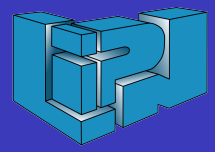


Mémoire de projet de fin d'études

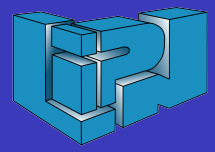
Spécification et vérification des réseaux de Petri temporels en Coq

Amal Chamakh

Tuteurs
Mme Micaela Mayero - M. Kais Klai

Laboratoire d'Informatique de Paris Nord
Université Paris 13

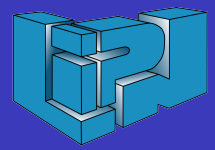
le 22 Janvier 2015



Contexte et Problématique

Prolifération de systèmes pour :

- Surveiller les automatismes des transports terrestres
- Piloter des avions et des fusées
- Contrôler des centrales nucléaires
- Transmettre des données confidentielles

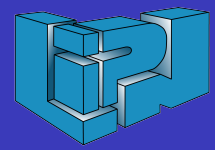


Contexte et Problématique

Prolifération de systèmes pour :

- Surveiller les automatismes des transports terrestres
- Piloter des avions et des fusées
- Contrôler des centrales nucléaires
- Transmettre des données confidentielles

$\rightarrow$ Erreur = désastres humains, financiers et environnementaux



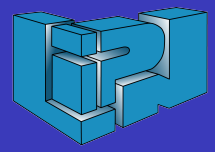
Contexte et Problématique

Prolifération de systèmes pour :

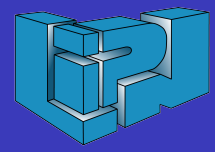
- Surveiller les automatismes des transports terrestres
- Piloter des avions et des fusées
- Contrôler des centrales nucléaires
- Transmettre des données confidentielles

$\rightarrow$ Erreur = désastres humains, financiers et environnementaux

Une solution : la vérification formelle.



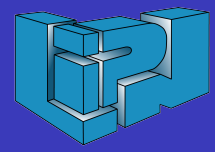
La vérification formelle : prouver mécaniquement que le programme vérifie une propriété



La vérification formelle : prouver mécaniquement que le programme vérifie une propriété

Model checking

Preuve Formelle

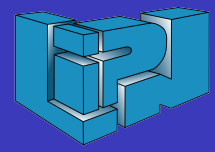


La vérification formelle : prouver mécaniquement que le programme vérifie une propriété

Model checking

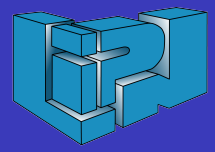
- automatique
- contre exemple fourni par un model checker
- décidable sur des systèmes finis $\rightarrow$ problème de l'explosion combinatoire

Preuve Formelle



La vérification formelle : prouver mécaniquement que le programme vérifie une propriété

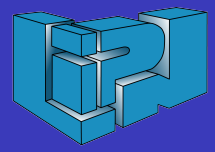
Model checking	Preuve Formelle
• automatique	• non automatique
• contre exemple fourni par un model checker	• fastidieuse
• décidable sur des systèmes finis $\rightarrow$ problème de l'explosion combinatoire	• applicable sur des systèmes paramétrés, infinis



La vérification formelle : prouver mécaniquement que le programme vérifie une propriété

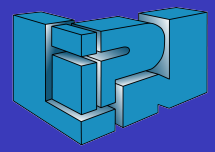
Model checking	Preuve Formelle
• automatique	• non automatique
• contre exemple fourni par un model checker	• fastidieuse
• décidable sur des systèmes finis $\rightarrow$ problème de l'explosion combinatoire	• applicable sur des systèmes paramétrés, infinis

$\Rightarrow$ Combiner model checking et preuve formelle.

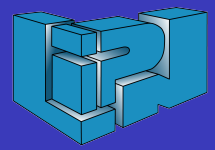


Objectifs

- Combiner preuve formelle et model checking de systèmes temporisés
- Validations qualitative et quantitative des systèmes temporisés
- Modélisation par des réseaux de Petri temporels



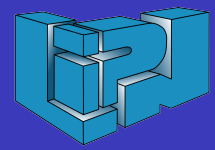
Plan



Réseaux de Petri temporels

Réseau de Petri P/T

Un outil de modélisation en phase de conception de systèmes.

