Vérification de programmes synchrones sous contraintes arithmétiques

Emmanuel Haucourt - Élodie-Jane Sims.

27/03/02

Plan :

- Introduction
- Interprétation abstraite - *L'algorithme général*
- Polyhèdres convexes - *Un treillis particulier*
- Conclusions

Articles étudiés :

- **Algorithme général :**
  Dynamic Parititioning in Analyses of Numerical Properties
  Bertrand Jeannet, Nicolas Halbwachs, Pascal Raymond

- **Polyhèdres convexes:**
  Verification of Real-Time Systems using Linear Relation Analysis
  Nicolas Halbwachs, Yann-Erick Proy, Patrick Roumanoff

- **Sémantique de Luste :**[1]
  About synchronous programming and abstract interpretation
  Nicolas Halbwachs

1. Non présentée ici, voir cours

Introduction

But : Vérifier des propriétés de sûreté sur des programmes synchrones.

Méthode : Interprétation abstraite.

Exemple de propriété à vérifier

Programme :

```
initial not b0 and not b1 and (x=0) and (y=0);
transition
b0' = not b1;
b1' = b0;
x' = if b0=b1 then x+1 else x;
y' = if b0=b1 then y else y+1;
```

Valeurs prises par les variables :

b0	0 1 1 0 0 1 1 0 0 1 1 0 0 1 ...
b1	0 0 1 1 0 0 1 1 0 0 1 1 0 0 ...
x	0 1 1 2 2 3 3 4 4 5 5 6 6 7 ...
y	0 0 1 1 2 2 3 3 4 4 5 5 6 6 ...

Propriété à vérifier:

$$x \geq y$$

I - L' algorithme général

Idée : automates

États : valeurs des variables booléennes et valeurs des variables numériques

Mauvais états : ne vérifient pas la propriété de sûreté

But = "Mauvais états atteignables ?"

Problèmes :

→ taille de l'automate (constructible ?)
→ coût de la vérification dans un gros automate

D'où :

➠ interprétation abstraite pour diminuer le nombre d'états

Principe de base de l'interprétation abstraite

Associer un domaine abstrait à un domaine concret tel que :

on peut prouver des propriétés dans le domaine abstrait qui impliquent des propriétés dans le domaine concret.

→ Interêt : calculs dans le domaine abstrait sont finis, moins coûteux

Notations :

Domaine concret :

- États : $\mathcal{S} = \mathcal{B}^p \times \mathcal{N}^q$
- États initiaux : $\mathcal{S}_{init} \in 2^{\mathcal{B}}$
- Mauvais états : $\mathcal{S}_{error} \in 2^{\mathcal{B}}$
- Relation de transition : $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$

- Prouver pas de chemins $s_0, ..., s_n$ avec
  - $s_0 \in \mathcal{S}_{init}$
  - $s_n \in \mathcal{S}_{error}$
  - $s_{i-1} \rightarrow s_i$

Notations :

Domaine abstrait :

- Treillis états abstraits : $L = 2^{\mathcal{B}^p} \times L_{\mathcal{N}}$
- Structure de contrôle :
  $(Q, Q_0, Q_F, \rightsquigarrow, def)$
  - Q : états de l'automates
  - $Q_0 \subseteq Q$: états initiaux
  - $Q_F \subseteq Q$: états finaux
  - $\rightsquigarrow \subseteq Q \times Q$
  - $def : Q \rightarrow L$: fonction des états de l'automate aux états du treilli abstrait

Diagramme

$$
\begin{tikzcd}
Q \arrow[r, "def"] & L \arrow[ddd, "\gamma"] & = & 2^{\mathcal{B}^p} \arrow[dd, equal] & \times & L_{\mathcal{N}} \arrow[d, shift left, "\gamma_{\mathcal{N}}"] \\
& & & & & 2^{\mathcal{N}^q} \arrow[u, shift left, "\alpha_{\mathcal{N}}"] \\
& & & \mathcal{B}^p & \times & \mathcal{N}^q \arrow[u, "\in"] \\
& 2^{\mathcal{S}} & = & 2^{\mathcal{B}^p \times \mathcal{N}^q} & &
\end{tikzcd}
$$

Propriétés :

✓ À un état de l'automate q correspond $\gamma(def(q))$ un ensemble d'états concrets.

