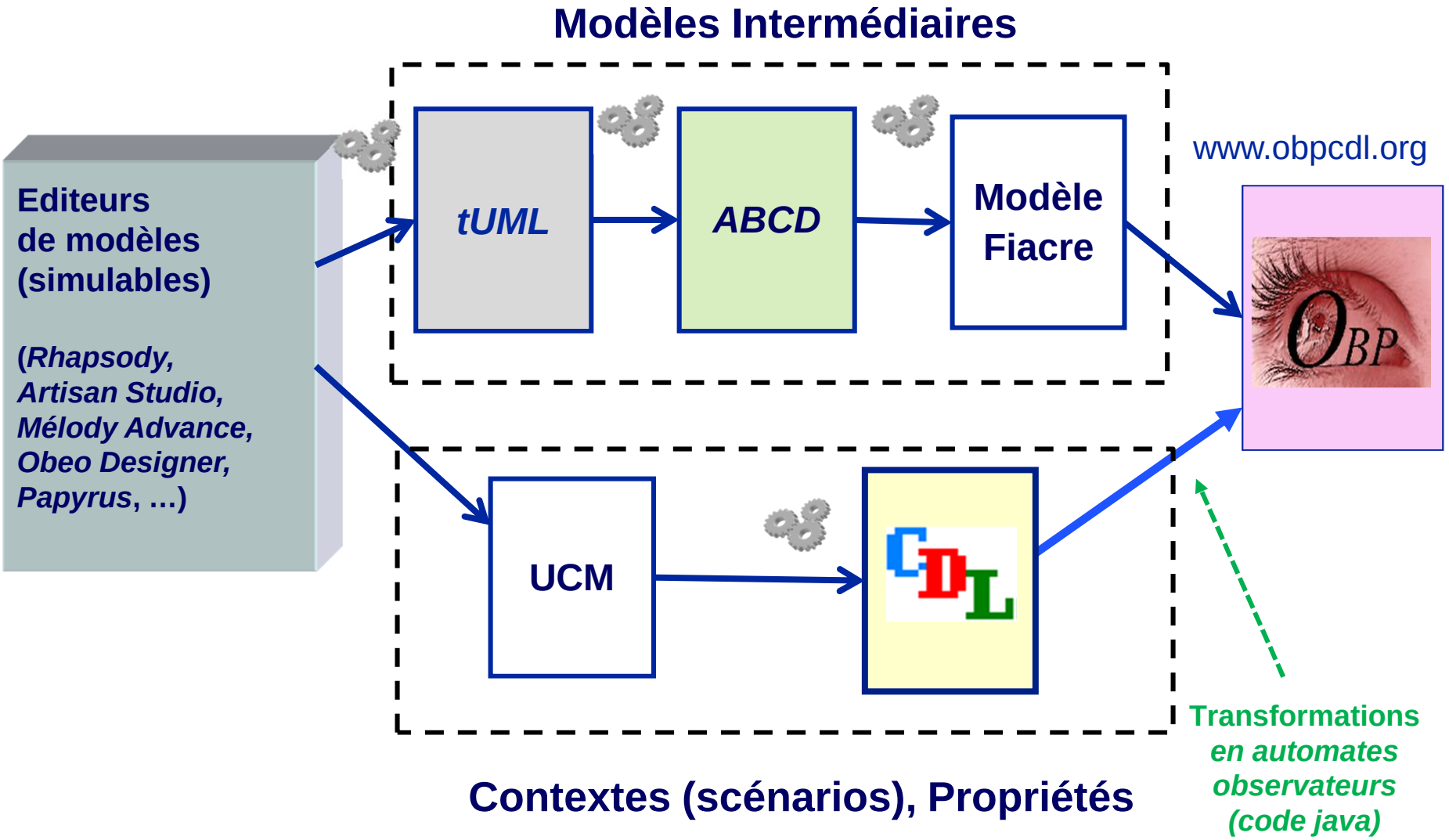
(2) Lab-STICC, UMR CNRS 6285 ENSTA Bretagne, name.firstname@ensta-bretagne.fr

(3) Lab-LITIO, Université Oran 1, Ahmed BENBELLA,Algérie, bahloul.safia@univ-oran.dz

Contexte : Chaîne de transformation de modèles vers explorateur de modèles OBP



Contexte : Chaîne de transformation de modèles vers explorateur de modèles OBP



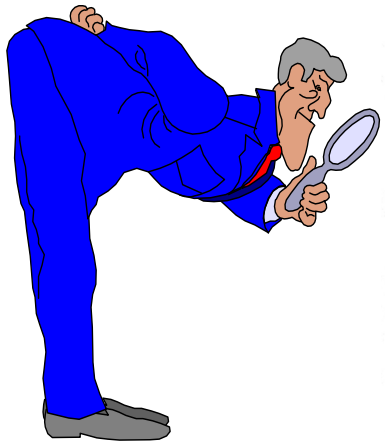
Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

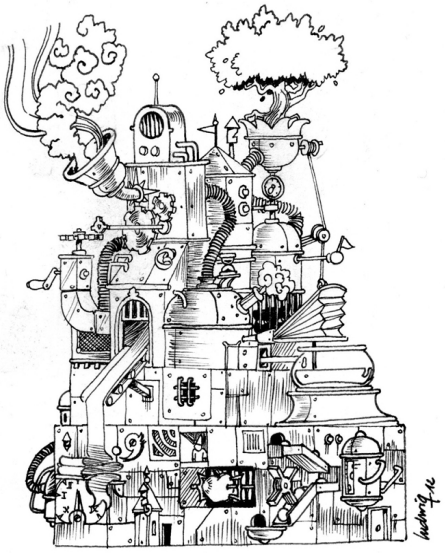
- Contexte et motivation
- Langage CDL pour l'expression des propriétés
- Sémantique des patrons CDL
- Formalisation de la transformation vers les observateurs
- Principe de la preuve de correction des transformations

