Langage pour la vérification de modèles par contraintes

JFPC 2017

Pierre Talbot Clément Poncelet
{talbot,poncelet}@ircam.fr

Institut de Recherche et Coordination Acoustique/Musique (IRCAM)
Université Pierre et Marie Curie (UPMC)

13 juin 2017

Introduction

Problématique

Model-checking + Programmation par contraintes = ?

Points d'accroches

- Deux techniques basées sur l'énumération plutôt que la prouvabilité.
- Exploration d'un espace d'état très large + optimisations.
- Élaboration du modèle d'un problème ou système.

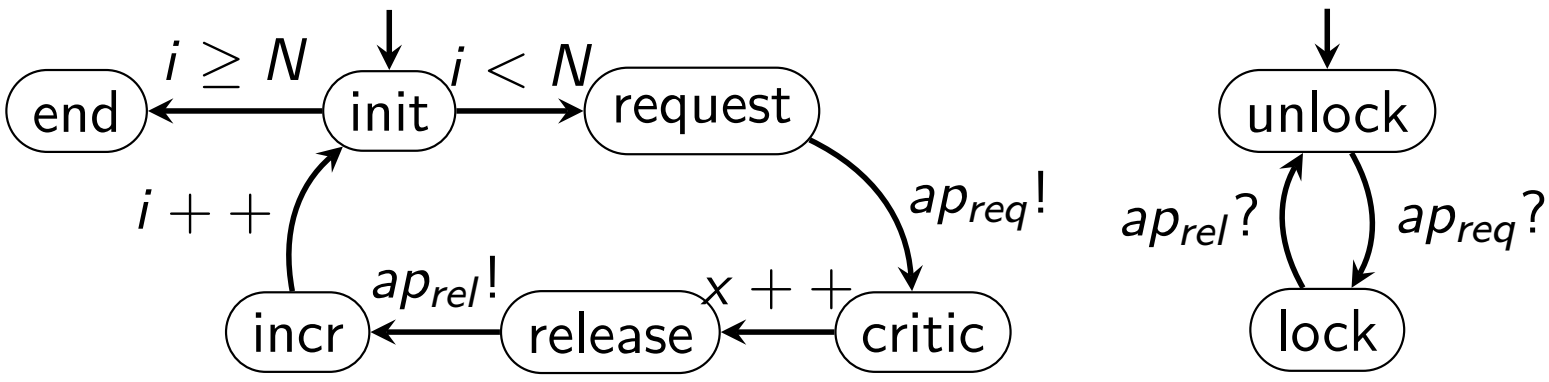
Que peut-on mettre en commun ?

Model-checking

Définition

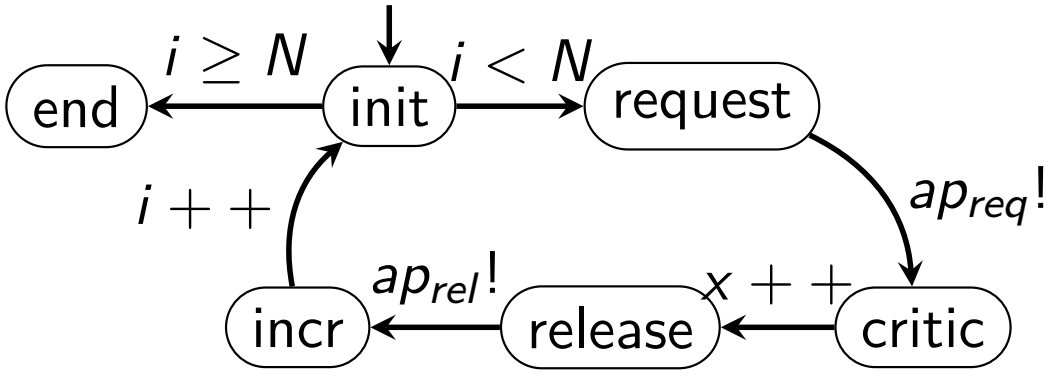
À partir d'un modèle M et d'une formule F, établir $M \models F$.

