Vérification par la preuve de propriétés temporelles sur des programmes C

Travail financé par le projet PFC du pôle de compétitivité System@tic.

Julien Groslambert[a] Nicolas Stouls[b]

[a] Trusted-Labs
Julien.Groslambert@trusted-labs.com

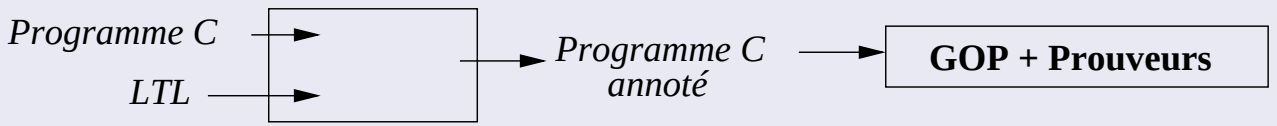
[b] Projet ProVal, INRIA Saclay – Île-de-France
INSA Lyon, CITI
Nicolas.Stouls@inria.fr

AFADL 2009

Introduction et état de l'art

Contexte

- Propriétés temporelles
- Génération d'annotations de programmes
- Outil Frama-C *(Vérification de programmes C annotés)*
- Projet PFC *(Plate-Formes de Confiance, pôle System@tic)*



État de l'art

- [Trentelman *et al.* - AMAST'02]
  Génération d'annotations JML à partir de propriétés JTPL
  Exemple de limitation JTPL : $(x = 0) \implies \circ(y = 3)$
- [Groslambert - Thèse 2007]
  Extension à la LTL + Preuve de correction
  Orienté vérification par animation (sous contexte)

Objectifs

Objectif

- Conformité de programmes C v-à-v de propriétés LTL
- Favoriser l'automatisation de la preuve
- *Pour l'instant : composantes sûreté + vivacité finie*

Difficultés

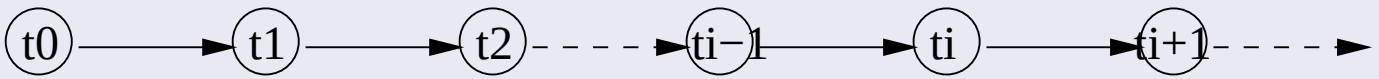
- Insuffisance des hypothèses pour la preuve
- Explosion combinatoire des OP *(nombre et complexité)*
- Programmes C *(appels imbriqués ou récursifs)*

Plan

Correction programme VS formule LTL

Principe *(Composante sûreté)*

- Programme : ensemble $\text{PATH}_{\text{Prog}}$ de traces d'exécution



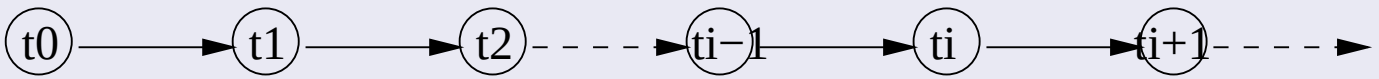
- LTL : ensemble $\text{PATH}_{\text{Büchi}}$ de chemins de l'automate de Büchi
- Vérification : acceptation de toute trace d'exécution

$$\forall t \in \text{PATH}_{\text{Prog}} \cdot \exists c \in \text{PATH}_{\text{Büchi}} \cdot \forall i \cdot t_i \models P_i(c)$$

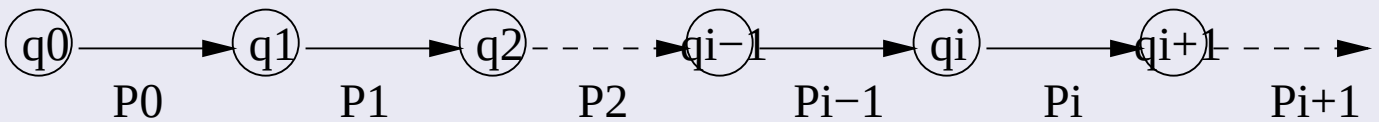
Correction programme VS formule LTL

Principe *(Composante sûreté)*

- Programme : ensemble $\text{PATH}_{\text{Prog}}$ de traces d'exécution



- LTL : ensemble $\text{PATH}_{\text{Büchi}}$ de chemins de l'automate de Büchi



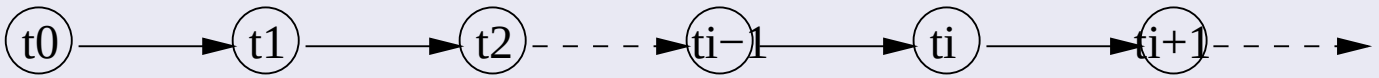
- Vérification : acceptation de toute trace d'exécution

$$\forall t \in \text{PATH}_{\text{Prog}} \cdot \exists c \in \text{PATH}_{\text{Büchi}} \cdot \forall i \cdot t_i \models P_i(c)$$

