Élimination de quantificateurs : extrapolation efficace par analyse structurelle et cœurs insatisfiables

Rémi Delmas (ONERA)
Adrien Champion (Rockwell Collins France)

MSR'2013 – 13 Novembre 2013

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Systèmes de transitions symboliques

Nous considérons un système réactif modélisé via un système de transition symbolique

$$S = \langle I(v), T(v, v') \rangle$$

où I et T sont deux prédicats[a] dits *d'initialisation* et *de transition* exprimés par des sur les *variables d'état* v (v'représente l'état suivant dans la transition).

[a] dans une logique adaptée, cf. reste de la présentation

Élimination de quantificateurs

Soient :

- $\mathcal{F}$ une formule logique *quantifier-free*,
- $v \in \mathit{FreeVars}(\mathcal{F})$,

Élimination de quantificateurs

Soient :

- $\mathcal{F}$ une formule logique *quantifier-free*,
- $v \in \mathit{FreeVars}(\mathcal{F})$,

L'*élimination de quantificateur* est l'operation

$$QE(v, \mathcal{F})$$

Élimination de quantificateurs

Soient :

- $\mathcal{F}$ une formule logique *quantifier-free*,
- $v \in \mathit{FreeVars}(\mathcal{F})$,

L'*élimination de quantificateur* est l'operation

$$QE(v, \mathcal{F})$$

Telle que:

- $QE(v, \mathcal{F}) = \mathcal{O}$
- $\mathcal{O} \equiv \exists v,\ \mathcal{F}$

Élimination de quantificateurs

Soient :

- $\mathcal{F}$ une formule logique *quantifier-free*,
- $v \in \mathit{FreeVars}(\mathcal{F})$,

L'*élimination de quantificateur* est l'operation

$$QE(v, \mathcal{F})$$

Telle que:

- $QE(v, \mathcal{F}) = \mathcal{O}$
- $\mathcal{O} \equiv \exists v,\ \mathcal{F}$

Exemple

- $\mathcal{F} \equiv (x = 39 \wedge y \geq 3 + x) \vee (x \geq 7 \wedge y = 11 - x)$
- $QE(x, \mathcal{F}) \equiv y \geq 42 \ \vee \ y \leq 4.$

Vérification d'invariants par calcul de préimage

Pour vérifier qu'une propriété $PO(v)$ est un invariant du système, on peut itérer le calcul suivant:

$$\mathcal{G}_1(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \neg PO(v'))$$
$$\mathcal{G}_{i+1}(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \mathcal{G}_i(v'))$$

qui calcule l'espace des état co-atteignables depuis $\neg PO$.

Vérification d'invariants par calcul de préimage

Pour vérifier qu'une propriété $PO(v)$ est un invariant du système, on peut itérer le calcul suivant:

$$\mathcal{G}_1(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \neg PO(v'))$$
$$\mathcal{G}_{i+1}(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \mathcal{G}_i(v'))$$

qui calcule l'espace des état co-atteignables depuis $\neg PO$.
Si on atteint un n tel que

$$\bigvee_{i \in [1,n+1]} G_i \rightarrow \bigvee_{i \in [1,n]} G_i$$

Vérification d'invariants par calcul de préimage

Pour vérifier qu'une propriété $PO(v)$ est un invariant du système, on peut itérer le calcul suivant:

$$\mathcal{G}_1(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \neg PO(v'))$$
$$\mathcal{G}_{i+1}(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \mathcal{G}_i(v'))$$

qui calcule l'espace des état co-atteignables depuis $\neg PO$.
Si on atteint un n tel que

$$\bigvee_{i \in [1,n+1]} G_i \rightarrow \bigvee_{i \in [1,n]} G_i$$

et que

$$I(v) \wedge \bigvee_{i \in [1,n+1]} G_i(v)$$

est UNSAT,

Vérification d'invariants par calcul de préimage

Pour vérifier qu'une propriété $PO(v)$ est un invariant du système, on peut itérer le calcul suivant:

$$\mathcal{G}_1(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \neg PO(v'))$$
$$\mathcal{G}_{i+1}(v) \equiv QE(v', PO(v) \wedge T(v, v') \wedge \mathcal{G}_i(v'))$$

qui calcule l'espace des état co-atteignables depuis $\neg PO$.
Si on atteint un n tel que

$$\bigvee_{i \in [1,n+1]} G_i \rightarrow \bigvee_{i \in [1,n]} G_i$$

et que

$$I(v) \wedge \bigvee_{i \in [1,n+1]} G_i(v)$$

est UNSAT, alors $PO(v)$ est un invariant du système.

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Quantifier free linear real integer arithmetic (QF_LRIA)

Quantifier free linear real integer arithmetic (QF_LRIA)

$$
\begin{aligned}
B &::= \top \mid \bot \mid A \mid \neg B \mid B \wedge B \mid B \vee B \\
A &::= \langle boolId \rangle \mid R = R \mid R < R \mid I = I \mid I < I \\
I &::= \langle intId \rangle \mid \langle intCst \rangle \mid I - I \mid I + I \\
R &::= \langle realId \rangle \mid \langle realCSt \rangle \mid \langle realCst \rangle * R \mid R - R \mid R + R
\end{aligned}
\tag{1}
$$

Quantifier free linear real integer arithmetic (QF_LRIA)

$$
\begin{aligned}
B &::= \top \mid \bot \mid A \mid \neg B \mid B \wedge B \mid B \vee B \\
A &::= \langle boolId \rangle \mid R = R \mid R < R \mid I = I \mid I < I \\
I &::= \langle intId \rangle \mid \langle intCst \rangle \mid I - I \mid I + I \\
R &::= \langle realId \rangle \mid \langle realCSt \rangle \mid \langle realCst \rangle * R \mid R - R \mid R + R
\end{aligned}
\tag{1}
$$

Avec:

- $a \implies b \equiv \neg a \vee b$
- $a = b \equiv (a \implies b) \wedge (b \implies a)$

Quantifier free linear real integer arithmetic (QF_LRIA)

$$
\begin{aligned}
B &::= \top \mid \bot \mid A \mid \neg B \mid B \wedge B \mid B \vee B \\
A &::= \langle boolId \rangle \mid R = R \mid R < R \mid I = I \mid I < I \\
I &::= \langle intId \rangle \mid \langle intCst \rangle \mid I - I \mid I + I \\
R &::= \langle realId \rangle \mid \langle realCSt \rangle \mid \langle realCst \rangle * R \mid R - R \mid R + R
\end{aligned}
\tag{1}
$$

Avec:

- $a \implies b \equiv \neg a \vee b$
- $a = b \equiv (a \implies b) \wedge (b \implies a)$

En somme:
Booléens + arithmétique linéaire réelle + octogones entiers.