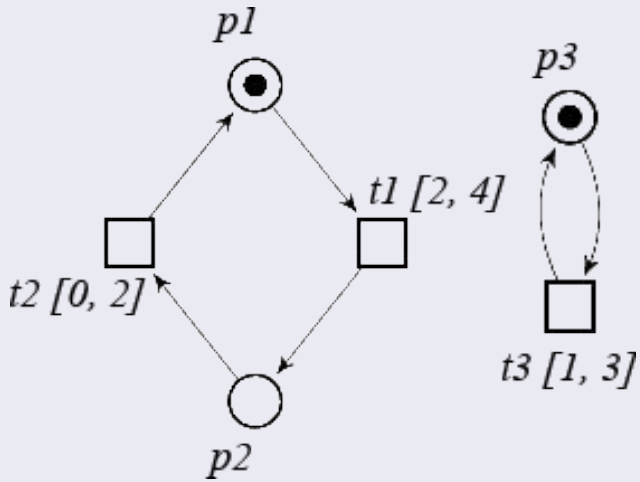


Réseaux de Petri temporels

Réseau de Petri P/T

Un outil de modélisation en phase de conception de systèmes.

Réseau de Petri temporel (TPN)

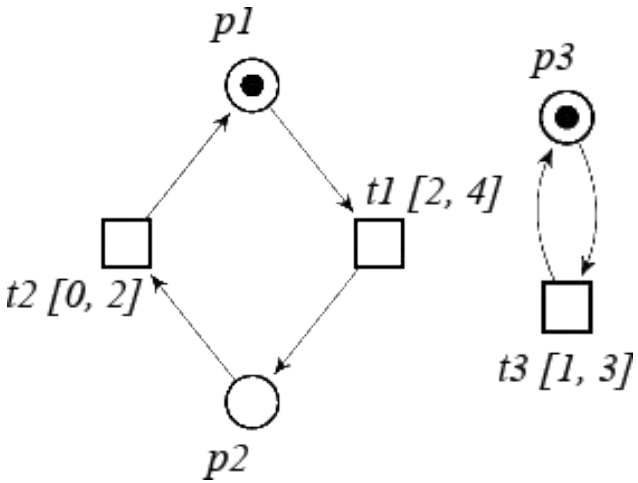


Extension des RdPs P/T :

- Intervalle de temps (statique)
- Horloge v

pour chaque transition

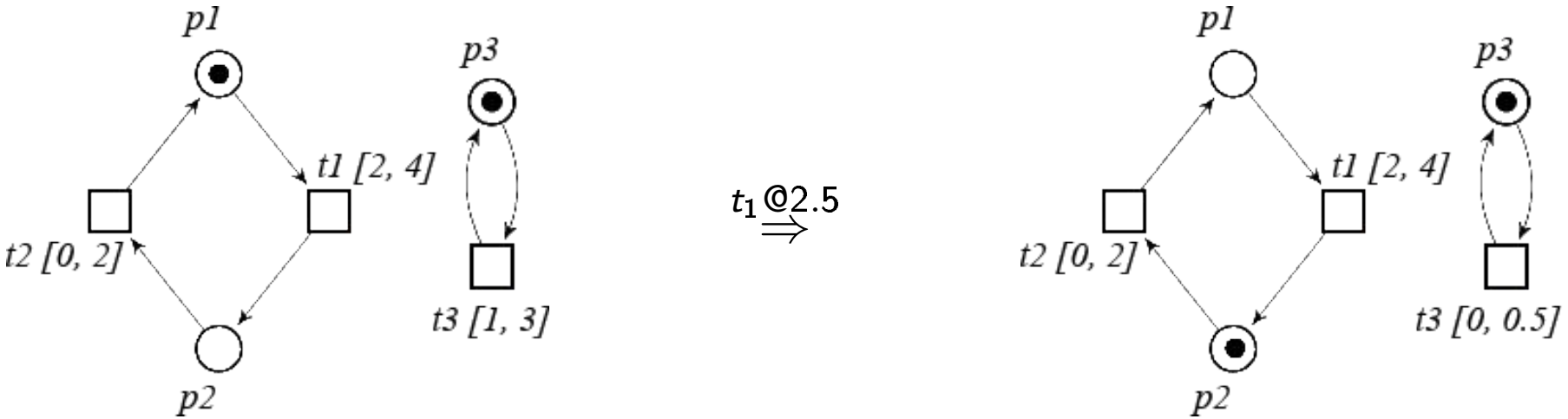
Réseaux de Petri temporels



$$Etat_0 \begin{cases} m_0=(1,0,1) \\ v_0=(0,\perp,0) \end{cases} \xrightarrow{0.5} Etat_1 \begin{cases} (1,0,1) \\ (0.5,\perp,0.5) \end{cases} \xrightarrow{2} Etat_2 \begin{cases} (1,0,1) \\ (2.5,\perp,2.5) \end{cases} \xrightarrow{t_1} Etat_3 \begin{cases} (0,1,1) \\ (\perp,0,2.5) \end{cases}$$