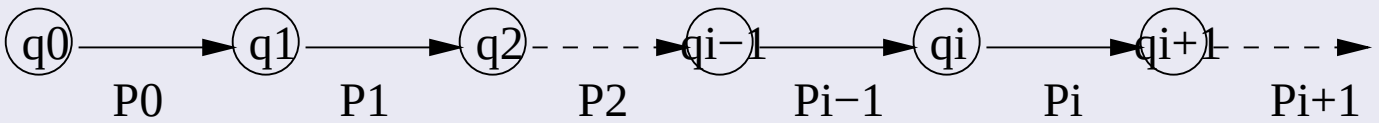
Correction programme VS formule LTL

Principe *(Composante sûreté)*

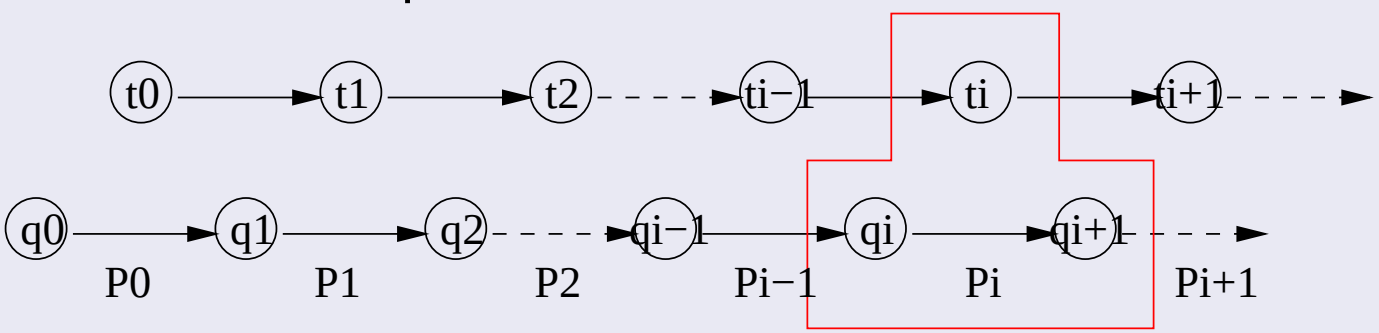
- Programme : ensemble $\text{PATH}_{\text{Prog}}$ de traces d'exécution



- LTL : ensemble $\text{PATH}_{\text{Büchi}}$ de chemins de l'automate de Büchi



- Vérification : acceptation de toute trace d'exécution



$$\forall t \in \text{PATH}_{\text{Prog}} \cdot \exists c \in \text{PATH}_{\text{Büchi}} \cdot \forall i \cdot t_i \models P_i(c)$$

Application à la vérification par la preuve

Forme générale des OP générées

H_1	/* Programme et propriété précédemment synchronisés */
$H_{2\ldots n}$	/* Hypothèses liées à la mise à jour de l'automate */
$H_{n+1\ldots m}$	/* Hypothèses liées au corps de l'opération */
But	/* Programme et propriété restent synchronisés */

Application à la vérification par la preuve

Forme générale des OP générées

H_1 $\quad \bigvee_{i=0}^{NbÉtats-1} \backslash \mathbf{old}(États[i]) \neq 0$

$H_{2\ldots n}$ $\quad$ */* Hypothèses liées à la mise à jour de l'automate */*

$H_{n+1\ldots m}$ $\quad$ */* Hypothèses liées au corps de l'opération */*

But $\quad \bigvee_{i=0}^{NbÉtats-1} États[i] \neq 0$

Application à la vérification par la preuve

Forme générale des OP générées

H_1	$\bigvee_{i=0}^{NbÉtats-1} \backslash\mathbf{old}(États[i]) \neq 0$
$H_{2\ldots n}$	/* Hypothèses liées à la mise à jour de l'automate */
$H_{n+1\ldots m}$	/* Hypothèses liées au corps de l'opération */
But	$\bigvee_{i=0}^{NbÉtats-1} États[i] \neq 0$