Littéral

$$l_i \equiv a$$
$$l_i \equiv \neg a$$

Littéral

$$\begin{aligned} l_i &\equiv a \\ l_i &\equiv \neg a \end{aligned}$$

Clause, forme normale conjonctive (CNF)

$$\begin{aligned} C_i &\equiv l_1 \vee \cdots \vee l_n \\ \mathcal{F} &\equiv C_1 \wedge \cdots \wedge C_m \end{aligned}$$

Littéral

$$\begin{aligned} l_i &\equiv a \\ l_i &\equiv \neg a \end{aligned}$$

Clause, forme normale conjonctive (CNF)

$$\begin{aligned} C_i &\equiv l_1 \vee \cdots \vee l_n \\ \mathcal{F} &\equiv C_1 \wedge \cdots \wedge C_m \end{aligned}$$

Cube, forme normale disjonctive (DNF)

$$\begin{aligned} K_i &\equiv l_1 \wedge \cdots \wedge l_n \\ \mathcal{F} &\equiv K_1 \vee \cdots \vee K_m \end{aligned}$$

Fonction d'évaluation

Une *fonction d'évaluation* M pour une formule $\mathcal{F}$ associe:

Fonction d'évaluation

Une *fonction d'évaluation* M pour une formule $\mathcal{F}$ associe:

- un littéral booléen (resp. entier, réel) à tout identificateur booléen (resp. entier, réel) de $\mathcal{F}$,

Fonction d'évaluation

Une *fonction d'évaluation* M pour une formule $\mathcal{F}$ associe:

- un littéral booléen (resp. entier, réel) à tout identificateur booléen (resp. entier, réel) de $\mathcal{F}$,
- et telle que :

$$\begin{aligned} M(\neg a) &\equiv \neg M(a) \\ M(a \ \langle op \rangle \ b) &\equiv M(a) \ \langle op \rangle \ M(b) \\ &\text{pour } \langle op \rangle \in \{\wedge, \vee, +, -, =, <\} \end{aligned} \tag{2}$$

où les opérateurs prennent leur sémantique naturelle sur les littéraux booléens, entiers et réels.

Fonction d'évaluation

Une *fonction d'évaluation* M pour une formule $\mathcal{F}$ associe:

- un littéral booléen (resp. entier, réel) à tout identificateur booléen (resp. entier, réel) de $\mathcal{F}$,
- et telle que :

$$\begin{aligned} M(\neg a) &\equiv \neg M(a) \\ M(a \ \langle op \rangle \ b) &\equiv M(a) \ \langle op \rangle \ M(b) \\ &\text{pour } \langle op \rangle \in \{\wedge, \vee, +, -, =, <\} \end{aligned} \tag{2}$$

où les opérateurs prennent leur sémantique naturelle sur les littéraux booléens, entiers et réels.

Modèle d'une formule

Un *modèle* pour une formule $\mathcal{F}$ est une fonction d'évaluation telle que $M(\mathcal{F}) \equiv \top$.

QE par enumération paresseuse de modèles (Monniaux [10])

Algorithm 1 QE par énumération paresseuse de modèles.

Require: $\mathcal{F}$: une formule QF_LRIA
Require: V : collection de variables à éliminer de $\mathcal{F}$.
Ensure: $\mathcal{O}$: une formule en forme DNF telle que $\mathcal{O} \equiv \exists V, \mathcal{F}$

```
function QE(V, F)
    O ← ⊥
    while isSatisfiable(F ∧ ¬O) do
        M ← getModel(F ∧ ¬O)          ▷ génération d'un modèle
        E ← extrapolate(F, M)         ▷ extrapolation du modèle
        P ← project(E, V)             ▷ projection de l'extrapolant
        O ← O ∨ P                     ▷ construction incrémentale du résultat
    end while
    return O                          ▷ retrouner la formule DNF
end function
```

Détails de l'algorithme

Détails de l'algorithme

$getModel(\mathcal{F} \wedge \neg\mathcal{O})$: utilise un solveur SMT pour trouver un modèle de $\mathcal{F} \wedge \neg\mathcal{O}$ tant qu'il en existe, sinon l'algorithme termine.

Détails de l'algorithme

$getModel(\mathcal{F} \wedge \neg\mathcal{O})$: utilise un solveur SMT pour trouver un modèle de $\mathcal{F} \wedge \neg\mathcal{O}$ tant qu'il en existe, sinon l'algorithme termine.

$extrapolate(\mathcal{F}, M)$ produit à partir d'un modèle M de $\mathcal{F}$ une nouvelle formule $\mathcal{E}$ qui se veut plus générale que le modèle dont elle est issue et qui implique $\mathcal{F}$

Détails de l'algorithme

getModel$(\mathcal{F} \wedge \neg\mathcal{O})$: utilise un solveur SMT pour trouver un modèle de $\mathcal{F} \wedge \neg\mathcal{O}$ tant qu'il en existe, sinon l'algorithme termine.

extrapolate$(\mathcal{F}, M)$ produit à partir d'un modèle M de $\mathcal{F}$ une nouvelle formule $\mathcal{E}$ qui se veut plus générale que le modèle dont elle est issue et qui implique $\mathcal{F}$

project$(\mathcal{E}, V)$ élimine de $\mathcal{E}$ les variables de V par une projection polyédrique, fournie par exemple par la Parma Polyhedra Library [2].

Détails de l'algorithme

$getModel(\mathcal{F} \wedge \neg\mathcal{O})$: utilise un solveur SMT pour trouver un modèle de $\mathcal{F} \wedge \neg\mathcal{O}$ tant qu'il en existe, sinon l'algorithme termine.

$extrapolate(\mathcal{F}, M)$ produit à partir d'un modèle M de $\mathcal{F}$ une nouvelle formule $\mathcal{E}$ qui se veut plus générale que le modèle dont elle est issue et qui implique $\mathcal{F}$

$project(\mathcal{E}, V)$ élimine de $\mathcal{E}$ les variables de V par une projection polyédrique, fournie par exemple par la Parma Polyhedra Library [2].

Le résultat de chaque projection nourrit la construction de la formule DNF résultat $\mathcal{O}$ qui, lorsque $\mathcal{F} \wedge \neg\mathcal{O}$ n'admet plus de modèle, est équivalente à $(\exists V,\ \mathcal{F})$ et ne contient plus aucune variable de V.

Extrapolant [1]

Soient deux formules A et B. Une formule C est un extrapolant de A et B si elle est telle que :

- si $A \wedge B$ est **unsat** alors $C \equiv \bot$
- si $A \wedge B$ est **sat** alors $A \wedge C$ est **sat** et $C \wedge \neg B$ est **unsat**.

Extrapolant [1]

Soient deux formules A et B. Une formule C est un extrapolant de A et B si elle est telle que :

- si $A \wedge B$ est **unsat** alors $C \equiv \bot$
- si $A \wedge B$ est **sat** alors $A \wedge C$ est **sat** et $C \wedge \neg B$ est **unsat**.

Extrapolants dans l'élimination de quantificateurs

Si on pose:

- $M \models \mathcal{F}$,
- $A \equiv (\bigwedge_{v \in Vars(\mathcal{F})} v = M(v))$,
- $B \equiv \mathcal{F}$,

Extrapolant [1]

Soient deux formules A et B. Une formule C est un extrapolant de A et B si elle est telle que :

- si $A \wedge B$ est **unsat** alors $C \equiv \bot$
- si $A \wedge B$ est **sat** alors $A \wedge C$ est **sat** et $C \wedge \neg B$ est **unsat**.

