Méthodes d'ordre partiel pour la vérification de systèmes concurrents

Élodie-Jane Sims.

17/04/02

Articles étudiés :

- Partial-Order Methods for Temporal Verification
  Pierre Wolper, Patrice Godefroid

- Partial-Order Methods for Model Checking: From Linear Time to Branching Time
  Bernard Willems, Pierre Wolper

- On the Costs and Benefits of using Partial-Order Methods for the Verification of Concurrent Systems
  Patrice Godefroid

- Using Partial-Order Methods in the Formal Validation of Industrial Concurrent Programs
  Patrice Godefroid, Doron Peled, Mark Staskauskas

But :
Vérifier des systèmes concurrents

Méthode :
Calcul du produit des descriptions des différents processus.

Problème :
Taille du produit.

➠ Utiliser les ordres partiels pour la réduire

Idée :
Différentes étapes peuvent permuter pour produire le même résutat
on n'explore pas toutes les permutations possibles.

Notations

Processus P_i :

$$A_i = (\Sigma_i, S_i, \Delta_i, s_{0i})$$

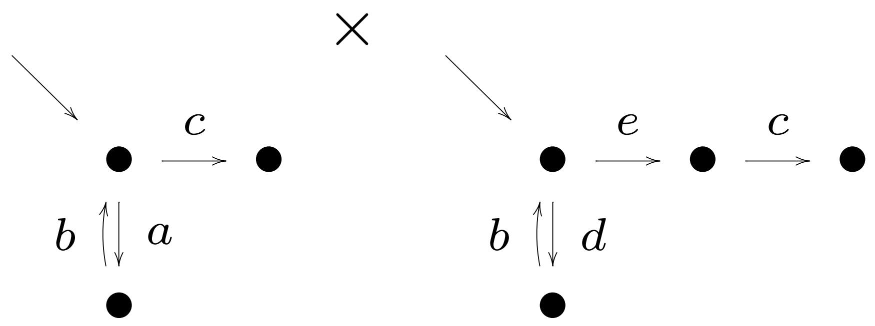
Programme P :

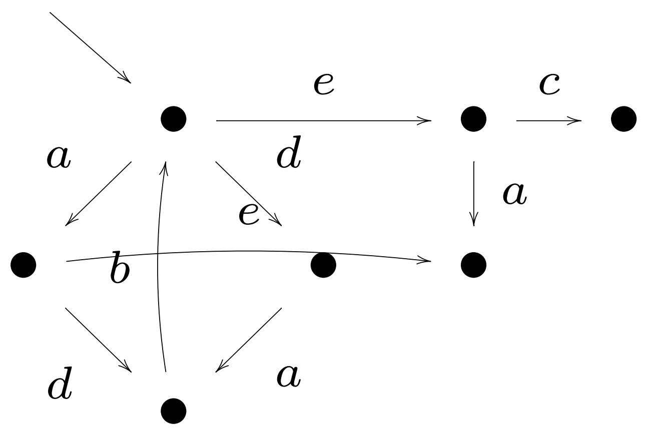
$$A = (\Sigma, S, \Delta, s_0)$$

- $\Sigma = \cup \Sigma_i$
- $S = \Pi S_i$
- $s_0 = (s_{01}, ..., s_{0n})$
- $(s, a, s') \in \Delta$:
  - $a \in \Sigma_i$: $(s(i), a, s'(i)) \in \Delta_i$
  - $a \notin \Sigma_i$: $s(i) = s'(i)$

A_G : A restreint au états atteignables à partir de s_0

Exemple de produit de processus





Algorithmes : *recherche sélective*

– on ne veut pas explorer toutes les permutations

– mais en explorer assez pour prouver les propriétés que l'on veut

➠ plusieurs solutions possibles... combinables...