Difficultés liées à la preuve automatique

- $H_{n+1\ldots m}$ difficilement exploitables
- H_1 trop faible
- Explosion du nombre d'OP avec le nombre d'états

Application à la vérification par la preuve

Forme générale des OP générées

H_1	$\bigvee_{i=0}^{NbÉtats-1} \backslash\mathbf{old}(États[i]) \neq 0$
$H_{2\ldots n}$	*(* Hypothèses liées à la mise à jour de l'automate *)*
$H_{n+1\ldots m}$	*(* Hypothèses liées au corps de l'opération *)*
But	$\bigvee_{i=0}^{NbÉtats-1} États[i] \neq 0$

Propositions

- Créer des liens Büchi ($H_{n+1\ldots m}$) – Prog ($H_{2\ldots n}$)
  ($H_{m+1\ldots k}$: Spécification de l'automate et de ses transitions)
- Limiter les disjonctions *But* et H_1
  (Analyse statique des états inatteignables)

Création de liens Büchi-Prog

Principe *(en 3 points)*

- Axiomatisation de l'automate
- Mémorisation des transitions franchies
- Ajout d'invariants de liaison automate-programme

Mise en œuvre

```
/*@ axiomatic transStart{ logic integer transStart(integer tr) ;
  @   axiom transStart_0 : transStart(0) = /* État de départ de la transition 0 */ ;
  @   axiom transStart_1 : transStart(1) = /* État de départ de la transition 1 */ ;
  @   ...}
  @ axiomatic transStop{ logic integer transStop(integer tr) ;
  @   axiom transStop_0 : transStop(0) = /* État d'arrivée de la transition 0 */ ;
  @   ...}
  @ predicate transCond(integer tr)=
  @   (tr = 0 ⇒ .../* Condition de la transition 0 */) ∧
  @   (tr = 1 ⇒ .../* Condition de la transition 1 */) ∧   ...  */
```

Création de liens Büchi-Prog

Principe *(en 3 points)*

- Axiomatisation de l'automate
- Mémorisation des transitions franchies
- Ajout d'invariants de liaison automate-programme

Mise en œuvre

- Ajout d'une variable *Trans*
- Redondance partielle avec *États (simplification des OP)*
- Cohérence *Trans* / *États* préservée par invariant

Création de liens Büchi-Prog

Principe *(en 3 points)*

- Axiomatisation de l'automate
- Mémorisation des transitions franchies
- Ajout d'invariants de liaison automate-programme

Mise en œuvre

/*@ **global invariant** Non-franchissabilité :
@ $\forall tr; tr \in 0..NbTrans \wedge (\acute{E}tats_{Old}[transStart(tr)] = 0 \vee \neg transCond(tr))$
@ $\Rightarrow Trans[tr] = 0;$
@
@ **global invariant** Non-atteignabilité :
@ $\forall st; st \in 0..NbÉtats \wedge$
@ $(\forall tr; tr \in 0..NbTrans \Rightarrow Trans[tr] = 0 \vee transStop(tr) \neq st)$
@ $\Rightarrow \acute{E}tats[st] = 0;$ */

Première approximation des pré/post-conditions

- Approche : sur-approximation depuis la propriété :
  - Retrait des transitions/états syntaxiquement interdits
    - *Abstraction des expressions (sur variables)*
    - *Exploitation des prédicats (sur nom d'opérations)*
  - Pré-condition de *main* : état initial de la propriété
  - Post-condition de *main* : états d'acceptation de la propriété

Algo de construction pour chaque opération *Op*

$Pré(Op).Trans \leftarrow \{tr \mid \text{l'appel d'}Op\text{ n'est pas interdit par }tr\}$
$Pré(Op).États \leftarrow transStop[Pré(Op).Trans]$

$Post(Op).Trans \leftarrow \{tr \mid \text{le retour d'}Op\text{ n'est pas interdit par }tr\}$
$Post(Op).États \leftarrow transStop[Post(Op).Trans]$