✓ Tous les états initiaux concrets sont dans la correspondance des états initiaux de l'automate.

✓ Tous les mauvais états concrets sont dans la correspondance des états finaux de l'automate.

✓ S'il n'y a pas de chemin dans l'automate d'un état initial à un état final, il n'y a pas de mauvais états atteignables depuis l'état initial du programme.

Création "dynamique" de l'automate de contrôle :

Abstraction = Approximation

Pour un mauvais état atteignable, on va trouver un mauvais chemin dans l'automate. $\ddot\smile$

Mais : Pour un mauvais chemin trouvé dans l'automate, on peut ne pas avoir de mauvais état atteignable. $\ddot\frown$

Problème : Trouver le bon def pour être assez précis pour ne pas avoir trop de "mauvaises alertes" et pas trop pour que ça ne soit pas trop coûteux.

Solution : Création "dynamique" de l'automate de contrôle.

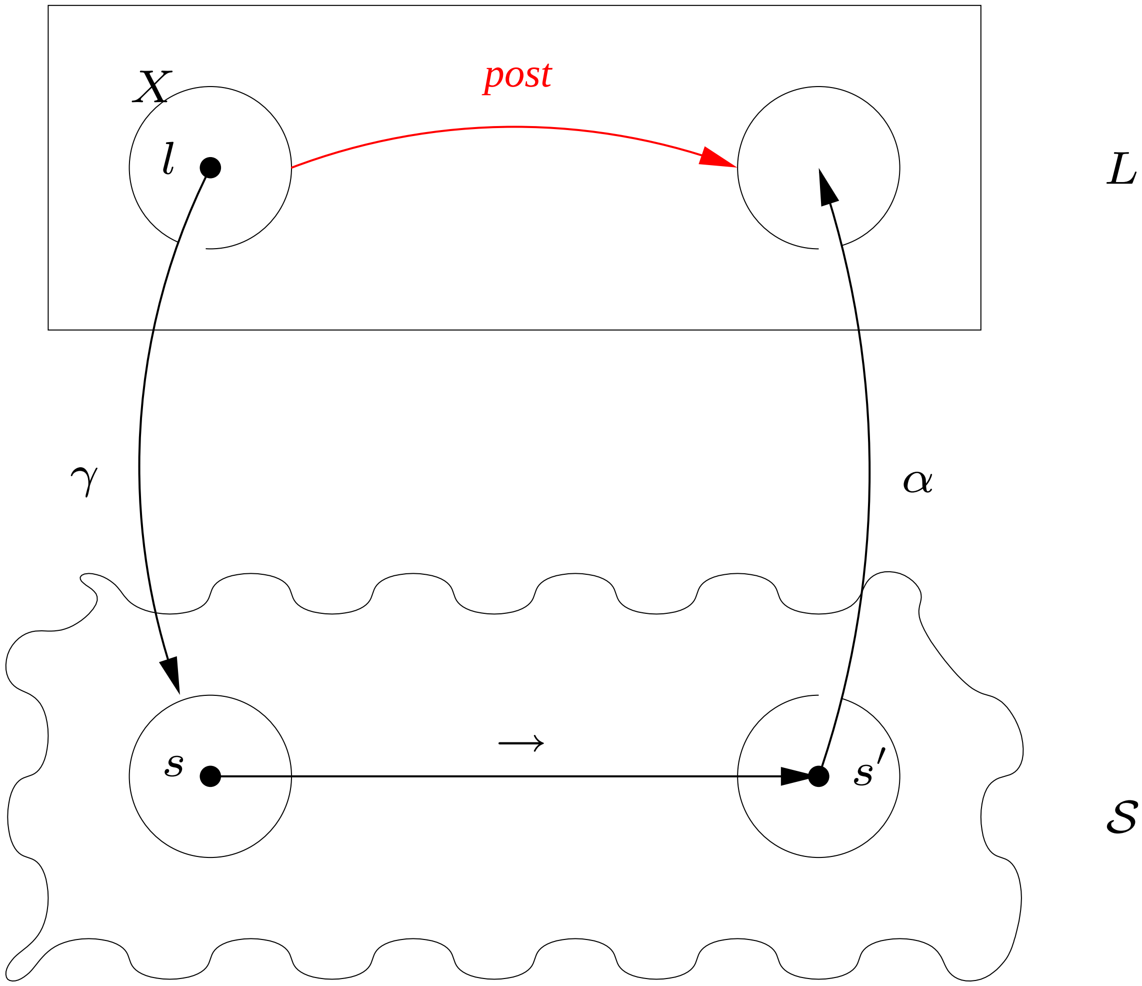
Définitions :