Exemples

Pour la détection d'états de bloquage :

- *Ensembles persistants*
- *Ensembles d'oublis*

Pour l'accessibilité :

- *Ensembles cycle-proviso*

Pour la vérification de modèles :

- *Invisibilité*

Notions de base

Transition du programme ou *Transition* :
classe d'équivalence des transitions de A_G

– même a

– $\forall i\, a \in \Sigma_i$ même $s(i)$ et même $s'(i)$

Transition *activée* en s :
une des transitions de A_G de cette classe commence en s

Transitions indépendantes

$t1$ et $t2$ **indépendantes** si :

- si $t1$ activée en s, $s \rightarrow^{t1} s'$ alors
  ($t2$ activée en s ssi $t2$ activée en s')
- si $t2$ activée en s, $s \rightarrow^{t2} s'$ alors
  ($t1$ activée en s ssi $t1$ activée en s')
- si $t1$ et $t2$ activées en s
  $\exists! s'$ t.q. $s \Rightarrow^{t1t2} s'$ et $s \Rightarrow^{t2t1} s'$

Transitions indépendantes

$$\begin{tikzcd}
 & s' \arrow[dr, dashed, "t_2"] & \\
s \arrow[ur, "t_1"] \arrow[dr, "t_2"] & & {} \\
 & {} &
\end{tikzcd}$$

$$\begin{tikzcd}
 & s' \arrow[dr, "t_2"] & \\
s \arrow[ur, "t_1"] \arrow[dr, dashed, "t_2"] & & {} \\
 & {} &
\end{tikzcd}$$

$$\begin{tikzcd}
 & {} \arrow[r, Rightarrow, dashed, "t_1"] & {} \arrow[dr, Rightarrow, dashed, "t_2"] & \\
s \arrow[ur, "t_1"] \arrow[dr, "t_2"] & & & !s' \\
 & {} \arrow[r, Rightarrow, dashed, "t_2"] & {} \arrow[ur, Rightarrow, dashed, "t_1"] &
\end{tikzcd}$$

Ensembles persistants

Un ensemble T de transitions activées en s est ***persistant*** si toutes les transitions hors de T activées en s lui sont indépendantes et on reste toujours indépendant de T en effectuant des transitions hors de T

Propriété : Une recherche sélective par sélection d'ensembles persistants (non vides) atteint tous les états de blocage.

Idée de la preuve :
Par induction sur la taille d'un chemin de s à un état d de blocage

- Cas n=0 trivial
- Cas n+1 :
  - on atteint forcement un état dans les persistants de s (sinon comme hors du persistant est indépendant on peut toujours faire les trans. dans persistant et on est pas bloqué)
  - on peut switché pour l'atteindre au début et se ramener au cas n

Calcul d'ensembles persistants:

- Par analyse du programme.
- Par analyse des conditions nécessaire pour activer une transitions (*ensembles têtus*)

Ensembles d'oublis

Idée : Ne pas analyser le programme mais se souvenir du "passé".

Principe : Associer aux états un ensembles de transitions d'oublis qui n'ont pas besoin d'être explorées.

Exemple : si a et b sont indépendantes

$$\begin{tikzcd}
& s2 \arrow[dr, "b"] & \\
s1 \arrow[ur, "a"] \arrow[dr, "b"'] & & s3 \\
& s4 \arrow[ur, "a"'] &
\end{tikzcd}$$

on explore d'abord $s1 \rightarrow^a s2$, puis $s1 \rightarrow^b s4$ puis $s2 \rightarrow^b s3$ et là on se souvient pour $s4$ qu'il n'y a pas besoin d'explorer $s4 \rightarrow^a s3$.

Propriété : Il existe un algorithme de calcul d'ensembles d'oublis qui assure que tout état de blocage sera détecté.

Ensembles cycle-proviso

Pour l'accessibilité : *Problème d'indifférence*
$t1 = (s_{01}, a, s_{01})$, $t2 = (s_{02}, b, l)$
$\{t1\}$ est un ensemble persistant, donc lors de l'analyse on n'atteint pas (s_{01}, l) et on ne détecte pas de blocage. Il n'y a pas de blocage car on peut toujours faire $t1$ mais par contre il est faux que l'on n'atteint pas (s_{01}, l).