Type et expression des propriétés CDL (sûreté)

Disposer de primitives élémentaires d'observation, à grain fin

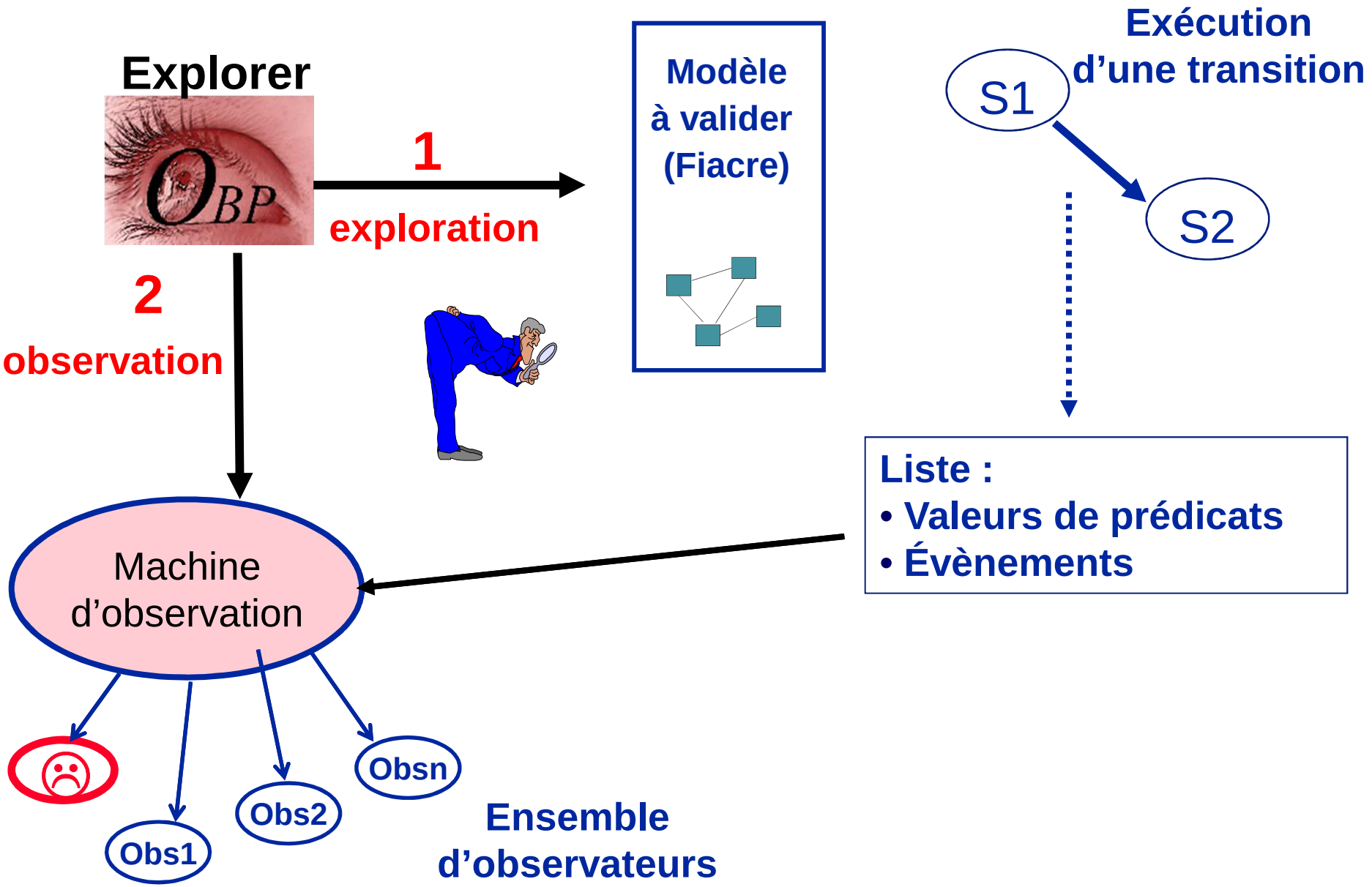
- **Invariants :** expression basée sur des prédicats
  (valeur d'une variable, état d'un processus, état d'une fifo)
- **Observateurs :** expression basée sur des patrons de définition de propriétés :
  Réponse, Précédence, Absence, Existence





S'inspire des patrons de Dwyer…
[Dwyer 99, Smith & Cheng 02, Konrad 05, …]