Soit α tq $\alpha(x) \subseteq y \Leftrightarrow x \subseteq \gamma(y)$ (pseudo-inverse de γ).

- $post(X) : L \to L$
  $= \{\alpha(s') | \exists l \in X, \exists s \in \gamma(l) : s \to s'\}$
- $pre(X) : L \to L$
  $= \{\alpha(s) | \exists l \in X, \exists s' \in \gamma(l) : s \to s'\}$
- $reach(q) : Q \to L$
  $= \bigsqcup_{q' \leadsto q} post(reach(q')) \sqcap def(q)$
- $danger(q) : Q \to L$
  $= \bigsqcup_{q \leadsto q'} pre(danger(q')) \sqcap reach(q)$

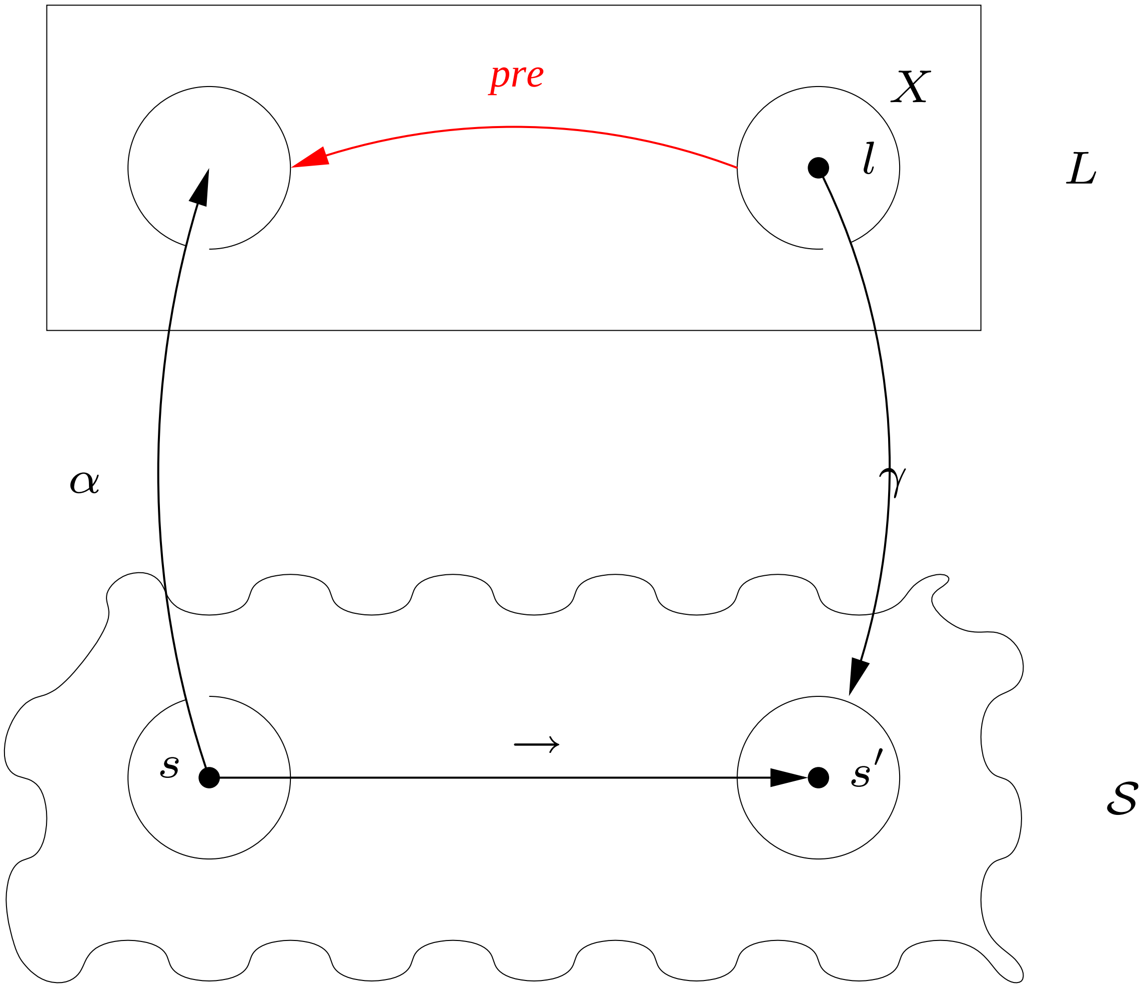
Post

$$post(X) : L \longrightarrow L$$
$$= \{\alpha(s')|\exists l \in X, \exists s \in \gamma(l) : s \longrightarrow s'\}$$