Extrapolants dans l'élimination de quantificateurs

Si on pose:

- $M \models \mathcal{F}$,
- $A \equiv (\bigwedge_{v \in Vars(\mathcal{F})} v = M(v))$,
- $B \equiv \mathcal{F}$,

Alors un extrapolant $\mathcal{E}$ est tel que :

- $A \wedge \mathcal{E}$ est SAT, *i.e.* $M(\mathcal{E}) \equiv M(\mathcal{F}) \equiv \top$, *i.e.* $M \models \mathcal{E}$;
- $\mathcal{E} \wedge \neg \mathcal{F}$ est UNSAT, *i.e.* $\mathcal{E} \implies \mathcal{F}$.

Minimisation SMT-test

- On pose $\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$

Minimisation SMT-test

- On pose $\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$
- Minimisation: Pour chaque $l' \in \mathcal{L}$, on teste la satisfiabilité de $((\bigwedge_{l \in \mathcal{L},\, l \neq l'} l) \wedge \neg \mathcal{F})$: Si SAT, on retire l' de $\mathcal{L}$, sinon on le garde.

Minimisation SMT-test

- On pose $\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$
- Minimisation: Pour chaque $l' \in \mathcal{L}$, on teste la satisfiabilité de $((\bigwedge_{l \in \mathcal{L},\, l \neq l'} l) \wedge \neg \mathcal{F})$: Si SAT, on retire l' de $\mathcal{L}$, sinon on le garde.
- On forme l'extrapolant $(\bigwedge_{l \in \mathcal{L}_{min}} l)$.

Minimisation SMT-test

- On pose $\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$
- Minimisation: Pour chaque $l' \in \mathcal{L}$, on teste la satisfiabilité de $((\bigwedge_{l \in \mathcal{L},\, l \neq l'} l) \wedge \neg \mathcal{F})$: Si SAT, on retire l' de $\mathcal{L}$, sinon on le garde.
- On forme l'extrapolant $(\bigwedge_{l \in \mathcal{L}_{min}} l)$.

Remarques

- Remonter aux litéraux impliqués par les valeurs concrètes du modèle est une première forme de généralisation
- La minimisation poursuit cette généralisation

Questions étudiées dans ces travaux

Questions étudiées dans ces travaux

- Qu'est-ce qu'un bon extrapolant?
  - Nombre de littéraux?
  - Nombre de variables?
  - Temps de calcul?

Questions étudiées dans ces travaux

- Qu'est-ce qu'un bon extrapolant?
  - Nombre de littéraux?
  - Nombre de variables?
  - Temps de calcul?
- Peut-on tirer parti des capacités des solveurs SMT modernes pour calculer des extrapolants plus efficacement?
  - incrémentalité
  - cœurs insatisfiables

Questions étudiées dans ces travaux

- Qu'est-ce qu'un bon extrapolant?
  - Nombre de littéraux?
  - Nombre de variables?
  - Temps de calcul?
- Peut-on tirer parti des capacités des solveurs SMT modernes pour calculer des extrapolants plus efficacement?
  - incrémentalité
  - cœurs insatisfiables
- Peut-on adapter des analyses structurelles issues d'autres applications pour générer des extrapolants?
  - identifications de *causes minimales*?
  - fonction *d'explication*?

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Si on considère la formule:

$$\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$$

Cœurs insatisfiables minimaux

Si on considère la formule:

$$\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$$

Et que l'on considère un cœur insatisfiable minimal suivant:

$$\mathcal{U} \equiv unsatCore(((\bigwedge_{l \in \mathcal{L}} l) \wedge \neg \mathcal{F}))$$

Cœurs insatisfiables minimaux

Si on considère la formule:

$$\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$$

Et que l'on considère un cœur insatisfiable minimal suivant:

$$\mathcal{U} \equiv unsatCore(((\bigwedge_{l \in \mathcal{L}} l) \wedge \neg \mathcal{F}))$$

Alors le cube:

$$K \equiv \bigwedge_{l \in \mathcal{U} \subseteq \mathcal{L}} l$$

est un extrapolant optimal du point de vue de son nombre de littéraux.

Cœurs insatisfiables minimaux

Si on considère la formule:

$$\mathcal{L} \equiv \{\text{si } M(a) \text{ alors } a \text{ sinon } \neg a \mid a \in Atoms(\mathcal{F})\}$$

Et que l'on considère un cœur insatisfiable minimal suivant:

$$\mathcal{U} \equiv unsatCore(((\bigwedge_{l \in \mathcal{L}} l) \wedge \neg \mathcal{F}))$$

Alors le cube:

$$K \equiv \bigwedge_{l \in \mathcal{U} \subseteq \mathcal{L}} l$$

est un extrapolant optimal du point de vue de son nombre de littéraux.
Les solveurs SMT modernes peuvent calculer les cœurs insatisfiables des cœurs unsat minimaux.

Exprimentations

Les expérimentations ont montré que le gain par rapport a SMT-test n'était pas notable (on aurait pu s'en douter)

Exprimentations

Les expérimentations ont montré que le gain par rapport a SMT-test n'était pas notable (on aurait pu s'en douter)

- Le calcul des cœurs unsat impose un surcoût sur la resolution SMT (book-keeping)

Exprimentations

Les expérimentations ont montré que le gain par rapport a SMT-test n'était pas notable (on aurait pu s'en douter)

- Le calcul des cœurs unsat impose un surcoût sur la resolution SMT (book-keeping)
- Les cœurs unsat minimaux sont longs à calculer: on retombe (plus ou moins) sur SMT-test, mais côté solveur

Exprimentations

Les expérimentations ont montré que le gain par rapport a SMT-test n'était pas notable (on aurait pu s'en douter)

- Le calcul des cœurs unsat impose un surcoût sur la resolution SMT (book-keeping)
- Les cœurs unsat minimaux sont longs à calculer: on retombe (plus ou moins) sur SMT-test, mais côté solveur
- Par contre, les cœurs unsat non-minimaux sont virtuellement gratuits!

Exprimentations

Les expérimentations ont montré que le gain par rapport a SMT-test n'était pas notable (on aurait pu s'en douter)

- Le calcul des cœurs unsat impose un surcoût sur la resolution SMT (book-keeping)
- Les cœurs unsat minimaux sont longs à calculer: on retombe (plus ou moins) sur SMT-test, mais côté solveur
- Par contre, les cœurs unsat non-minimaux sont virtuellement gratuits!
  $\Rightarrow$ Adapter l'algorithme SMT-test pour utiliser de manière opportuniste les cœurs non-minimaux!

Algorithm 2 SMT-test avec cœurs insatisfiables.