Propagation statique des contraintes

Différentes méthodes mises en œuvre

1. Interprétation abstraite avant-arrière
2. Restriction aux cas d'utilisation
3. Spécification des boucles
4. Raffinement des post-conditions

Mise en œuvre

Propagation statique des contraintes

Différentes méthodes mises en œuvre

1. Interprétation abstraite avant-arrière
2. Restriction aux cas d'utilisation
3. Spécification des boucles
4. Raffinement des post-conditions

Mise en œuvre

- Propagation des spécifications approximées
- Abstraction des expressions
- IA avant : limitation des post-conditions aux états atteignables
  $Post(Op) \leftarrow Post(Op) \cap_2 Fwd_{Op}(Pré(Op), body(Op))$
- IA arrière : limitation des pré-conditions aux états non-divergents
  $Pré(Op) \leftarrow Pré(Op) \cap_2 Backward_{Op}(Post(Op), body(Op))$

Propagation statique des contraintes

Différentes méthodes mises en œuvre

1. Interprétation abstraite avant-arrière
2. Restriction aux cas d'utilisation
3. Spécification des boucles
4. Raffinement des post-conditions

Mise en œuvre

- A chaque passe : tous les appels sont observés
- Recensement des états d'appel de chaque opération
- Restriction des specs aux cas d'utilisation
- Point fixe des spécifications termine

Propagation statique des contraintes

Différentes méthodes mises en œuvre

1. Interprétation abstraite avant-arrière
2. Restriction aux cas d'utilisation
3. Spécification des boucles
4. Raffinement des post-conditions

Mise en œuvre

- Problèmes :
  - Propager les pré/post à travers la boucle
  - Annoter la boucle pour rendre la vérification possible
    (Induction d'un invariant en terme des états/transitions de l'automate)

Propagation statique des contraintes

Différentes méthodes mises en œuvre

1. Interprétation abstraite avant-arrière
2. Restriction aux cas d'utilisation
3. Spécification des boucles
4. Raffinement des post-conditions

Mise en œuvre

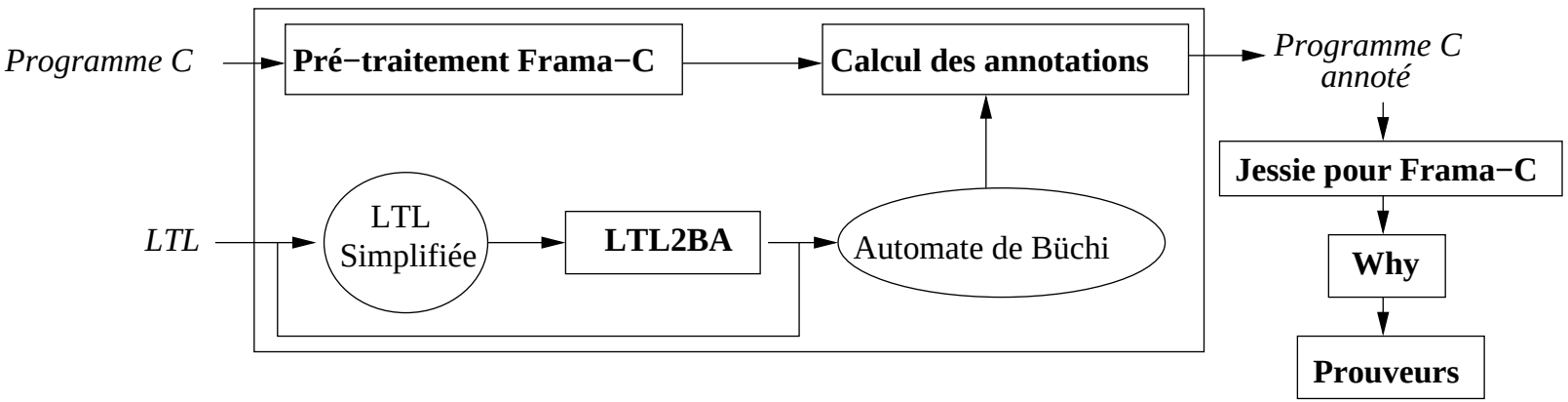
- Post-conditions en terme des états d'entrée
- Exemple :

behavior buch_0 :
  assumes *États*[0] $\neq 0$
  ensures . . . */* Post-condition liée à l'état d'entrée 0 */*
behavior buch_1 :
  assumes *États*[1] $\neq 0$
  ensures . . . */* Post-condition liée à l'état d'entrée 1 */*