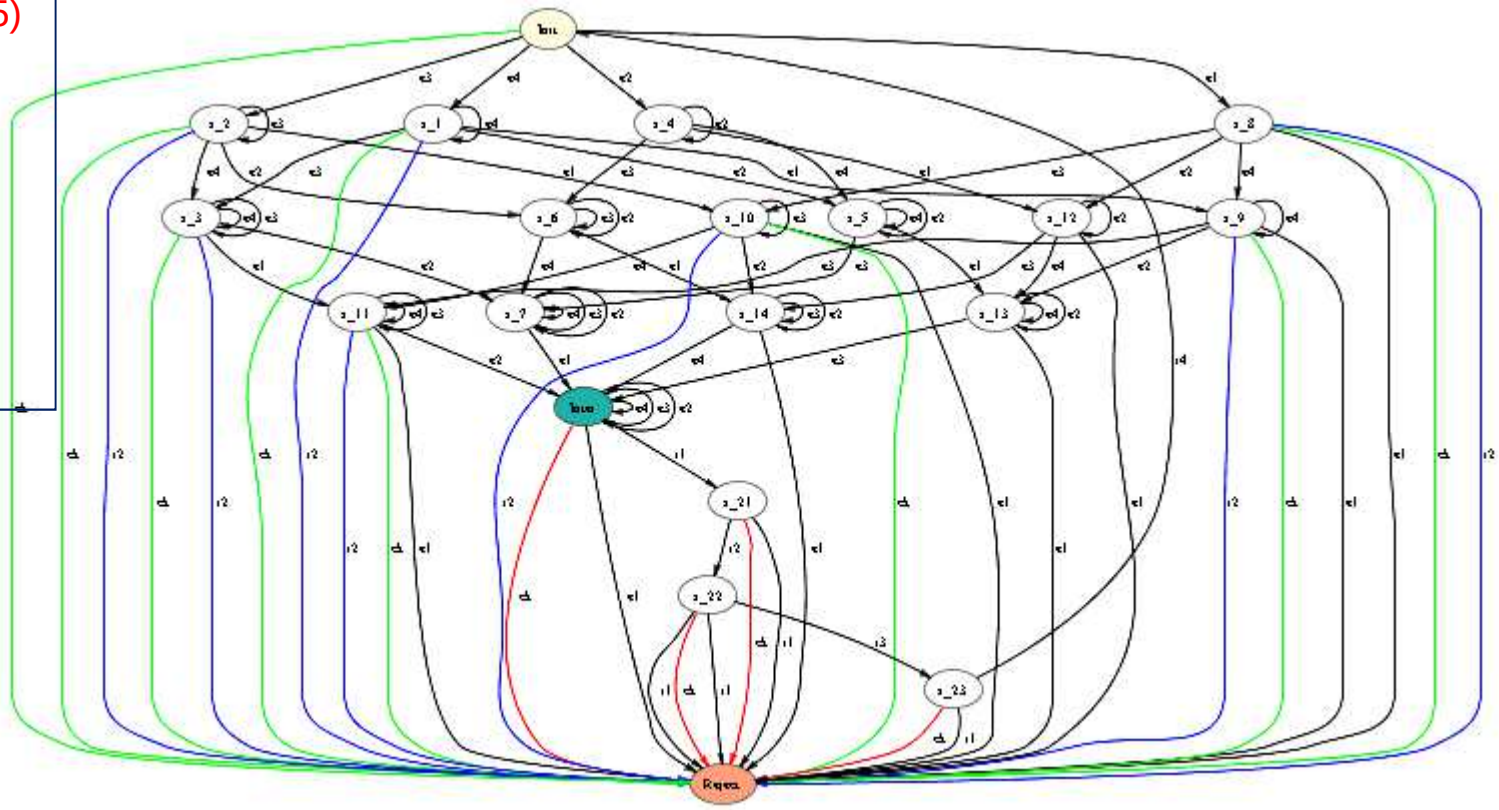
OBP : noyau d'observation (atteignabilité)



Constat : difficulté d'exprimer la sémantique d'un patron de propriété

```
Proprerty P1 is
(all_combined
    [exactly_one (e 1);
     one_or_more (e 2);
     one_or_more (e 3);
     exactly_one (e 4)]
)
eventually (leads_to 0 5)
(all_ordered
    [exactly_one (r 1);
     one_or_more (r 2);
     one_or_more (r 3);
     exactly_one (r 4)]
)
[r 2] cannot_occur (e 2)
(e 2) must occur.
```

Automate décrivant la sémantique de P1



20 nœuds,
91 transitions

Motivations

1. Formaliser la sémantique des patrons CDL
2. Formaliser les règles de transformation vers les observateurs
3. Assurer la correction des transformations vers les automates observateurs
4. Faciliter la définition de variantes sémantiques

→ Mise en œuvre de COQ pour la formalisation et les preuves

Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

- Contexte et motivation
- Langage CDL pour l'expression des propriétés
- Sémantique des patrons CDL
- Formalisation de la transformation vers les observateurs
- Principe de la preuve de correction des transformations

Un exemple d'observateur CDL (de type Réponse)

Property P is

(ALL Ordered
exactly one occurrence of S_CP_hasReachState_Init
exactly one occurrence of login1
)
eventually leads-to [0..maxD_log]
(AN
one or more occurrence of ackLog (id)
)

S_CP_hasReachState_Init **may never occurs**
login1 **may never occurs**
one of ackLog (id) **cannot occur before** login1.

Un exemple d'observateur CDL (de type Réponse)

Property P is

(ALL Ordered
exactly one occurrence of S_CP_hasReachState_Init
exactly one occurrence of login1
)
eventually leads-to [0..maxD_log]
(AN
one or more occurrence of ackLog (id)
)

S_CP_hasReachState_Init **may never occurs**
login1 **may never occurs**
one of ackLog (id) **cannot occur before** login1.