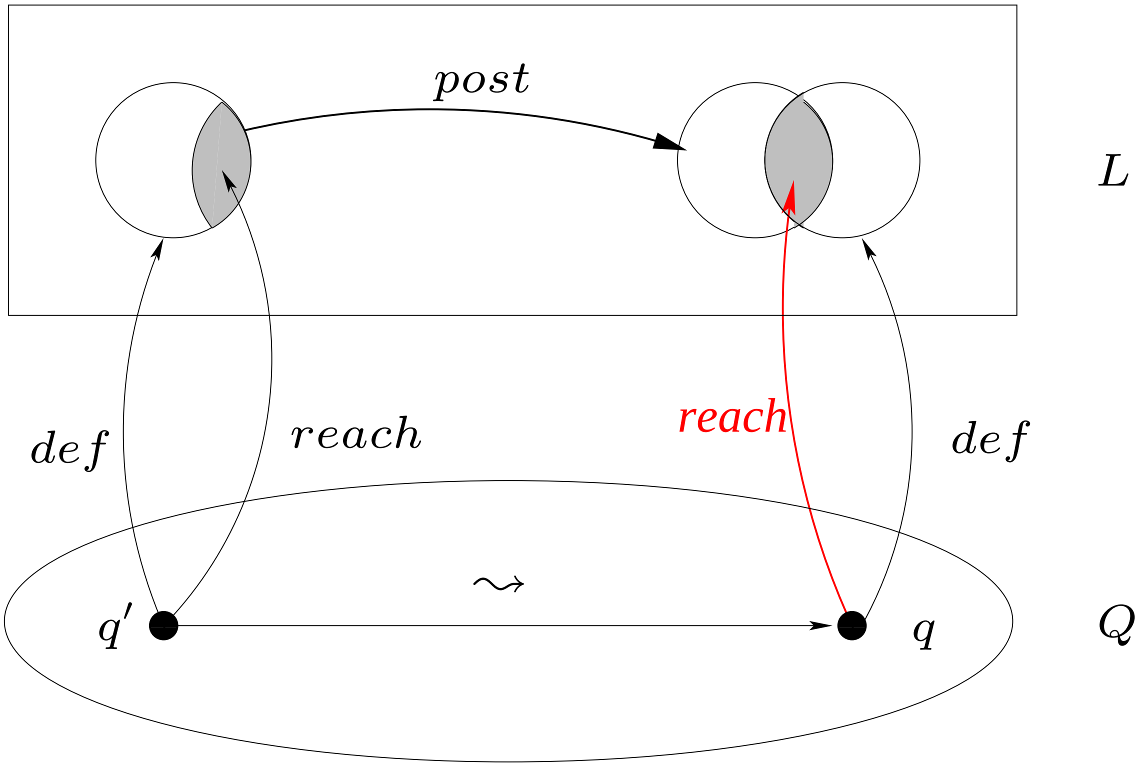
Pre

$$pre(X) : L \longrightarrow L$$
$$= \{\alpha(s) | \exists l \in X, \exists s' \in \gamma(l) : s \longrightarrow s'\}$$



