# ACI PERSÉE

## Techniques symboliques pour la vérification automatique des systèmes critiques hétérogènes

Philippe Schnoebelen

http://www.lsv-ens-cachan.fr/aci-persee/

# Plan de la présentation

- Participants

- Contexte scientifique : vérification symbolique, enjeux, obstacles

- Notre programme de recherche : vérification exacte des systèmes hétérogènes complexes

– représentations symboliques

– stratégies d'accélération

– techniques d'abstraction

# Les participants

**LSV (Cachan)**
Ph. Schnoebelen, A. Finkel, J.-M. Couvreur, D. Nowak, ...

**LIAFA (Paris 7)**
A. Bouajjani, E. Asarin, M. Sighireanu, P. Habermehl, Y. Jurski, ...

**LaBRI (Bordeaux)**
G. Sutre, A. Griffault, K. Musumbu, F. Herbreteau, ...

# Vérification des systèmes critiques

Le besoin de vérifier formellement les systèmes critiques est maintenant largement admis.

Actuellement, la vérification automatique (= le model-checking) est plus populaire que les méthodes de preuves assistées.

Les principales limitations du model-checking :

1. le système doit être abstrait sous la forme d'un modèle fini
2. explosion combinatoire
3. on vérifie essentiellement des propriétés de sûreté

$\Rightarrow$ s'applique essentiellement à des parties d'un système, la construction du modèle abstrait nécessite une grande expertise.

# Au delà des modèles finis

De nombreuses approches proposent de vérifier automatiquement des systèmes à infinité d'états :
– systèmes temporisés,
– algorithmes distribués,
– systèmes mobiles,
– systèmes hybrides,
– systèmes à pile,
– protocoles asynchrones,
– utilisation de pointeurs,
...

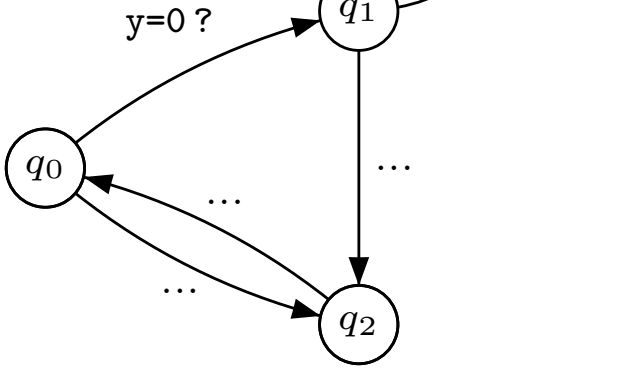
Un cadre générique répandu :
modèles = automates étendus
vérification = model-checking symbolique + accélération de points fixes.

# La vérification symbolique

```
channel c
   process P1(u)
    send u -> c
      ...
||  process P2(v)
    receive c <- n
    if #P1<#P2 ···
```

# La vérification symbolique

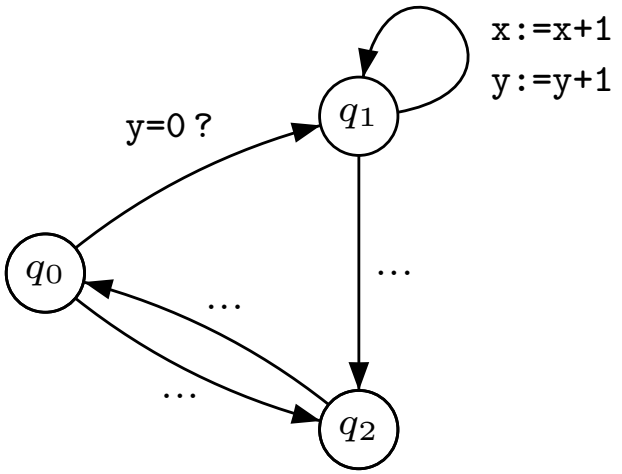
```
channel c
   process P1(u)
    send u -> c
      ...
||  process P2(v)
    receive c <- n
    if #P1<#P2 ···
```