Solution :
Ensembles cycle-proviso : pour tous les états d'un cycle on sélectionne toutes les transitions possibles.

Rappel syntaxe logiques temporelles

LTL : $\mathcal{P}$, $\vee, \wedge, \neg$, $\mathcal{U}, \circ$
LTL-$\circ$: $\mathcal{P}$, $\vee, \wedge, \neg$, $\mathcal{U}$

CTL : $\mathcal{P}$, $\vee, \wedge, \neg$, $A\mathcal{U}, A\circ, E\mathcal{U}, E\circ$
CTL-$\circ$: $\mathcal{P}$, $\vee, \wedge, \neg$, $A\mathcal{U}, E\mathcal{U}$
CTL* : $\mathcal{P}$, $\vee, \wedge, \neg$, $\mathcal{U}, \circ$, A, E

Vérifier des propriétés de logique temporelle

On ajoute à chaque état de A_G un ensemble de propriété atomique valide dans cet état

$$L : S \to \mathcal{P}$$

Vérification : $A_G \times A_{\neg\phi}$ is $\emptyset$?

On cherche à nouveau des critères pour faire une recherche sélective pour réduire
- A_G : technique de pré-réduction
- $A_G \times A_{\neg\phi}$: technique à-la-volée

Pré-réduction : cas temps linéaire

– Un ensemble T de transitions est $\mathcal{P}$-**invisible** en s si pour toute transition de s vers s' dans T
$L(s)$ et $L(s')$ coincident en $\mathcal{P}$.

– Un ensemble T de transitions est $\mathcal{P}$-**persistement-invisible** en s si il est $\mathcal{P}$-invisible et
la $\mathcal{P}$-invisibilité est persistante.
càd même notion de persistance que précédement mais pour $\mathcal{P}$-invisibilité au lieu d'indépendance.

Propriété : Si la recherche sélective se fait sur des ensembles persistants, cycle-provisos, $\mathcal{P}$-persistement-invisibles elle préserve les propriétés LTL-$\circ$ formée sur $\mathcal{P}$.

Pré-réduction : cas temps arborescent

- Un ensemble T de transitions est $\mathcal{F}$-**invisible** en s si pour toute transition de s vers s' dans T
  s et s' satisfont les mêmes formules dans $\mathcal{F}$.

- Un ensemble T de transitions est $\mathcal{F}$-**persistement-invisible** en s sous les conditions attendues.

- La sélection est **singleton** si les ensembles sélectionnés sont des singletons ou toutes les transitions possibles.

Propriété : Si la recherche sélective se fait sur des ensembles persistants, singleton, cycle-provisos, $\mathcal{F}$-persistement-invisibles elle préserve les propriétés CTL*-$\circ$ formées sur $\mathcal{P}$.

Conclusion

- Les méthodes d'ordre partiels sont des méthodes génériques pour résoudre le pb d'explosion d'état en vérification.

- Pour certains cas (beaucoup d'inter-eactions) elles n'apportent pas grand chose (mais n'empire pas la recherche).

- Dans la plupart des cas en pratique, même avec peu de processus, apporte une amélioration significative du temps et de l'espace nécessaire.

- Ces méthodes sont génériques et doivent être adaptées spécifiquement à chaque problème (choix d'un bon ensemble persistant).

Remarques variées

- Explorer un nombre minimal de transitions activées est juste une heuristique et n'amène pas forcement à explorer un nombre minimal d'états.

- Les problèmes de sûreté se ramènent à des problèmes d'accessibilité.

- Il y a des programmes pour lesquels une recherche sélective améliore significativement la vérification et d'autres où elle n'a aucun effet.

- On peut rajouter un processus représentant la propriété à vérifier.