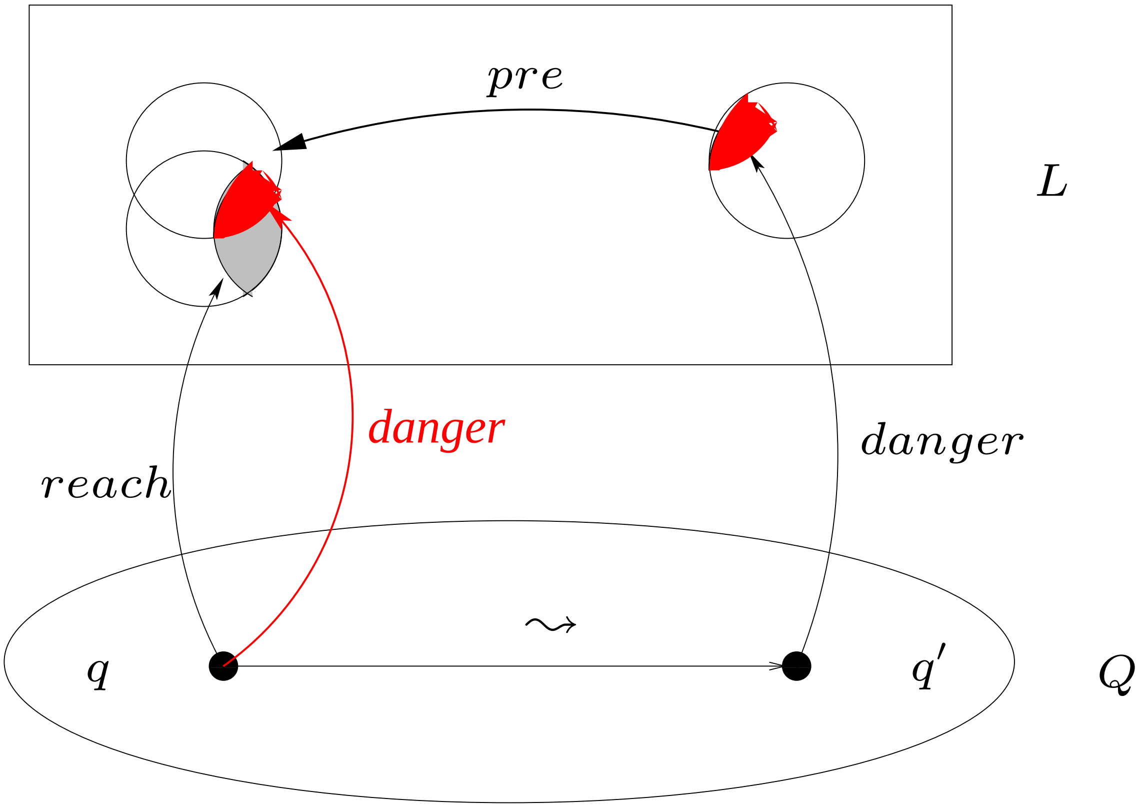
Reach

$$reach(q) : Q \to L$$
$$= \bigsqcup_{q' \rightsquigarrow q} post(reach(q')) \sqcap def(q)$$