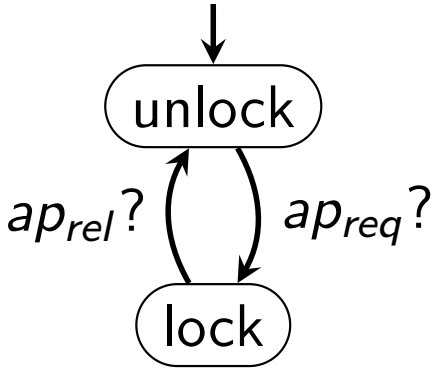
- Problème d'exclusion mutuelle de Peterson.

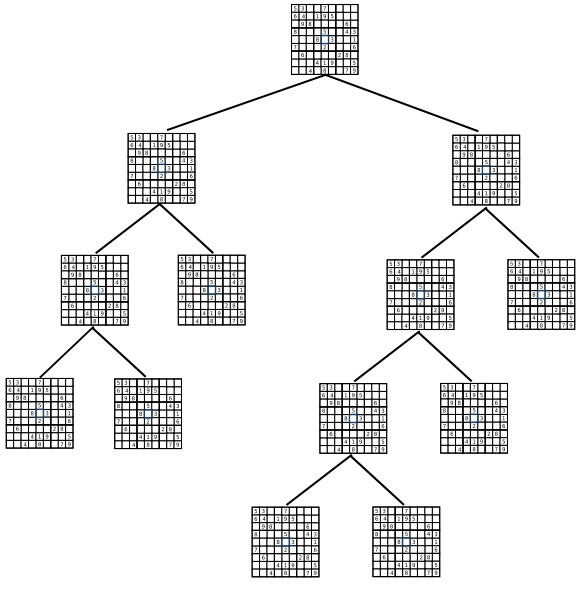


Formule logique : $\Box\neg(p_1.critic \wedge p_2.critic)$

En images...


end
init
request
incr
release
critic
$i \geq N$
$i < N$
$i++$
$ap_{req}!$
$ap_{rel}!$
$x++$


unlock
lock
$ap_{rel}?$
$ap_{req}?$



Model-checking : différences avec CP

- Modèle construit dynamiquement “pendant l’exploration”.
- Pure énumération des variables, pas de propagation.
- Vérification d’une formule logique pendant l’exploration.

Proposition

Spacetime programming

- Langage de stratégies d'exploration d'un espace d'états.
- Calcul de processus synchrone basé sur les treillis.

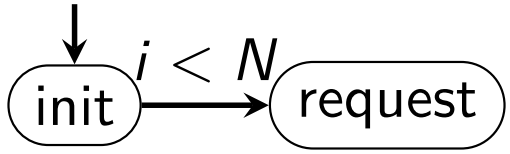
Proposition

- Formalisation commune aux deux paradigmes.
- Utilisation de spacetime pour créer un processus par responsabilité :
  - Le solveur CP.
  - Le système de transition en model-checking.
  - Une formule logique devient une stratégie d'exploration.

Ce travail est une étude préliminaire.

Mise en commun

Soit $i = 0$ et le modèle paramétré par N entre 0 et 10.