$$\xrightarrow{0.5} Etat_4 \begin{cases} (0,1,1) \\ (\perp,0.5,3) \end{cases} \xrightarrow{t_3} Etat_5 \begin{cases} (0,1,1) \\ (\perp,0.5,0) \end{cases} \xrightarrow{1.5} Etat_6 \begin{cases} (0,1,1) \\ (\perp,2,1.5) \end{cases} \cdots$$

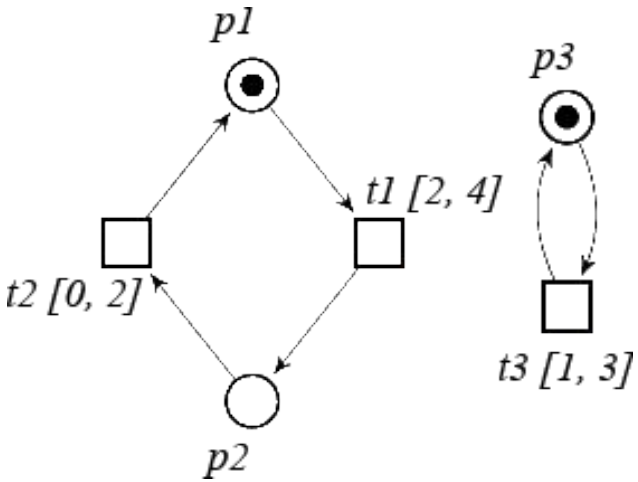
Réseaux de Petri temporels



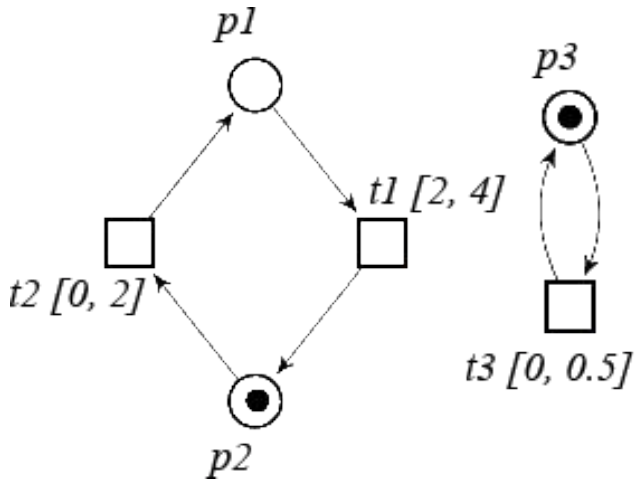
$$Etat_0 \begin{cases} m_0=(1,0,1) \\ v_0=(0,\perp,0) \end{cases} \xrightarrow{0.5} Etat_1 \begin{cases} (1,0,1) \\ (0.5,\perp,0.5) \end{cases} \xrightarrow{2} Etat_2 \begin{cases} (1,0,1) \\ (2.5,\perp,2.5) \end{cases} \xrightarrow{t_1} Etat_3 \begin{cases} (0,1,1) \\ (\perp,0,2.5) \end{cases}$$

$$\xrightarrow{0.5} Etat_4 \begin{cases} (0,1,1) \\ (\perp,0.5,3) \end{cases} \xrightarrow{t_3} Etat_5 \begin{cases} (0,1,1) \\ (\perp,0.5,0) \end{cases} \xrightarrow{1.5} Etat_6 \begin{cases} (0,1,1) \\ (\perp,2,1.5) \end{cases} \cdots$$

Réseaux de Petri temporels



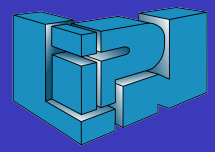
$t_1 @ 2.5 \Rightarrow$



$$Etat_0 \begin{cases} m_0=(1,0,1) \\ v_0=(0,\perp,0) \end{cases} \xrightarrow{0.5} Etat_1 \begin{cases} (1,0,1) \\ (0.5,\perp,0.5) \end{cases} \xrightarrow{2} Etat_2 \begin{cases} (1,0,1) \\ (2.5,\perp,2.5) \end{cases} \xrightarrow{t_1} Etat_3 \begin{cases} (0,1,1) \\ (\perp,0,2.5) \end{cases}$$