Require: $\mathcal{F}$ une formule QF_LRIA, M un modèle de $\mathcal{F}$.
Ensure: Un extrapolant de $\overline{M}$ par rapport à $\mathcal{F}$ sous la forme d'un ensemble de littéraux.

```
function smtTestCores(F, M)
    toCheck ← L                          ▷ Ensemble L de littéraux à tester (cf. Eq ??)
    toKeep ← ∅
    while toCheck ≠ ∅ do                 ▷ ensemble des littéraux de l'extrapolant
        solver.assert(¬F)
        literal ← toCheck.first          ▷ prendre le prochain littéral à tester
        toCheck ← (toCheck \ {literal})  ▷ supprimer le littéral de l'ensemble
        for lit ∈ toKeep ∪ toCheck do
            solver.assert(lit)           ▷ supposer les littéraux à garder/tester
        end for
        if solver.checksat() = sat then
            toKeep ← (toKeep ∪ {literal})   ▷ ajouter le littéral dans l'ensemble résultat
        else                             ▷ supprimer les littéraux n'étant pas dans le cœur unsat
            toCheck ← solver.getUnsatCore() ∩ toCheck
        end if
        solver.restart()                 ▷ réinitialiser le solveur pour la prochaine itération
    end while
    return toKeep                        ▷ retourner l'ensemble de littéraux formant l'extrapolant
end function
```

Génération des causes d'une formule

Si on considère une formule $\mathcal{F}$ et un modèle $M \models \mathcal{F}$:

Génération des causes d'une formule

Si on considère une formule $\mathcal{F}$ et un modèle $M \models \mathcal{F}$:

- Il est possible de raisonner de proche en proche sur la structure de $\mathcal{F}$ à partir des valeurs de M modèle pour identifier quelles sont les sous formules pertinentes pour expliquer la valeur de $\mathcal{F}$.

Génération des causes d'une formule

Si on considère une formule $\mathcal{F}$ et un modèle $M \models \mathcal{F}$:

- Il est possible de raisonner de proche en proche sur la structure de $\mathcal{F}$ à partir des valeurs de M modèle pour identifier quelles sont les sous formules pertinentes pour expliquer la valeur de $\mathcal{F}$.
- Idée introduite dans [4] pour identifier les chemins de propagation de valeurs dans les modèles Simulink/SCADE, adaptée ici pour procéder à l'extrapolation.

Explications: exemple

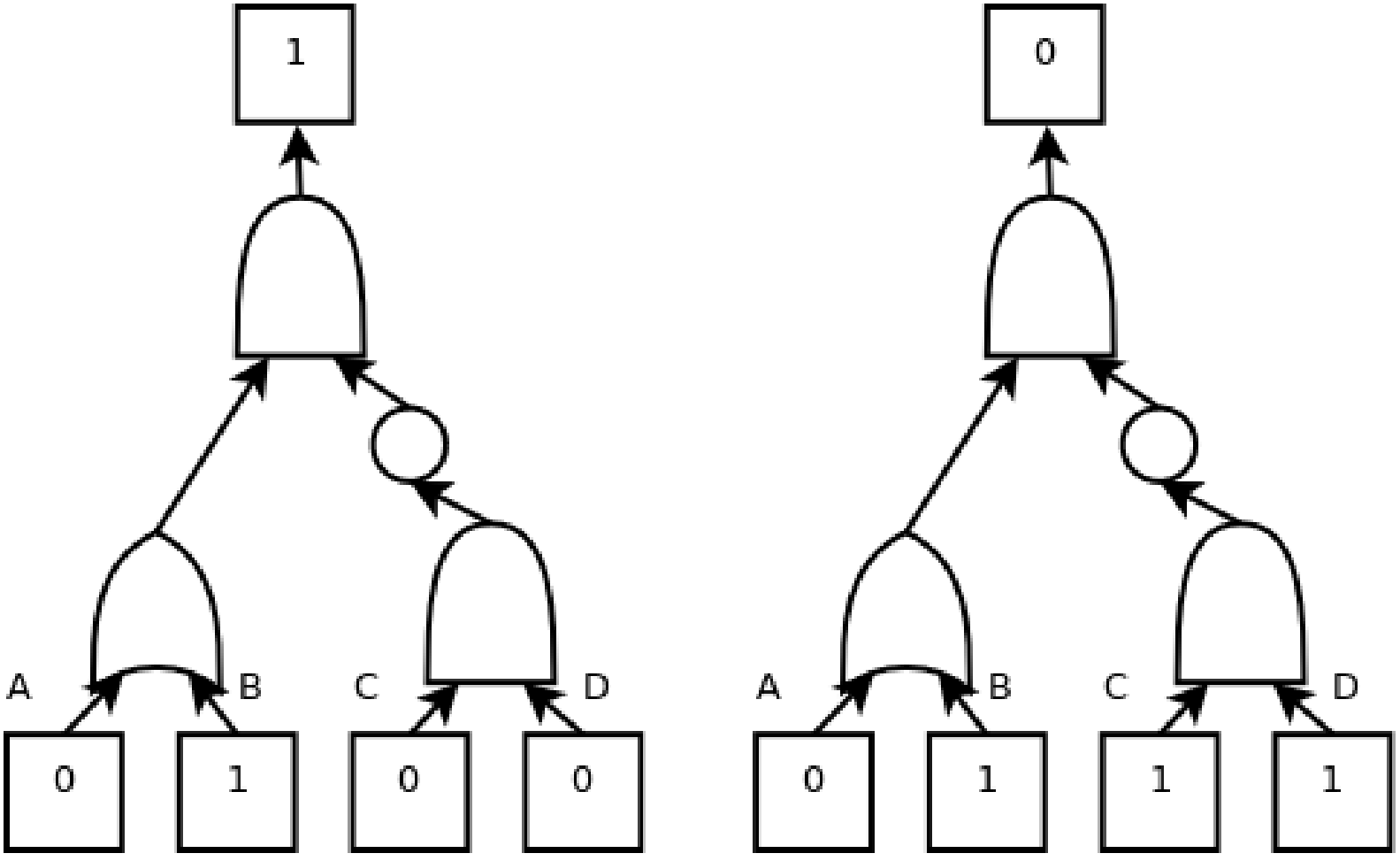
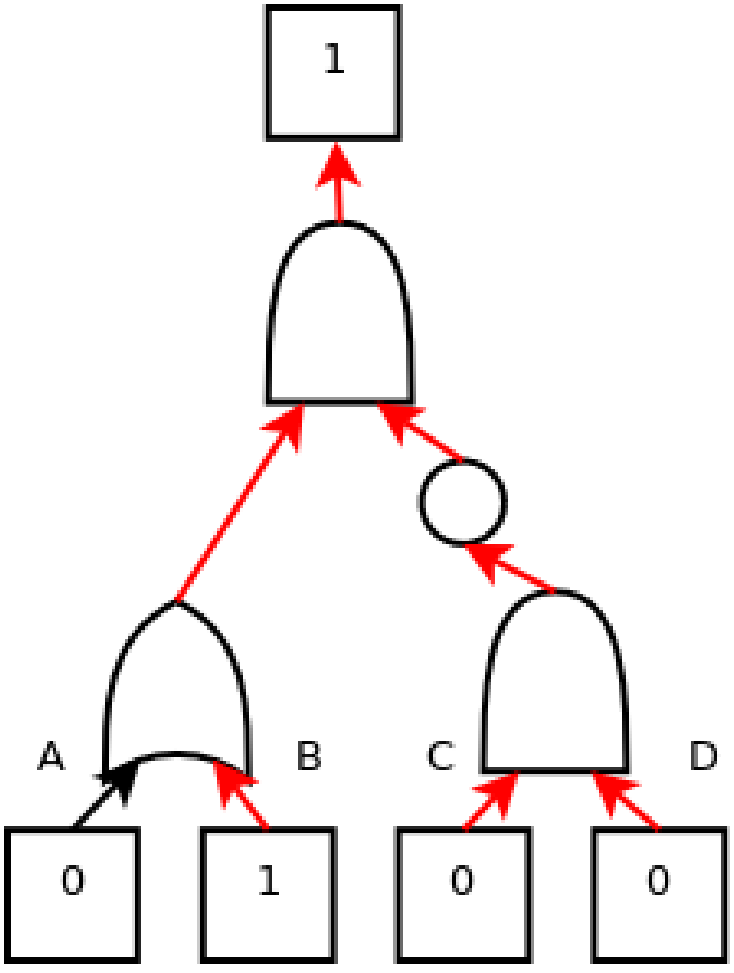
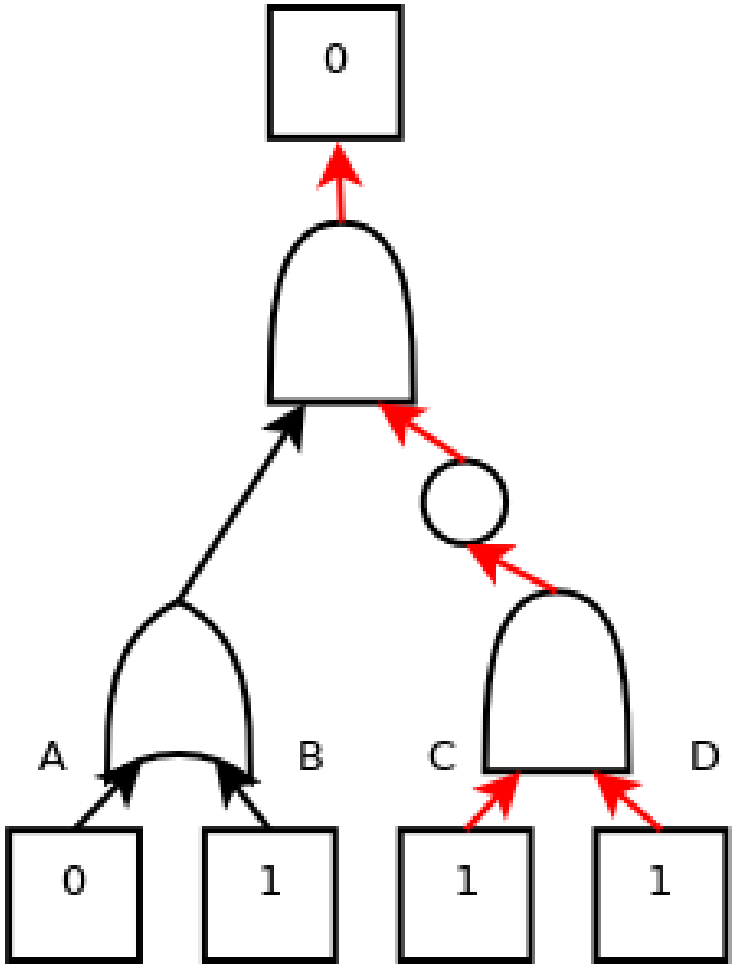