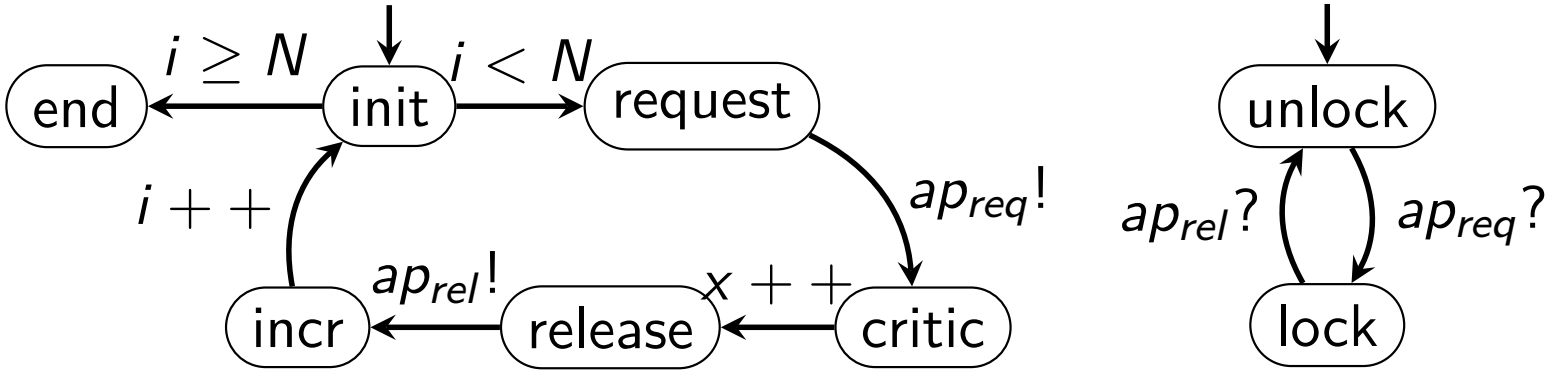
Extension de la fonction d'affectation

- Model-checking : fonction d'affectation $Eval : Var \to \mathbb{N}$
- Contraintes : $\langle d, C \rangle$

- Les gardes et effets (sur les transitions) sont transformés en contraintes.
- Un CSP remplace la fonction d'affectation.

- La satisfiabilité du CSP est établie dans chaque nud du modèle.
- Un CSP insatisfiable représente un **état inatteignable**.

Affectation



- $\langle \{i : 0, x : 0, N : 0..10\}, \emptyset \rangle$
- $\langle \{i : 0, x : 0, N : 1..10\}, \{i < N\} \rangle$: Restreint le domaine de N.
- $\langle \{i : 0, x : (0, 1), N : 1..10\}, \{i < N, x_1 = x_0 + 1\} \rangle$: Variable de flux.
- $\langle \{i : (0, 1), x : (0, 1), N : 1..10\}, \{i < N, x_1 = x_0 + 1, i_1 = i_0 + 1\} \rangle$

- **Variable de flux** : CSP dynamique qui “grandit” au fur et à mesure de l’exploration.
- **Idée** : utiliser des techniques d’interprétation abstraite sur ces flux.

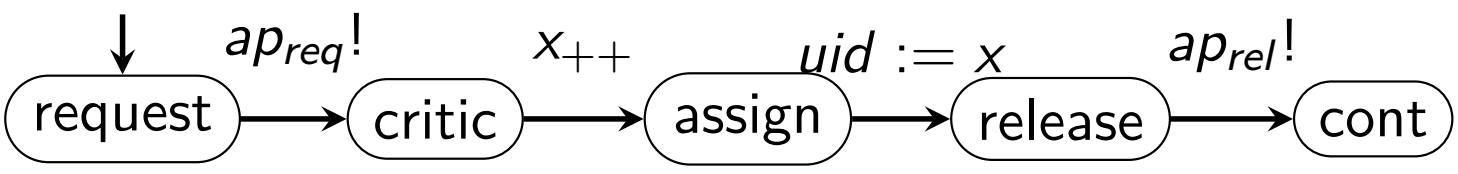
Spacetime programming

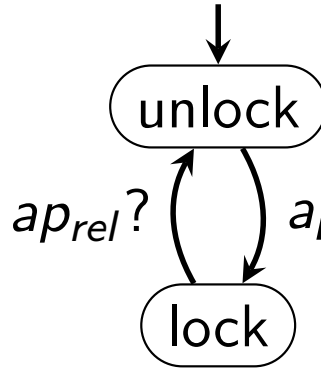
Difficulté de la mise en uvre : **double non-déterminisme**.

- Sur les branches du modèle.
- Dans chaque nud quand on résout le CSP.

```
module Solver =
  world_line VStore domains;
  world_line CStore constraints;
  proc search =
    par
    || propagation()
    || branch()
    end
  proc propagation =
    loop
      constraints.propagate(domains);
      pause;
    end
  proc branch =
    loop
      single_time IntVar x =
        fail_first (domains);
      single_time Integer v =
        middle_value(x);
      space constraints <- x.leq(v);
      space constraints <- x.ge(v);
      pause;
    end
end
```

Contraintes globales pour capturer des propriétés





On peut utiliser `distinct([`$uid_1, \ldots, uid_n$`])` pour s'assurer que les UIDs sont uniques.

Conclusion

- Model-checking et CP ne sont pas incompatibles.
  - Pour le model-checking : propagation (efficacité).
  - Pour la CP : construction dynamique d'un modèle.
- Utilisation du paradigme spacetime.

 `github.com/ptal/bonsai`

 `github.com/ptal/bonsai-model-checking`

Merci de votre attention