Évènements

Un exemple d'observateur CDL (de type Réponse)

Property P is

(ALL Ordered
    exactly one occurrence of S_CP_hasReachState_Init
    exactly one occurrence of login1
)
eventually **leads-to** [0..maxD_log]
(AN
    one or more occurrence of ackLog (id)
)

S_CP_hasReachState_Init **may never occurs**
login1 **may never occurs**
one of ackLog (id) **cannot occur before** login1.

Évènements
options

Un exemple d'observateur CDL (de type Réponse)

Property P is

(ALL Ordered
  exactly one occurrence of S_CP_hasReachState_Init
  exactly one occurrence of login1
)
eventually **leads-to** [0..maxD_log]
(AN
  one or more occurrence of ackLog (id)
)

S_CP_hasReachState_Init **may never occurs**
login1 **may never occurs**
one of ackLog (id) **cannot occur before** login1.

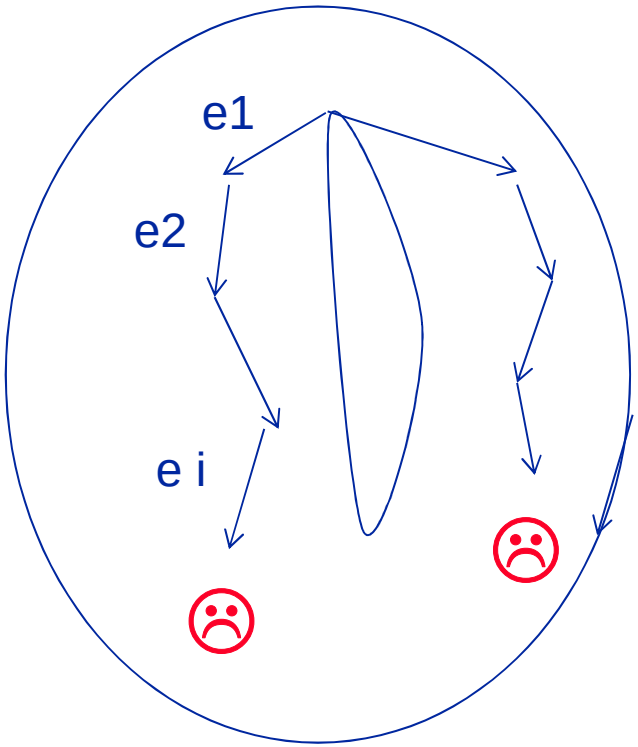
Évènements
Options
Occurrences

Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

- Contexte et motivation
- Langage CDL pour l'expression des propriétés
- Sémantique des patrons CDL
- Formalisation de la transformation vers les observateurs
- Principe de la preuve de correction des transformations

Sémantique des propriétés CDL (sûreté)

Soit un patron T : Sémantique définie par une relation qui spécifie les contre exemples à la propriété exprimée par T



Par exemple : *notmodels*

telle que :

- ***path* :** ensemble des chemins (liste d'évènements)
- ***T notmodels path* :** *path* est rejeté par le patron *T*

(path est contre exemple de la propriété exprimée par T)

Transformation des observateurs CDL

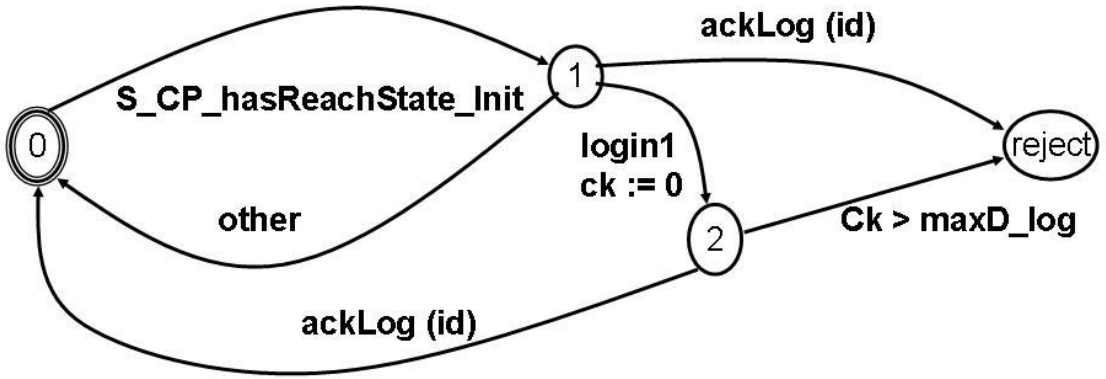
Property P is

ALL Ordered
exactly one occurrence of S_CP_hasReachState_Init
exactly one occurrence of login1
end
eventually leads-to [0..maxD_log]
AN
one or more occurrence of ackLog (id)
end