Danger

$$danger(q) : Q \longrightarrow L$$
$$= \bigsqcup_{q \rightsquigarrow q'} pre(danger(q')) \sqcap reach(q)$$



Algorithme

But : montrer que $\forall q \; danger(q) = \bot_L$

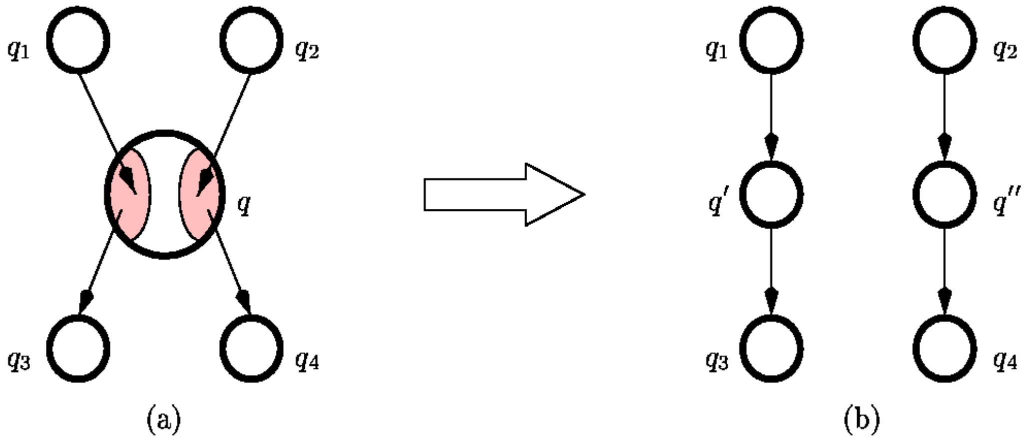
On va itérer les calculs de $reach$ et $danger$ et "updater" l'automate en cours de route jusqu'à convergence.

- ✗ Départ : Q_o et Q_F sont pris suivant un choix qui vérifie :
  $\bigcup_{q \in Q_0} \gamma(def(q)) = \mathcal{S}_{init}$
  $\bigcup_{q \in Q_F} \gamma(def(q)) = \mathcal{S}_{error}$
- ✗ $def(q) = danger(q)$ (on ne cherche à vérifier que les états de danger
- ✗ Mise à jours de $\rightsquigarrow$ après un calcul de $reach$
  $q \rightsquigarrow q' \Leftrightarrow post(reach(q)) \sqcap def(q') \neq \bot_L$
- ✗ Mise à jours de $\rightsquigarrow$ après un calcul de $danger$
  $q \rightsquigarrow q' \Leftrightarrow pre(danger(q')) \sqcap reach(q) \neq \bot_L$

Raffinement

Comme dit précédement, on peut avoir un automate plus ou moins précis.

Exemple de raffinement :



Le choix du raffinement est une heuristique, qui dépend du treillis choisis.

Par exemple,

➠ le treillis des polyhèdre

Remarque sur la correction de l'algorithme

- La preuve de la correction repose sur l'affirmation que nous sommes dans la cadre d'interprétation abstraite avec des connections de Gallois.
- Les auteurs suppose que l'on travail dans un domaine abstrait, le treillis complet L, qui est en connection de Galois avec le domaine concret $\mathcal{S}$. Ce qui est assuré par l'hypothèse que $L_{\mathcal{N}}$ est un treillis complet en connection de Galois avec $\mathcal{N}^p$.
- Mais il faut aussi s'assurer que def est bien à tout moment de l'algorithme une surapproximation.

II - Un treillis particulier

Polyhèdres convexes

Conclusion