Figure: Une formule, deux modèles

Explications: exemple



$K_1 \equiv B \wedge \neg C \wedge \neg D$



$K_2 \equiv C \wedge D$

Figure: Chemins actifs et causes

Règles de calcul de l'analyse structurelle.

Terme	Condition	Sous-formules actives	Cause générée/ Combinaison
$\langle id \rangle$ booléen	$\top$	[]	$\langle id \rangle$ si $M(\langle id \rangle)$, $\neg\langle id \rangle$ sinon
$\neg a$	$\top$	$[a]$	$K(a)$
$a \wedge b$	$M(\neg a)$	$[a]$	$K(a)$
	$M(a \wedge \neg b)$	$[b]$	$K(b)$
	$M(a \wedge b)$	$[a, b]$	$K(a) \wedge K(b)$
$a \vee b$	$M(a)$	$[a]$	$K(a)$
	$M(\neg a \wedge b)$	$[b]$	$K(b)$
	$M(\neg a \wedge \neg b)$	$[a, b]$	$K(a) \wedge K(b)$
$x = y$	$M(x = y)$	[]	$(x = y)$
	$M(x < y)$	[]	$(x < y)$
	$M(y < x)$	[]	$(y < x)$
$x < y$	$M(x < y)$	[]	$(x < y)$
	$M(\neg(x < y))$	[]	$\neg(x < y)$

Algorithm 3 Extrapolation par méthode structurelle.

Require: f une formule, M une fonction d'évaluation (*cf.* Equation (2)).
Ensure: Si $M \models f$ alors le résultat est un extrapolant de M par rapport à f.

```
function struct(f, M)
    activeSubFormulas ← active(f, M)      ▷ sous-formules actives (cf. Tableau 4)
    subCauses ← activeSubFormulas map {x → struct(x, M)}    ▷ appel récursif
    return combine(f, subCauses, M)                ▷ combinaison (cf. Tableau 4)
end function
```

Proposition

- Utiliser la méthode structurelle pour restreindre l'ensemble de litéraux de départ dans l'algorithme SMT-test
- Utiliser les cœurs unsat non-minimaux
- Utiliser les possibilités incrémentales des solveurs

Algorithme hybride: structurel+SMT

Algorithm 4 $hybrid(\mathcal{F}, M)$ ▷ Extrapolation hybride.

Require: $\mathcal{F}$ une formule, M un modèle de $\mathcal{F}$.
Ensure: Un extrapolant de M par rapport à $\mathcal{F}$, sous la forme d'un ensemble de littéraux.

```
toCheck ← struct(F, M)        ▷ ensemble de littéraux par analyse structurelle (cf. Alg. 3)
toKeep ← ∅
solver.assert(¬F)             ▷ insérer ¬F au niveau 0
solver.push(1)                ▷ à niveau 1 (littéraux à garder)
while toCheck ≠ ∅ do
    literal ← toCheck.get     ▷ prendre un littéral dans l'ensemble toCheck
    toCheck ← (toCheck \ {literal})   ▷ le supprimer de l'ensemble
    solver.push(1)            ▷ à niveau 2 (littéral à tester)
    for lit ∈ toCheck do
        solver.assert(lit)    ▷ insérer le littéral à tester à niveau 2
    end for
    if solver.checksat() = sat then
        toKeep ← (toKeep ∪ {literal})
        solver.pop(1)
        solver.assert(literal)    ▷ insérer le littéral à garder à niveau 1
    else                          ▷ éliminer les littéraux hors du cœur insatisfiable
        solver.pop(1)
        toCheck ← (solver.getUnsatCore() ∩ toCheck)
    end if
end while
return toKeep
```

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Implémentation

- Solveur SMT: Microsoft Z3
- Projection: Parma Polyedra Library
- Extrapolation: nous mêmes[a]

[a] Available at https://cavale.enseeiht.fr/redmine/projects/smt-qe/files

Source des benchmarks

- Base de benchmarks Lustre de l'outil Kind (Univ. d'Iowa)[a]
- Calcul des 20 premières pré-images: $\Rightarrow$ 2800 jobs de QE[b]
- Temps total de calcul pour chaque job de QE
- Nombre de cubes dans la formule DNF obtenue

[a] http://clc.cs.uiowa.edu/Kind/index.php?page=experimental-results.
[b] https://cavale.enseeiht.fr/redmine/projects/smt-qe/files.