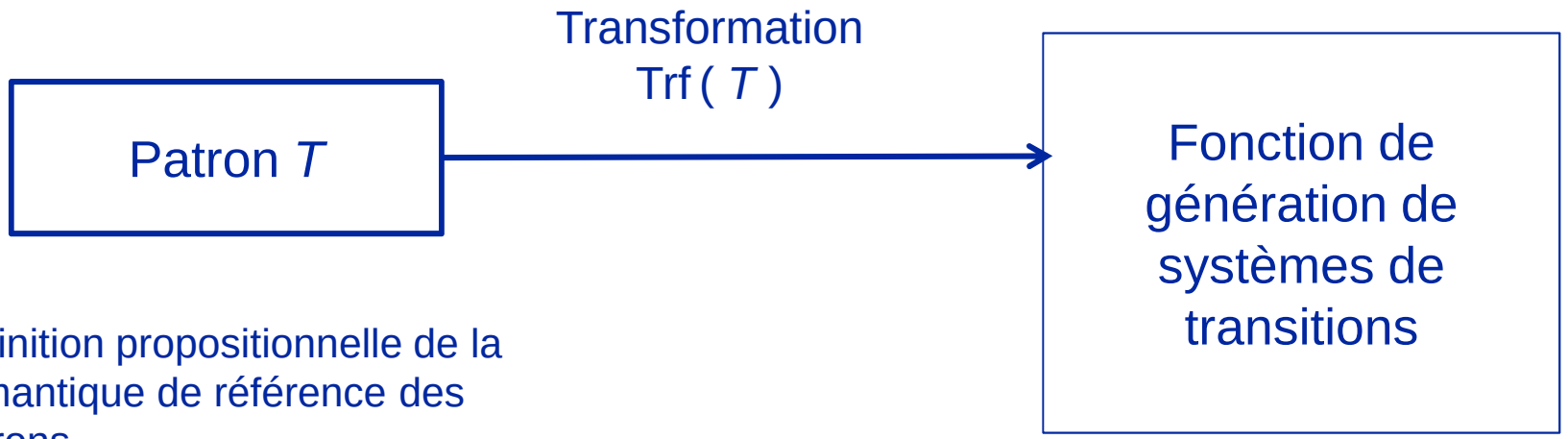
S_CP_hasReachState_Init **may never occurs**
login1 **may never occurs**
one of ackLog (id) **cannot occur before** login1.

lts := Trf (T)

Transformation vers un système de transitions



Transformations

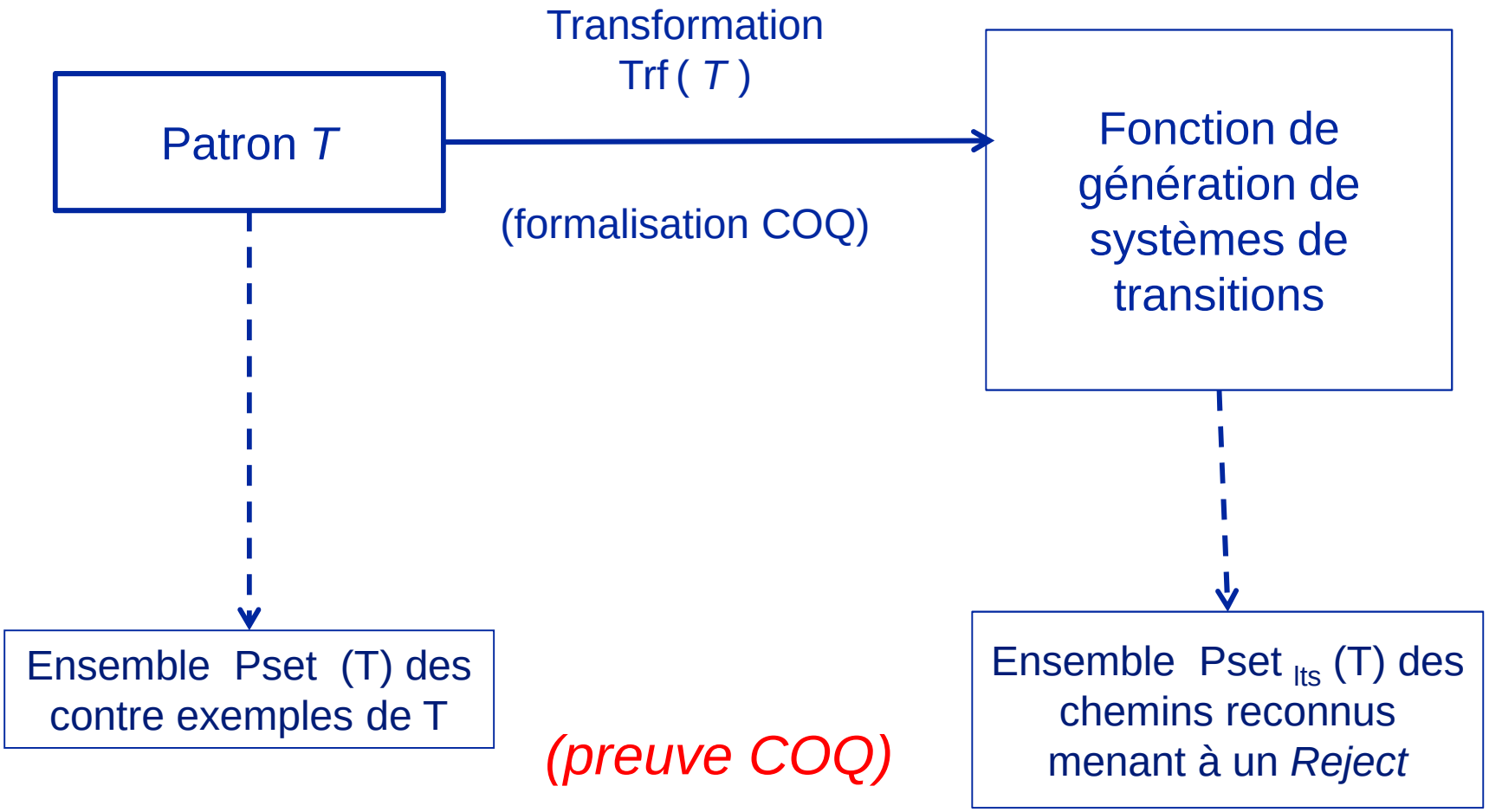


Définition propositionnelle de la sémantique de référence des patrons
(Relation "notmodels«)