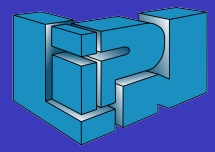
$$\xrightarrow{0.5} Etat_4 \begin{cases} (0,1,1) \\ (\perp,0.5,3) \end{cases} \xrightarrow{t_3} Etat_5 \begin{cases} (0,1,1) \\ (\perp,0.5,0) \end{cases} \xrightarrow{1.5} Etat_6 \begin{cases} (0,1,1) \\ (\perp,2,1.5) \end{cases} \cdots$$

$\Rightarrow$ Espace d'états infini



Preuves formelles

Coq : assistant de preuve basé sur le langage Gallina
- types usuels des langages de programmation
- exprimer des assertions sur les expressions

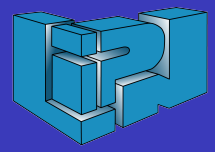


Preuves formelles

Coq : assistant de preuve basé sur le langage Gallina
- types usuels des langages de programmation
- exprimer des assertions sur les expressions

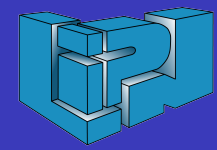
Etapes

- Modéliser le système
- Spécifier formellement le modèle : axiomes, règles d'inférence...
- Écrire des preuves formelles :
  1. Théorème : Proposition
  2. Preuve : Ensemble de tactiques $\rightarrow$ démontrer le théorème



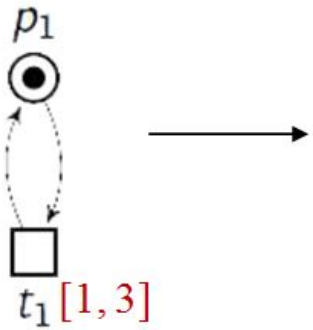
Modélisation via PNML

PNML (Petri Net Markup Language) : format standard de description de réseaux de Petri.



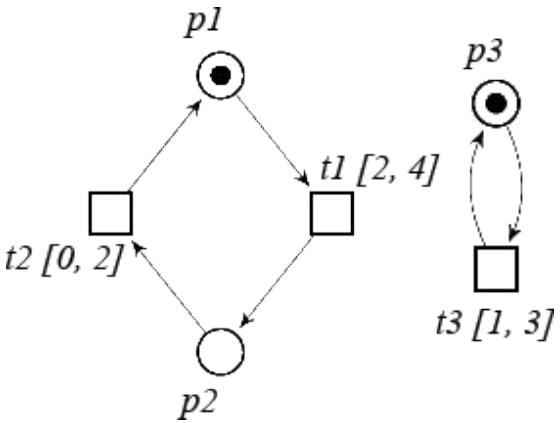
Modélisation via PNML

PNML (Petri Net Markup Language) : format standard de description de réseaux de Petri.




```
<name><text>example1</text></name>
<page id="page1">

<!-- places -->
<place id="p1">
<name><text>p1</text></name>
<initialMarking> <text>1</text>
</initialMarking>
</place>

<!-- transitions -->
<transition id="t1">
<name><text>t1</text></name>
<min><text>1</text></min>
<max><text>3</text></max>
</transition>

<!-- arcs -->
<arc id="arc0" source="p1" target="t1"/>
<arc id="arc1" source="t1" target="p1">
<inscription><text>1</text></inscription>
</arc>
</page>
```

De PNML à Coq



Parser ⇒

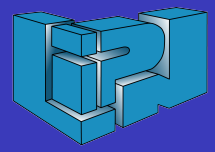
Flex et Bison

CoqIde

File Edit View Navigation Try Tactics Templates Queries Compile Windows Help

module_tpn.v

```
Require Import  Reals List Recdef Bool Arith Min Max.


Record Place : Type := mkPlace { Pl : nat }.
Record Transition : Type := mkTransition { Tr : nat}.
Record State : Type :=  mkstate { mark : list nat ; time : list R}.

Definition P: list Place :=
((mkPlace 1)::(mkPlace 2)::(mkPlace 3)::nil).

Definition T: list Transition :=
((mkTransition 1)::(mkTransition 2)::(mkTransition 3)::nil).

Definition m0: list nat := (5::1::0::nil).
Definition I: list (Transition*nat*nat) :=
((mkTransition 1,2,4)::(mkTransition 2,0,2)::(mkTransition 3,1,3)::nil).

Definition Pre: list (Transition* (list nat)) :=
((mkTransition 1,(1::0::0::nil))::(mkTransition 2,(0::1::0::nil))::
(mkTransition 3,(0::0::1::nil))::nil).

Definition Post: list (Transition* (list nat)) :=
((mkTransition 1,(0::1::0::nil))::(mkTransition 2,(1::0::0::nil))::
(mkTransition 3,(0::0::1::nil))::nil).
```