# La vérification symbolique

```
channel c
   process P1(u)
    send u -> c
      ...
||  process P2(v)
    receive c <- n
    if #P1<#P2 ···
```

$\Rightarrow$

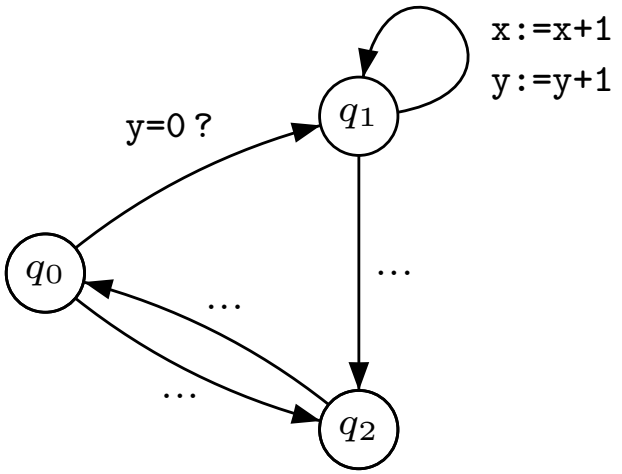



$S_0 = \langle q_0, x = *, y = * \rangle$

# La vérification symbolique

```
channel c
   process P1(u)
    send u -> c
      ...
||  process P2(v)
    receive c <- n
    if #P1<#P2 ···
```

$\Rightarrow$



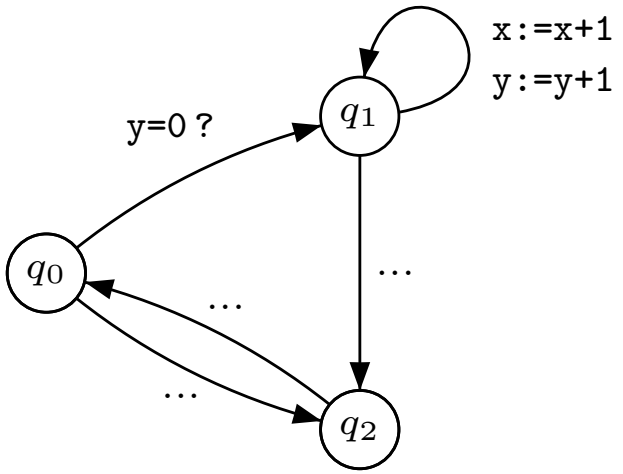

$$S_0 = \langle q_0, x = *, y = * \rangle$$

$$S_1 = \langle q_0, x = *, y = * \rangle + \langle q_1, x = *, y = 0 \rangle$$

# La vérification symbolique

```
channel c
   process P1(u)
    send u -> c
      ...
||  process P2(v)
    receive c <- n
    if #P1<#P2 ···
```

$\Rightarrow$




$$S_0 = \langle q_0, x = *, y = * \rangle$$

$$S_1 = \langle q_0, x = *, y = * \rangle + \langle q_1, x = *, y = 0 \rangle$$

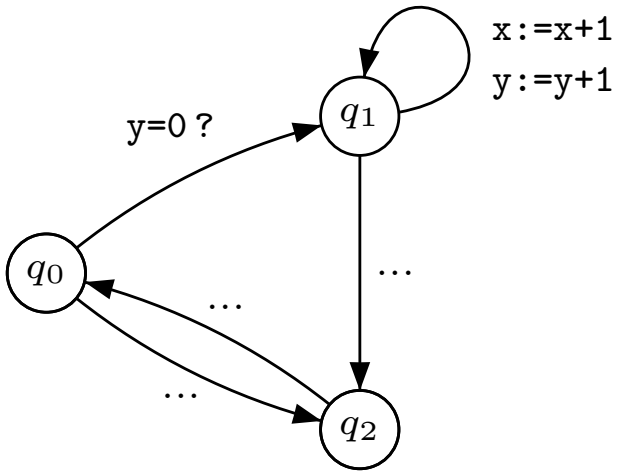
$$S_2 = \langle q_0, x = *, y = * \rangle \ + \ \langle q_1, x = *, y = 0 \rangle + \langle q_1, x > 0, y = 1 \rangle$$

$$\vdots$$

# La vérification symbolique

```
channel c
   process P1(u)
    send u -> c
      ...
||  process P2(v)
    receive c <- n
    if #P1<#P2 ···
```