- ***path*** **:** ensemble des chemins (liste d'évènements)
- ***T notmodels path*** **:** *path* est rejeté par le patron *T*

- $\boldsymbol{path_{lts}}$ **:** ensemble de transitions (liste de transitions)
- ***(Trf T)*** $\boldsymbol{notmodels_{lts}}$ $\boldsymbol{path_{lts}}$ **:** $path_{lts}$ aboutit à un état ***Reject***

Corrrection des transformations



Relation à démontrer : *forall p : path,*
T notmodels p ⇔ (Trf T) notmodels$_{lts}$ (Trf p)

Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

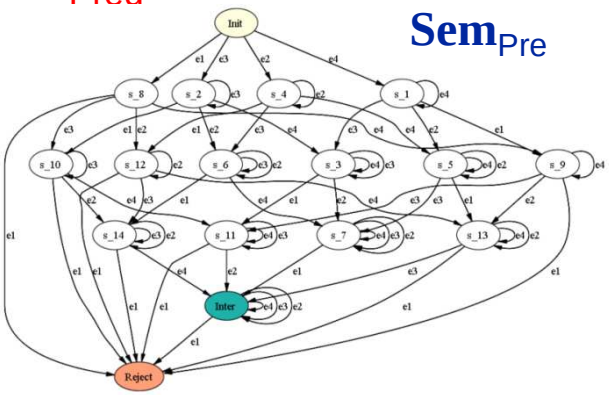
- Contexte et motivation
- Langage CDL pour l'expression des propriétés
- Sémantique des patrons CDL
- Formalisation de la transformation vers les observateurs
- Principe de la preuve de correction des transformations

$Sem_{P1} = Sem_{Pre} \times Sem_{Post} \times Sem_{Imm} \times Sem_{Nullity} \times Sem_{Pred}$

```
(all_combined
    [exactly_one (e 1);
     one_or_more (e 2);
     one_or_more (e 3);
     one_or_more (e 4)]
)
```

Pre clause (arrity *all combined*)

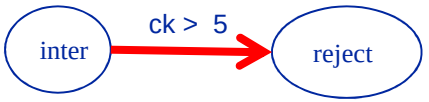
Sem_{Pre}



eventually (leads_to 0 5)

Immediacy

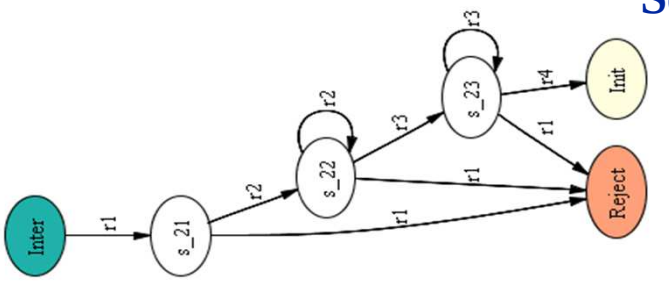
Sem_{Imm}




```
(all_ordered
    [ exactly_one (r 1);
     one_or_more (r 2);
     one_or_more (r 3);
     exactly_one (r 4)]
)
```

Post clause (arrity *all combined*)

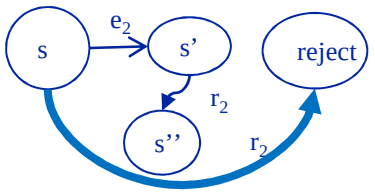
Sem_{Post}



[r 2] cannot_occur (e 2)

Precedency

Sem_{Pre} d



(e 2) must occur .

Nullity

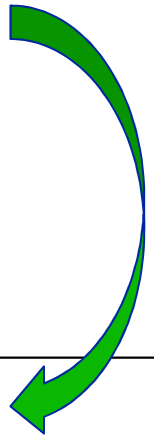
$Sem_{Nullity}$

Sémantique des patrons CDL (transformationnelle)

Pour chaque type de patron, ses combinaisons et options :

Type P = Response, occ_descript (occ_list_pre) = an, loc (occ_list_pre) = Pre, occ_descript (occ_list_post) = an, loc (occ_list_post) = Post, immediacy, nullity (list), precedency (list, ev)
an_pre_occ_list (occ_list_pre), an_post_occ_list (occ_list_post) Immediacy, nullity (list), precedency (list, ev)

Règle d'inférence



Code Gallina :

```
Fixpoint   property_response (pre : occurrence_expression)
                             (Imm :  immediacy)  (tps : list transition)
                             (post : occurrence_expression)
                             (nullity :  l_ev :  list event )
                             (l_ev :  list event)  (preced : precedency) (ev : event ) : list transition :=

match pre, post with …
```