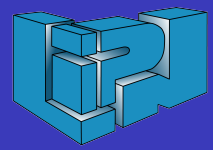


Spécification en Coq I

Aspect statique : composants du réseau et conditions

- État initial du réseau
- Transition sensibilisée
- Transition nouvellement sensibilisée
- Transition persistante
- Transition franchissable

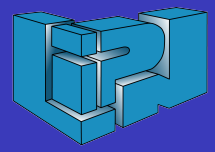
→ Fonctions récursives :*Fixpoint*



Spécification en Coq II

Aspect dynamique : opérations et changements d'états

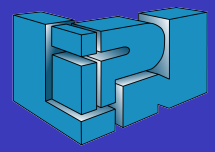
- Relation de transition continue : écoulement du temps
- Relation de transition discrète : franchissement d'une transition



Aspect dynamique : opérations et changements d'états

- Relation de transition continue : écoulement du temps
- Relation de transition discrète : franchissement d'une transition

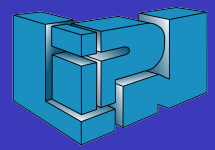
$$(m, v) \overset{(t,d)}{\rightarrow} (m', v')$$



Preuves sur les TPNs

Théorème (Proposition) :

- Franchissabilité d'une séquence S de transitions
- Accessibilité d'un marquage (état)
- Franchissabilité de S dans un temps donné (calcul du temps minimum de franchissement)

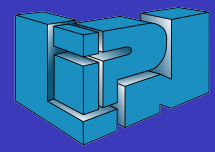


Preuves sur les TPNs

Théorème (Proposition) :

- Franchissabilité d'une séquence S de transitions
- Accessibilité d'un marquage (état)
- Franchissabilité de S dans un temps donné (calcul du temps minimum de franchissement)

S : contre exemple fourni par un model checker



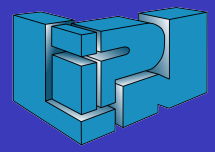
Preuves sur les TPNs

Théorème (Proposition) :

- Franchissabilité d'une séquence S de transitions
- Accessibilité d'un marquage (état)
- Franchissabilité de S dans un temps donné (calcul du temps minimum de franchissement)

S : contre exemple fourni par un model checker

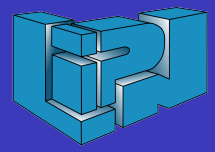
Preuves récursives : Propositions inductives
Preuves récursives et infinies : Propositions coinductives



Preuves sur les TPNs

Le langage Ltac :

- écriture de tactiques personnalisées
- construction par filtrage (pattern matching)
- backtracking : retour en arrière

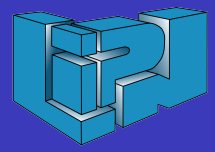


Preuves sur les TPNs

Le langage Ltac :

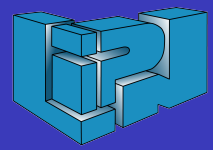
- écriture de tactiques personnalisées
- construction par filtrage (pattern matching)
- backtracking : retour en arrière

$\Rightarrow$ Automatisation des preuves



Preuves sur les TPNs

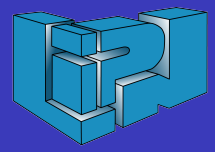
```
Lemma Franchissable: franchissable_i S02 State0.
Proof.
 cofix .
 rewrite (frob_eq S02).
compute.
 prove_franchissable .
Qed.
```



Conclusion

Difficultés :

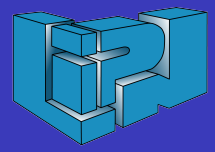
- La manière de faire collaborer les deux approches de vérification
- Calcul sur les nombres réels et la complexité de Coq
- L'automatisation et l'optimisation du temps d'exécution d'une preuve



Conclusion

Dans le futur :

- Étudier le gain de ce travail sur des systèmes complexes : temps, efficacité..
- Faire collaborer notre approche avec le model checking parallèle



Merci

de votre attention