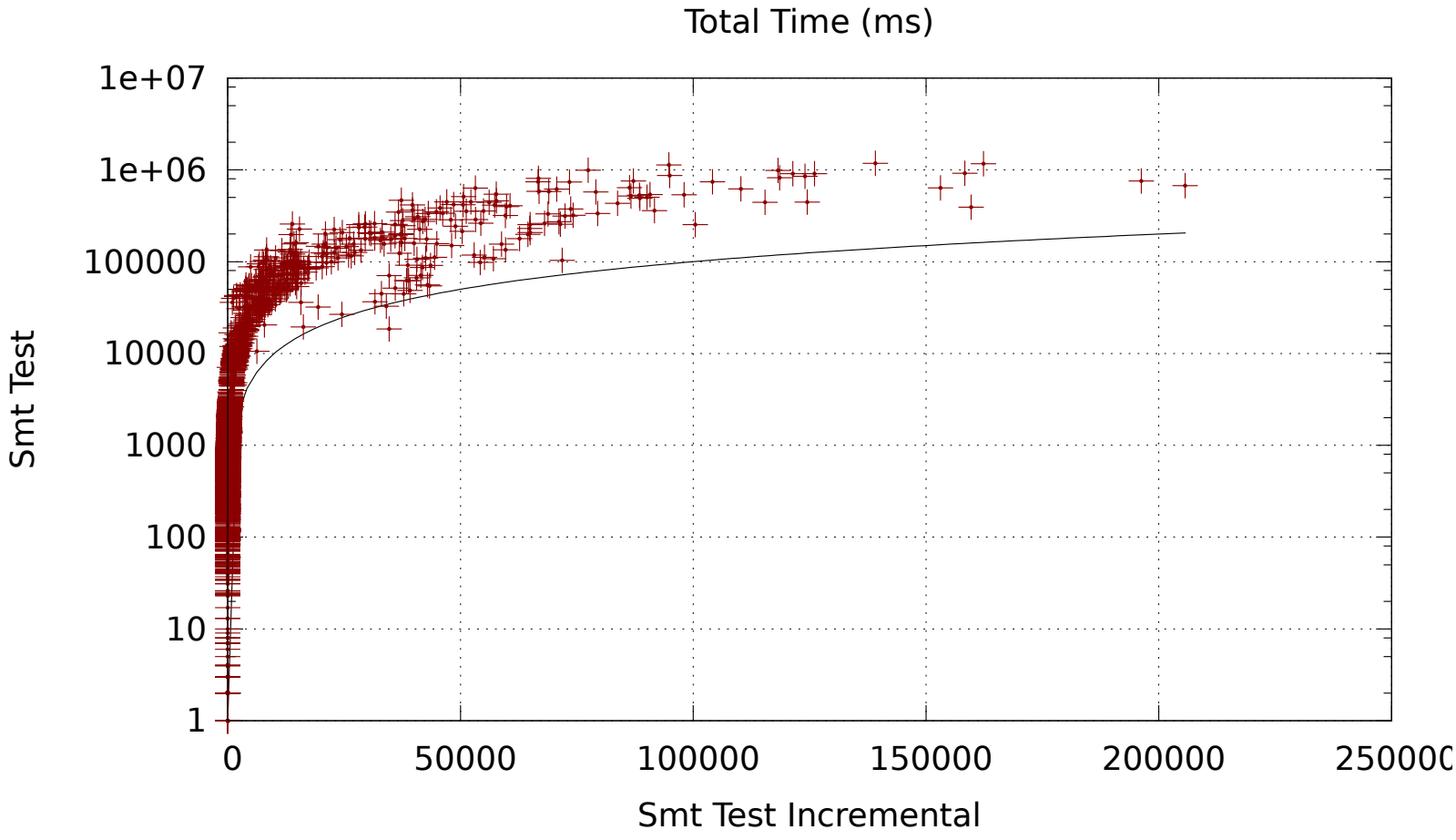
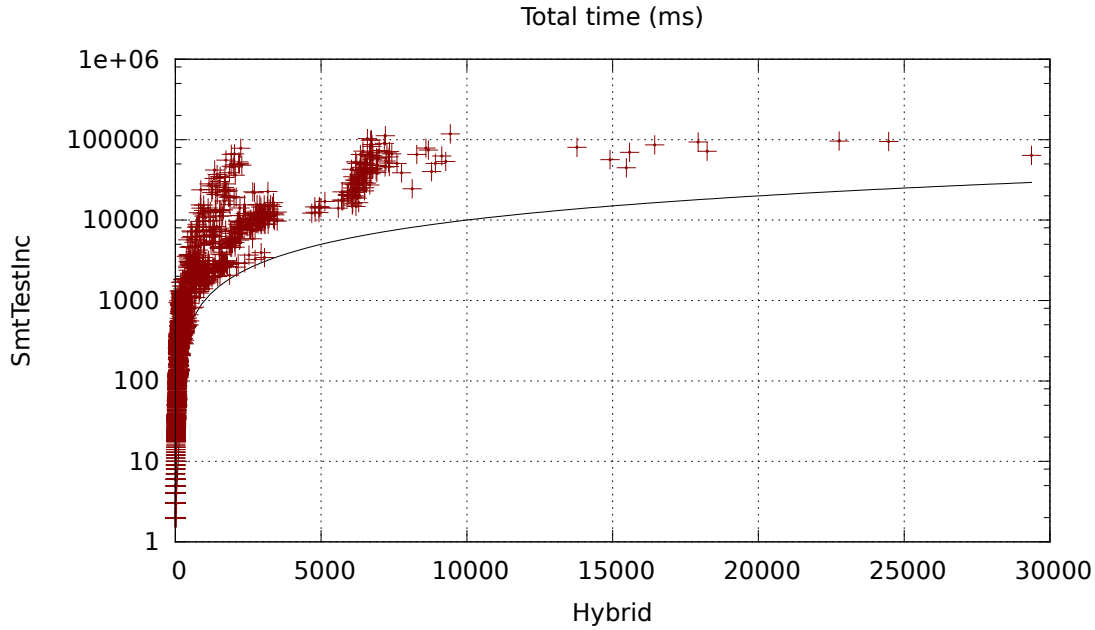


Figure: Incremental SMT-test / SMT-test.