Règle de transformation : All_ordered_pre

$$allOrdered\_pre\_occ\_list\ (occ\_list) = \quad trans\_allOrdered\ (occ\_list,\ Init,\ Inter)$$

$$trans\_allOrdered\ (occ\_list,\ begin,\ end) =$$

$$\begin{array}{ll}
s\ =\ begin; & \\
\forall occ_i \in occ_list, & \\
if\ (occ_i\ =\ last\ (occ_list)) & if\ (arity\ (occ_i)\ =\ exactly_one) \\
 & \quad (s,\ identif\ (occ_i),\ end) \\
 & if\ (arity\ (occ_i)\ =\ one_or_more\ and\ end\ =\ Init) \\
 & \quad (s,\ identif\ (occ_i),\ end) \\
 & if\ (arity\ (occ_i)\ =\ one_or_more\ and\ end \neq\ Init) \\
 & \quad loop\ (s,\ identif\ (occ_i),\ end) \\
else & next\ =\ nextState(s); \\
 & if\ (arity\ (occ_i)\ =\ exactly_one) \\
 & \quad (s,\ identif\ (occ_i),\ next); \\
 & if\ (arity\ (occ_i)\ =\ one_or_more) \\
 & \quad loop\ (s,\ identif\ (occ_i),\ next); \\
 & s\ =\ next;
\end{array}$$

Règle de transformation : option precedency

Option Precedency : Règle *precedency*

La règle *an_precedency* permet de construire l'ensemble des transitions t entre l'état *Init* et l'état *Reject* en cas de $Precedency = Cannot$. Ces transitions sont associées aux occurences de la partie post.

$$\frac{\begin{array}{c} precedency\ (P) = Cannot, \\ \forall occ_i \in occ\_list,\ \ loc\ (occ_i) = Post \end{array}}{\forall\ occ_i \in occ\_list,\ t_i = (Init,\ identif\ (occ_i),\ Reject)}\ [\text{precedency}]$$

Si $Precedency = May$, aucune transition supplémentaire n'est créee.

Code Gallina :

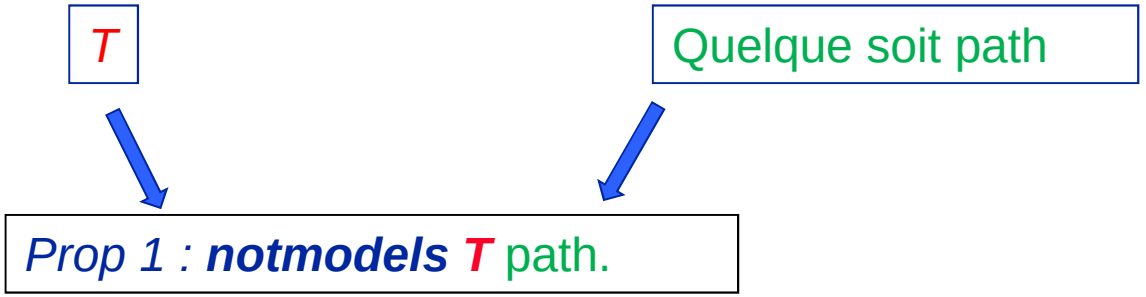
```
Fixpoint Cannot_sem  (aList : list event) (e: event) : list transition  :=
match aList  with
            | nil => nil
            | ev :: list2 =>  (Init, ev, Reject) :: (Cannot_sem (list2) (e))
end.
```

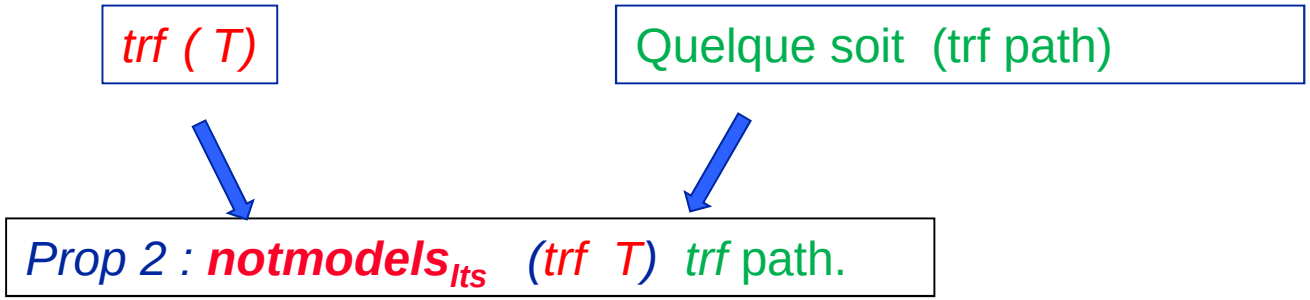
Formalisation d'une Approche Compositionnelle de Patrons de Propriétés

- Contexte et motivation
- Langage CDL pour l'expression des propriétés
- Sémantique des patrons CDL
- Formalisation de la transformation vers les observateurs
- Principe de la preuve de correction des transformations

Principes de la preuve de correction des transformations

Pour un patron T de type t :

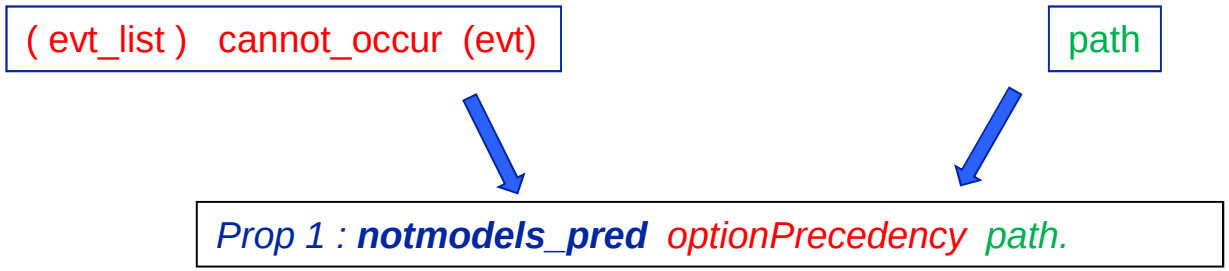




La preuve de correction de trf (pour le patron T) est : Prop 1 ⇔ Prop 2.