$\Rightarrow$




$$S_0 = \langle q_0, x = *, y = * \rangle$$

$$S_1 = \langle q_0, x = *, y = * \rangle + \langle q_1, x = *, y = 0 \rangle$$

$$S_2 = \langle q_0, x = *, y = * \rangle + \langle q_1, x = *, y = 0 \rangle + \langle q_1, x > 0, y = 1 \rangle$$

$$\vdots$$

$$S_\omega = \langle q_0, x = *, y = * \rangle + \langle q_1, x = *, y \leq x \rangle$$

# Accélération de convergence

- Accélération $\neq$ élargissement !

# Accélération de convergence

- Accélération $\neq$ élargissement !

- Technique 1 : identification des situations résolubles

  p.ex., actions de la forme " si $cond_i(\vec{x})$ alors $\vec{x} := M_i\vec{x} + v_i$ "
  telles que le monoïde multiplicatif engendré par les $M_i$ soit fini.

# Accélération de convergence

- Accélération $\neq$ élargissement !

- Technique 1 : identification des situations résolubles

  p.ex., actions de la forme " si $cond_i(\vec{x})$ alors $\vec{x} := M_i\vec{x} + v_i$ "
  telles que le monoïde multiplicatif engendré par les $M_i$ soit fini.

- Technique 2: extrapolation

  1. détecter $S_n \xrightarrow{\Delta} S_{n+1} \xrightarrow{\Delta} S_{n+2}$
  2. vérifier que $S_n + k\Delta \xrightarrow{\Delta} S_n + (k+1)\Delta$ pour tout $k \in \mathbb{N}$
  3. déduire $S_\omega = \{s \mid \exists k \ : \ s \in S_n + k\Delta\}$

# Les systèmes hétérogènes

Protocoles modernes où apparaissent plusieurs sources différentes d'infinitude.

Exemple : le protocole BRP
– canaux asynchrones non fiables
– horloges
– compteurs
– paramètres

# Les systèmes hétérogènes

Protocoles modernes où apparaissent plusieurs sources différentes d'infinitude.

Exemple : le protocole BRP
– canaux asynchrones non fiables
– horloges
– compteurs
– paramètres

Modèle générique : automate (contrôle) + variables hétérogènes (domaines différents)

# Les systèmes hétérogènes

Protocoles modernes où apparaissent plusieurs sources différentes d'infinitude.

Exemple : le protocole BRP
– canaux asynchrones non fiables
– horloges
– compteurs
– paramètres

Modèle générique : automate (contrôle) + variables hétérogènes (domaines différents)

Le projet PERSÉE développe des techniques capables de vérifier automatiquement et exactement des systèmes hétérogènes complexes.

# Pb1 – Représentations symboliques

De nombreuses structures de données existent pour manipuler efficacement les valeurs de variables homogènes
– horloges : DBM, CDD, RED, ...
– compteurs : automates binaires, contraintes linéaires, formules de Presburger, ...
– files, piles : SRE, langages réguliers, ...

# Pb1 – Représentations symboliques

De nombreuses structures de données existent pour manipuler efficacement les valeurs de variables homogènes
– horloges : DBM, CDD, RED, ...
– compteurs : automates binaires, contraintes linéaires, formules de Presburger, ...
– files, piles : SRE, langages réguliers, ...

Comment prendre en compte les variables hétérogènes ?

– produit cartésien : canonicité ?