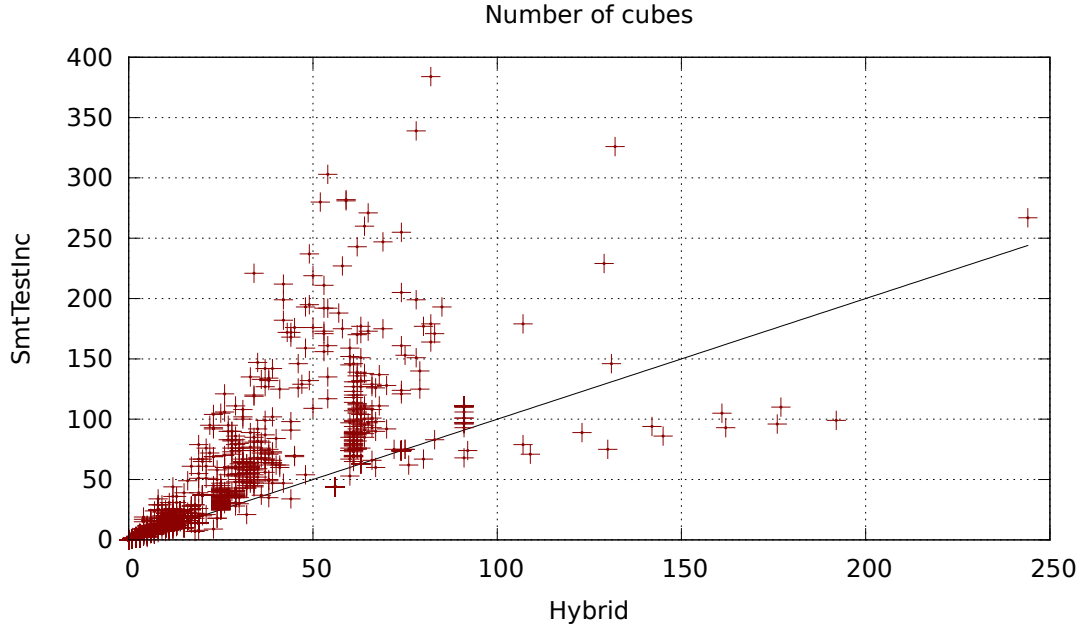


Figure: Hybrid / Incremental SMT-test

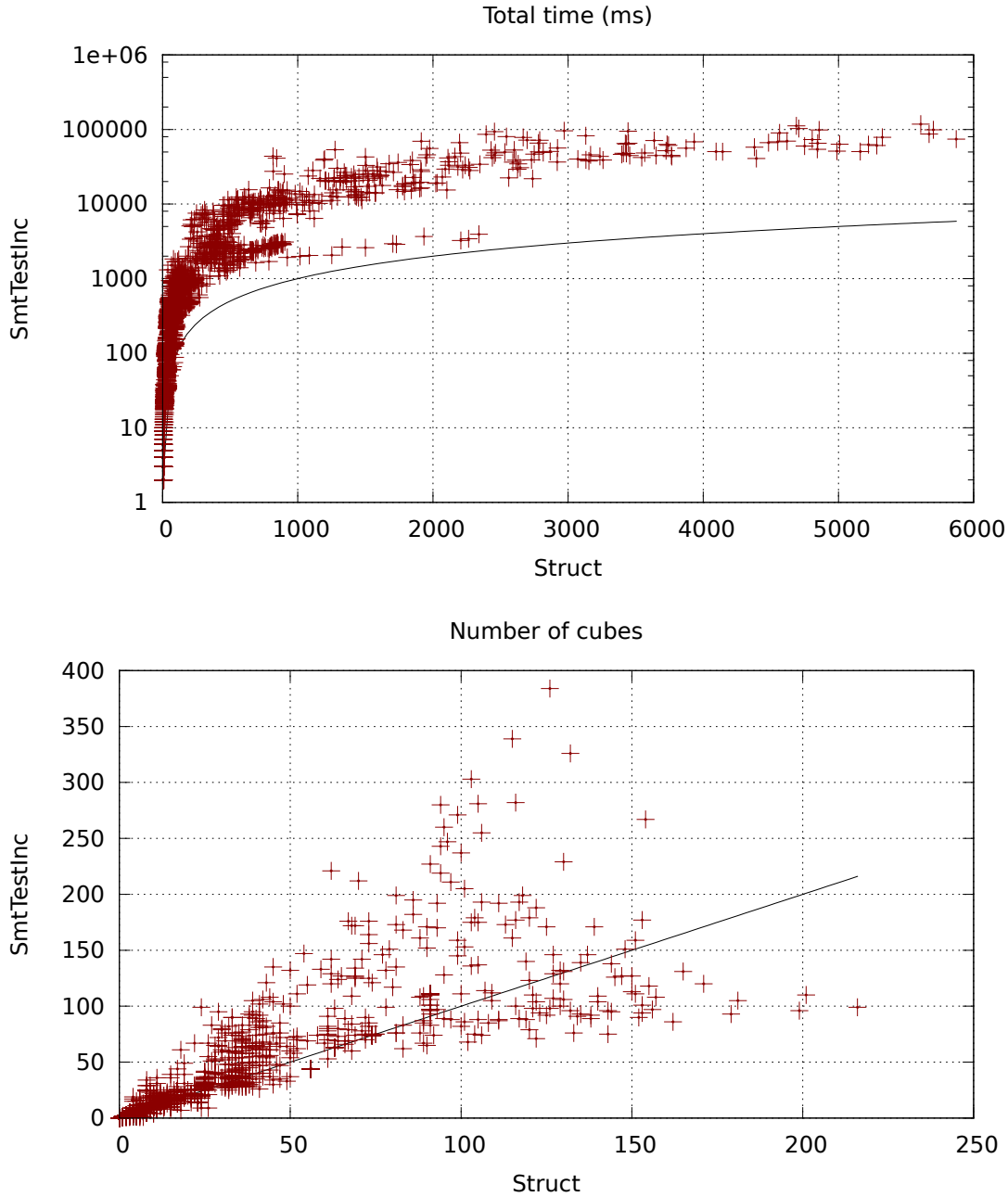


Figure: Structural analysis / Incremental SMT-test

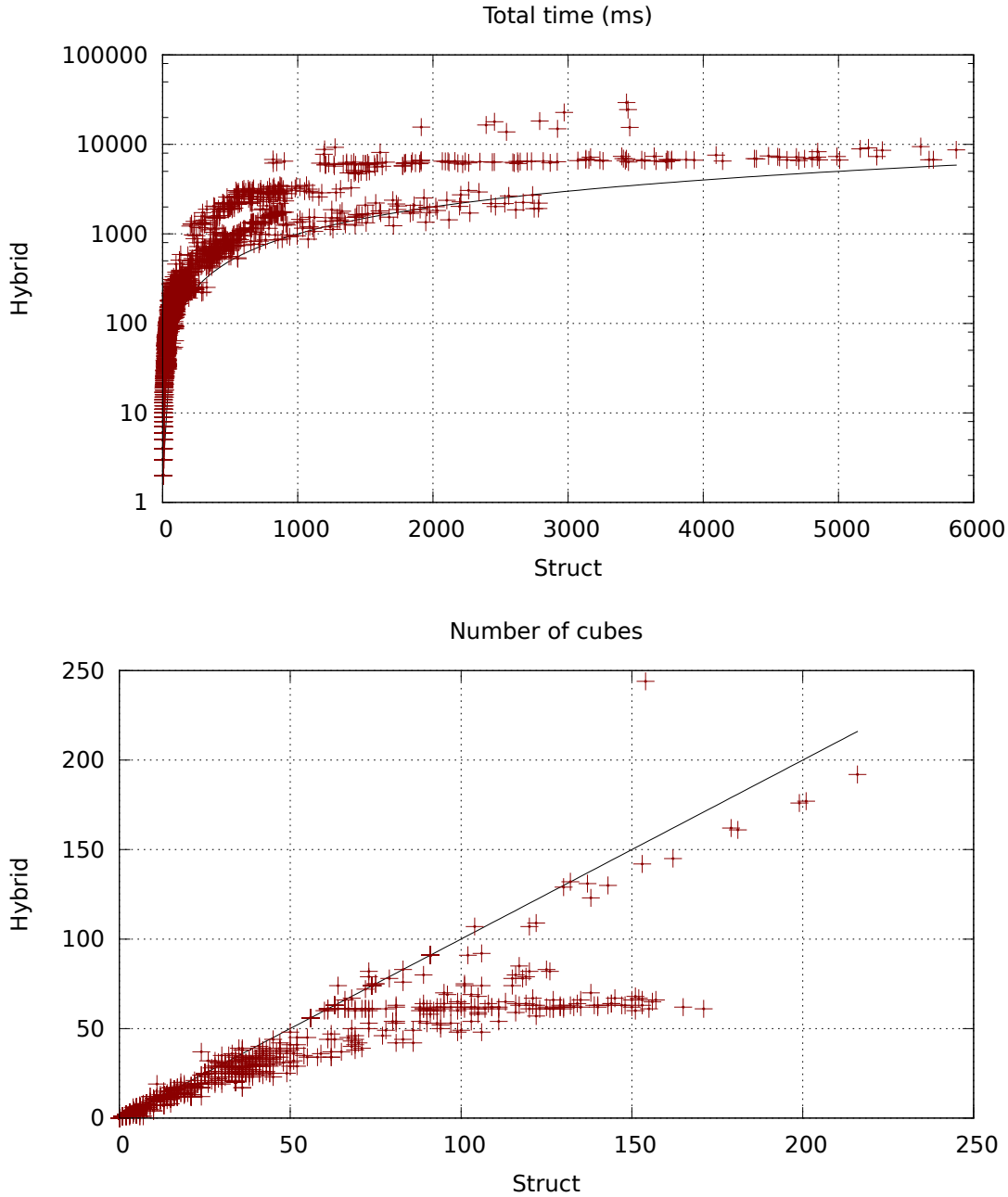


Figure: Structural analysis / Hybrid

Temps de calcul

- L'incrémentalité du solveur fait gagner jusqu'a un ordre de grandeur
- L'analyse structurelle est très loin devant (plusieurs ordres de grandeur)
- La méthode hybride est un peu plus lente
- La méthode SMT-test est la plus lente

Taille des formules DNFs

- La méthode hybride produit les DNFs les plus compactes
- La méthode structurelle produit les DNFs les moins compactes

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Sur la QE et l'extrapolation

Papiers QE: [1, 10, 3, 12] [14, 15, 5]
Une vague référence à une méthode d'extrapolation structurelle dan [8], mais aucun détail n'est donné.

Sur les applications

Les références d'utilisation de la QE pour la vérification abondent:

- Synthèse d'invariants numériques: [11, 9]
- Synthèse de contrôleurs: [13]
- Vérification par calcul de prémiage: [6]

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Perspectives

- Travail sur la QE non-linéaire en suivant l'approche structurelle + SMT
  - Solveur SMT et δ-satisfiabilité pour enumération de modèles [7]
  - Extrapolation structurelle (TODO)
  - Cylindrical Algebraic Decomposition pour projections des extrapolants [5]
- Synthèse d'invariants non-linéaires pour systèmes non-linéaires:
  - lois de commandes + modèle avion

Outline

1. Introduction: motivation des travaux
2. Quelques définitions
3. Extrapolation: Optimisations
   - Extrapolation par calcul de cœurs unsat
   - Extrapolation par analyse structurelle
   - Combinaison: analyse structurelle + cœurs unsat
4. Benchmarks des algorithmes proposés
5. Travaux connexes
6. Perspectives
7. Bibliographie

Phan Anh-Dung, Bjørner Nikolaj, and Monniaux David.
Anatomy of Alternating Quantifier Satisfiability (Work in progress).
In *SMT Workshop 2012 10th International Workshop on Satisfiability Modulo Theories*, page 6, Manchester, Royaume-Uni, June 2012.

R. Bagnara, P. M. Hill, and E. Zaffanella.
The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21, 2008.

N. Bjørner.
Linear quantifier elimination as an abstract decision procedure.
In J"urgen Giesl and Reiner H"ahnle, editors, *Automated Reasoning*, volume 6173 of *Lecture Notes in Computer Science*, pages 316–330. Springer Berlin / Heidelberg, 2010.