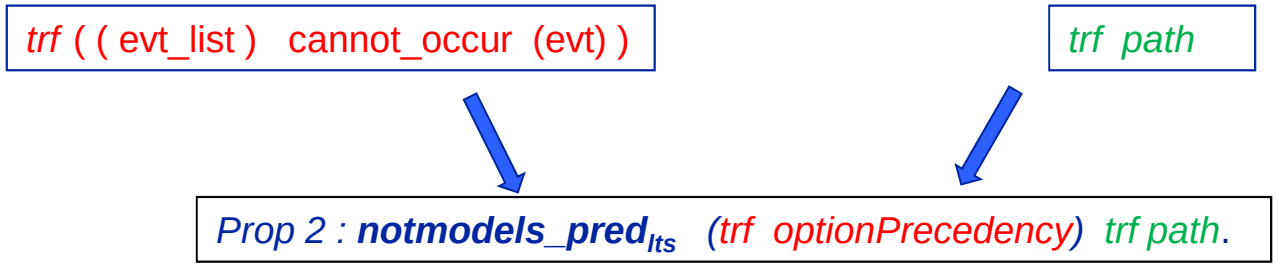
Exemple de preuve (fragment d'un patron : option precedency)

Soit la définition de l'option *precedency* :



Pour montrer si le chemin *path* mène à un échec, il faut montrer la véracité de cette proposition.

De même pour le résultat de la transformation de ce fragment :



La preuve de correction de trf (pour ce fragment) est : Prop 1 < = = > Prop 2.

Définition propositionnelle du fragment de pattern : option precedency

Proposition : *notmodels_pred* :

Evalue si un chemin (*path*) est « rejeté » par la définition fragment *precedency* d'un patron

Inductive *notmodels_pred* (preDecl : precedencyDecl) (p : path) : Prop :=

| **headPR** :
Pour une liste (postList : list event) (preEvt postEvt : event) (head tail : path),

Si postEvt est dans postList
et preDecl est de la forme : Precedency postList preEvt
et postEvt est dans p de la forme (head :: postEvt :: tail)
et preEvt n'est pas dans head
alors on a un postEvt avant le preEvt : p est rejeté

| **consPR** :
Pour une liste (postList : list event) (preEvt postEvt : event) (head tail : path),

Si postEvt est dans postList
et preDecl est de la forme : Precedency postList preEvt
et postEvt est dans p de la forme (head :: postEvt :: tail
et on rejette tail (recursion)
alors : p est rejeté quelquesoit head.

Codage de la proposition en Coq

```
Inductive notmodels_pred  (preDecl :
                precedencyDecl)  (p : path) : Prop :=

| headPR : forall     (rList : list event) (eEvt rEvt : event)
                        (head tail : path),

                In postEvt   postList  ->

                preDecl  = Precedency  postList preEvt ->

                p = head ++ [postEvt] ++ tail  ->

                not (In preEvt head)  ->

                 notmodels_pred   preDecl   p

| consPR : forall     (rList : list event) (eEvt  rEvt : event)
                        (head  tail : path),

                In postEvt postList->

                 preDecl = Precedency  rList  eEvt ->

                 p = head ++ [postEvt] ++ tail  ->

                 precedencyReject   preDecl   tail ->

                 notmodels_pred  preDecl p.
```

Si postEvt est dans postList...

et preDecl est de la forme Precedency postList preEvt.

et postEvt est dans p de la forme (head :: postEvt :: tail)

et preEvt n'est pas dans head ...

alors on a un postEvt avant le preEvt : p est rejeté

Si postEvt est dans postList...

et preDecl est de la forme :
Precedency postList preEvt

et postEvt est dans p de la forme (head :: postEvt :: tail

et on rejette tail (recursion)

alors on rejette l'ensemble du chemin, qu'importe head

Test de la proposition pour un chemin spécifique [e3; e1].

Soit la définition de l'option precedency : e2, e3 ne peuvent pas arriver avant e1

(e2, e3) cannot_occur (e1)

path : [e3; e1]

Prop1 : notmodels_pred optionPrecedency [e3; e1].

```
Example preuve_Prop1: :

        notmodels_pred optionPrecedency   path.
Proof.
        unfold optionPrecedency ;  unfold  path.

        apply headPR with
                (postList  := [e2; e3])
                (preEvt := e1)
                (postEvt := e3)
                (head := [])
                (tail := [e1]).

        simpl. right; left. reflexivity.
        reflexivity.
        simpl. reflexivity.
        unfold not; intro Hnot.
        inversion Hnot.
Qed.
```

Première étape, remplacer les références par ce qu'elles représentent.

Dans le chemin [e3; e1], on a un e3 avant un e1.
On doit donc pouvoir utiliser headPR.

Coq nous demande de prouver les 4 prérequis à headPR.

Conclusion

- Formalisation de la transformation (code Galina) patrons vers automates observateur
- Début de formalisation propositionnelle de la sémantique des patterns
  Exemple : *notmodels_pred*
- Début d'élaboration des preuves
  Modifier l'implantation Gallina des règles de transformation
- Définition compositionnelle de la sémantique (permet de raisonner sur des variantes)
- Roadmap :
  - Formalisation de la sémantique de l'ensemble des patterns (combinaisons et options)
  - Preuves de la transformation à réaliser pour tous les patterns