– corrélations entre variables : comment manipuler

$$\{(w, n, x) \mid w \in (a^*b)^*c \ \wedge \ |w|_a + |w|_c = n \ \wedge \ n \geq \lfloor x \rfloor\} \ ?$$

# Pb1 – Représentations symboliques

De nombreuses structures de données existent pour manipuler efficacement les valeurs de variables homogènes
– horloges : DBM, CDD, RED, ...
– compteurs : automates binaires, contraintes linéaires, formules de Presburger, ...
– files, piles : SRE, langages réguliers, ...

Comment prendre en compte les variables hétérogènes ?

– produit cartésien : canonicité ?

– corrélations entre variables : comment manipuler

$$\{(w, n, x) \mid w \in (a^*b)^*c \ \wedge \ |w|_a + |w|_c = n \ \wedge \ n \geq \lfloor x \rfloor\} \ ?$$

PERSÉE : Résultats attendus = identification de combinaisons manipulables exactement (et efficacement) + librairies de représentations symboliques

# Pb2 – Stratégies d'accélération

Constat : besoin d'heuristiques pour guider l'accélération

– la combinatoire des circuits est explosive

– les calculs pour un circuit sont très coûteux

# Pb2 – Stratégies d'accélération

Constat : besoin d'heuristiques pour guider l'accélération

– la combinatoire des circuits est explosive

– les calculs pour un circuit sont très coûteux

Exemple : `FAST` choisit des circuits $\sigma$ tels que $|\sigma^*(S)|_{\text{symb}} \approx |S|_{\text{symb}}$.

# Pb2 – Stratégies d'accélération

Constat : besoin d'heuristiques pour guider l'accélération

– la combinatoire des circuits est explosive

– les calculs pour un circuit sont très coûteux

Exemple : `FAST` choisit des circuits $\sigma$ tels que $|\sigma^*(S)|_{\text{symb}} \approx |S|_{\text{symb}}$.

PERSÉE :

expérimentations de nouvelles heuristiques (analyse de dépendance, . . . ),
stratégie définie par l'utilisateur (à la HyTech).
quelles notions d'extrapolations ?

# Pb3 – Abstractions de modèles

L'utilisation d'abstractions est indispensable pour vérifier des systèmes complexes.
(Y compris après avoir construit un modèle formel abstrait)

Une technique récente pour vérifier des programmes : l'abstraction automatique (par " abstraction de prédicats ") sous forme de modèles finis (`Blast`, ...)

# Pb3 – Abstractions de modèles

L'utilisation d'abstractions est indispensable pour vérifier des systèmes complexes.
(Y compris après avoir construit un modèle formel abstrait)

Une technique récente pour vérifier des programmes : l'abstraction automatique (par " abstraction de prédicats ") sous forme de modèles finis (`Blast`, ...)

1. Si l'abstraction vérifie la propriété de sûreté, alors le programme est sûr.

2. Sinon on analyse le contre-exemple. Peut-on le lifter de l'abstraction au programme réel ?

3. Si oui, le programme n'est pas sûr. Sinon on sait où raffiner l'abstraction : on recommence.

# Pb3 – Abstractions de modèles

L'utilisation d'abstractions est indispensable pour vérifier des systèmes complexes.
(Y compris après avoir construit un modèle formel abstrait)

Une technique récente pour vérifier des programmes : l'abstraction automatique (par " abstraction de prédicats ") sous forme de modèles finis (`Blast`, ...)

1. Si l'abstraction vérifie la propriété de sûreté, alors le programme est sûr.

2. Sinon on analyse le contre-exemple. Peut-on le lifter de l'abstraction au programme réel ?

3. Si oui, le programme n'est pas sûr. Sinon on sait où raffiner l'abstraction : on recommence.

PERSÉE : Adapter cette approche pour nos modèles étendus.
– Comment engendrer des modèles pour lesquels le model checking symbolique saura conclure ?
– Quelle forme pour les contre-exemples ?
– Comment les lifter au programme réel ?