Thomas Bochot, Pierre Virelizier, Hélène Waeselynck, and Virginie Wiels.
Paths to property violation: A structural approach for analyzing counter-examples.
In *HASE*, pages 74–83. IEEE Computer Society, 2010.

George E. Collins.
Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition.
In *Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages*, pages 134–183, London, UK, UK, 1975. Springer-Verlag.

Werner Damm, Stefan Disch, Hardi Hungar, Jun Pang, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz.
Automatic verification of hybrid systems with large discrete state space.

In Susanne Graf and Wenhui Zhang, editors, *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2006.

Sicun Gao, Soonho Kong, and Edmund M. Clarke.
dreal: An smt solver for nonlinear theories over the reals.
In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 208–214. Springer, 2013.

Ajith K. John and Supratik Chakraborty.
A quantifier elimination algorithm for linear modular equations and disequations.
In *CAV*, pages 486–503, 2011.

Deepak Kapur.
Program analysis using quantifier-elimination heuristics - (extended abstract).
In Manindra Agrawal, S. Barry Cooper, and Angsheng Li, editors, *TAMC*, volume 7287 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2012.

David Monniaux.
A quantifier elimination algorithm for linear real arithmetic.
In *LPAR (Logic for Programming Artificial Intelligence and Reasoning)*, number 5330 in Lecture Notes in Computer Science, pages 243–257. Springer Verlag, 2008.

David Monniaux.
Automatic modular abstractions for template numerical constraints.
Logical Methods in Computer Science, 6(3), 2010.

David Monniaux.
Quantifier elimination by lazy model enumeration.
In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 585–599. Springer, 2010.

Thomas Sturm and Ashish Tiwari.
Verification and synthesis using real quantifier elimination.
In Éric Schost and Ioannis Z. Emiris, editors, *ISSAC*, pages 329–336. ACM, 2011.

Volker Weispfenning.
Quantifier elimination for real algebra – the cubic case.
In *Proceedings of the international symposium on Symbolic and algebraic computation*, ISSAC '94, pages 258–263, New York, NY, USA, 1994. ACM.

Volker Weispfenning.
Quantifier elimination for real algebra – the quadratic case and beyond.
Appl. Algebra Eng. Commun. Comput., 8(2):85–101, 1997.