- Intérêt :
  Appels de fonctions depuis différents contextes

Plug-in Aoraï

- Implante les algos présentés
- Intégré dans Frama-C *(CEA-List + INRIA-Saclay)*
- Conversion LTL vers Büchi : LTL2BA *(LSV)*
- Génération OP+Preuve : Plug-in Jessie *(INRIA-Saclay)*
  - + Why *(INRIA-Saclay)*
  - + Simplify *(Compaq)*
  - + Alt-Ergo *(INRIA-Saclay)*
  - + . . .



Bilan

Résultats obtenus

- Vérification de propriétés LTL *(sûreté + vivacité finie)*
- Automatisation des preuves
  - Ajout d'hypothèses de lien Automate-Programme
  - Simplifications statiques
- Validation de l'approche par le développement d'un outil

Comparaison avec l'approche [Pavlova *et al.* - CARDIS'04]

Automates de propriétés et propagation des contraintes

- Propagation des contraintes après génération des annotations
- Propagation : union des pré/post accessibles
- Propriétés considérées semblent déterministes
- Vérification de la spécification externe de programmes

Bilan

Résultats obtenus

- Vérification de propriétés LTL *(sûreté + vivacité finie)*
- Automatisation des preuves
  - Ajout d'hypothèses de lien Automate-Programme
  - Simplifications statiques
- Validation de l'approche par le développement d'un outil

Comparaison avec l'approche [Pavlova *et al.* - CARDIS'04]

Automates de propriétés et propagation des contraintes

- Propagation des contraintes après génération des annotations
- Propagation : union des pré/post accessibles
- Propriétés considérées semblent déterministes
- Vérification de la spécification externe de programmes

Travaux futurs

Renforcement de la précision des annotations

- Ajout d'heuristiques d'IA *(élargissement + graphe d'appel)*
- Diminuer l'abstraction de l'IA
- Post-conditions sous hypothèses de valeurs des variables

Étendre à la composante vivacité

- Travaux initiaux prennent en compte la vivacité *(variant)*
- Exploiter les mécanismes de variant/decreases d'ACSL
- Joindre des techniques d'animation ? *(atteignabilité)*

Merci de votre attention

Démonstration de l'outil possible aux pauses

Annexe : Propagation statique des contraintes

Interprétation abstraite avant

$Fwd_{Op}(P, [\,]) \hat{=} P$

$Fwd_{Op}(P, (x = E) :: L) \hat{=} Fwd_{Op}(P, L)$

$Fwd_{Op}(P, Call(Op2) :: L) \hat{=} Fwd_{Op}(Post(Op2), L)$

$Fwd_{Op}(P, (\texttt{IF}\ (c)\ B_1\ \texttt{ELSE}\ B_2) :: L) \hat{=}$
$\quad$ **let** $p_1, p_2 = Fwd_{Op}(P, B_1), Fwd_{Op}(P, B_2)$ **in** $Fwd_{Op}(p_1 \cup_2 p_2, L)$

$Fwd_{Op}(P, (\texttt{return}\ e) :: []) \hat{=}$
$\quad (Trans = \{tr \mid transStart(tr) \in P.États \wedge$ le retour d'Op peut franchir $tr\}$,
$\quad\ États = transStop[Trans])$

Annexe : Propagation statique des contraintes

Spécification des boucles 1/2

Forme générale d'une boucle dans Frama-C :

$\{pré_{boucle}\}$
while(1) {
  $\{pré_{corps}\}$
  if(c) **goto** LabelEndLoop ;
  . . . /* *Corps de la boucle* */
  $\{post_{corps}\}$
}
LabelEndLoop :
$\{post_{boucle}\}$

Annexe : Propagation statique des contraintes

Spécification des boucles 2/2

- Pré/post de la boucle connus
- Construction des pré/post du corps par IA avant-arrière
- Propagation des contraintes du corps vers la spec de boucle
- Invariant construit :
  ll@ **Loop invariant** i : $(Init \Rightarrow Pré_{boucle}) \wedge (\neg Init \Rightarrow Post_{corps})$
  Init : variable fraîche identifiant la première itération.