Localisation d'erreurs

Une approche bornée à base de contraintes pour l'aide à la localisation d'erreurs

Bekkouche Mohammed, Collavizza Hélène, Rueher Michel

Univ. Nice Sophia Antipolis, CNRS, I3S, UMR 7271
06900 Sophia Antipolis, France
JFPC 2014

June 11, 2014

Plan

Introduction

Motivation

- Les outils de BMC(Bounded Model Checking) et de test génèrent un ou plusieurs contre-exemples
- Un contre-exemple fournit une trace d'exécution
- **La trace du contre-exemple** est souvent longue et compliquée à comprendre
- L'identification des portions erronées du code est complexe pour le programmeur

→ Nécessité de mettre en place des outils de localisation pour **assister le développeur** dans cette tâche

Introduction
Motivation

- Les outils de BMC(Bounded Model Checking) et de test génèrent un ou plusieurs contre-exemples
- Un contre-exemple fournit une trace d'exécution
- **La trace du contre-exemple** est souvent longue et compliquée à comprendre
- L'identification des portions erronées du code est complexe pour le programmeur

→ Nécessité de mettre en place des outils de localisation pour **assister le développeur** dans cette tâche

Introduction

Le problème

Entrées

- Un programme non conforme vis-à-vis de sa spécification : la postcondition POST violée
- Un contre-exemple CE fourni par un outil BMC

Sorties

Un ensemble réduit d'instructions suspectes permettant au programmeur de comprendre l'**origine de ses erreurs**

Introduction

Le problème

Entrées

- Un programme non conforme vis-à-vis de sa spécification : la postcondition POST violée
- Un contre-exemple CE fourni par un outil BMC

Sorties

Un ensemble réduit d'instructions suspectes permettant au programmeur de comprendre l'**origine de ses erreurs**

Introduction

Les idées

1. Le programme est modélisé en un CFG en forme DSA
2. Le programme et sa spécification sont traduits en contraintes numériques
3. CE un contre-exemple, PATH **un chemin erroné**
4. Le CSP $C = CE \cup PATH \cup POST$ **est inconsistant**

Points clés

- Quelles sont les instructions erronées dans $PATH$ qui rendent C **inconsistant** ?
- Quels sous-ensembles retirer pour que C devienne **faisable** ?
- Quels **chemins** explorer ? $\rightarrow$ **chemin** du CE, **déviations** à partir du CE

Introduction

Les idées

1. Le programme est modélisé en un CFG en forme DSA
2. Le programme et sa spécification sont traduits en contraintes numériques
3. *CE* un contre-exemple, PATH **un chemin erroné**
4. Le CSP $C = CE \cup PATH \cup POST$ **est inconsistant**

Points clés

- Quelles sont les instructions erronées dans *PATH* qui rendent *C* **inconsistant** ?
- Quels sous-ensembles retirer pour que *C* devienne **faisable** ?
- Quels **chemins** explorer ? $\rightarrow$ **chemin** du CE, **déviations** à partir du CE

Introduction

Les idées

1. Le programme est modélisé en un CFG en forme DSA
2. Le programme et sa spécification sont traduits en contraintes numériques
3. *CE* un contre-exemple, PATH **un chemin erroné**
4. Le CSP $C = CE \cup PATH \cup POST$ **est inconsistant**

Points clés

- Quelles sont les instructions erronées dans *PATH* qui rendent *C* **inconsistant** ?
- Quels sous-ensembles retirer pour que *C* devienne **faisable** ?
- Quels **chemins** explorer ? $\rightarrow$ **chemin** du CE, **déviations** à partir du CE

Introduction

Les idées

1. Le programme est modélisé en un CFG en forme DSA
2. Le programme et sa spécification sont traduits en contraintes numériques
3. *CE* un contre-exemple, PATH **un chemin erroné**
4. Le CSP $C = CE \cup PATH \cup POST$ **est inconsistant**

Points clés

- Quelles sont les instructions erronées dans *PATH* qui rendent C **inconsistant** ?
- Quels sous-ensembles retirer pour que C devienne **faisable** ?
- Quels **chemins** explorer ? $\rightarrow$ **chemin** du CE, **déviations** à partir du CE

Approche

Les **MCS**: Minimal Correction Subset

MCS: Définition

Soit C un ensemble de **contraintes infaisable**

$$M \subseteq C \text{ est un } \mathbf{MCS} \Leftrightarrow \begin{cases} M \subseteq C \\ Sol(< X, C \setminus M, D >) \neq \emptyset \\ \nexists C'' \subset M : Sol(< X, C \setminus C'', D >) = \emptyset \end{cases}$$

MCS: Exemple

- $C = \{c_1 : i = 0, c_2 : v = 5, c_3 : w = 6, c_4 : z = i + v + w, c_5 : ((z = 0 \vee i \neq 0) \wedge (v \geq 0) \wedge (w \geq 0))\}$ **est inconsistant**
- C a 4 **MCS**: $\{c_1\}$, $\{c_4\}$, $\{c_5\}$,$\{c_2, c_3\}$

Approche

Les **MCS**: Minimal Correction Subset

MCS: Définition

Soit C un ensemble de **contraintes infaisable**

$$M \subseteq C \text{ est un } \mathbf{MCS} \Leftrightarrow \begin{cases} M \subseteq C \\ Sol(< X, C \backslash M, D >) \neq \emptyset \\ \nexists C'' \subset M : Sol(< X, C \backslash C'', D >) = \emptyset \end{cases}$$

MCS: Exemple

- $C = \{c_1 : i = 0, c_2 : v = 5, c_3 : w = 6, c_4 : z = i + v + w, c_5 : ((z = 0 \vee i \neq 0) \wedge (v \geq 0) \wedge (w \geq 0))\}$ **est inconsistant**
- C a 4 **MCS**: $\{c_1\}$, $\{c_4\}$, $\{c_5\}$,$\{c_2, c_3\}$

Approche

Les **MCS**: Minimal Correction Subset

MCS: Définition

Soit C un ensemble de **contraintes infaisable**

$$M \subseteq C \text{ est un } \mathbf{MCS} \Leftrightarrow \begin{cases} M \subseteq C \\ Sol(< X, C \backslash M, D >) \neq \emptyset \\ \nexists C'' \subset M : Sol(< X, C \backslash C'', D >) = \emptyset \end{cases}$$

MCS: Exemple

- $C = \{c_1 : i = 0, c_2 : v = 5, c_3 : w = 6, c_4 : z = i + v + w, c_5 : ((z = 0 \vee i \neq 0) \wedge (v \geq 0) \wedge (w \geq 0))\}$ **est inconsistant**
- C a 4 **MCS**: $\{c_1\}$, $\{c_4\}$, $\{c_5\}$,$\{c_2, c_3\}$

Approche

Les **MCS**: Minimal Correction Subset

MCS: Définition

Soit C un ensemble de **contraintes infaisable**

$$M \subseteq C \text{ est un } \textbf{MCS} \Leftrightarrow \begin{cases} M \subseteq C \\ Sol(< X, C \backslash M, D >) \neq \emptyset \\ \nexists C'' \subset M : Sol(< X, C \backslash C'', D >) = \emptyset \end{cases}$$

MCS: Exemple

- $C = \{c_1 : i = 0, c_2 : v = 5, c_3 : w = 6, c_4 : z = i + v + w, c_5 : ((z = 0 \vee i \neq 0) \wedge (v \geq 0) \wedge (w \geq 0))\}$ **est inconsistant**
- C a 4 **MCS**: $\{c_1\}$, $\{c_4\}$, $\{c_5\}$,$\{c_2, c_3\}$

Approche

Algorithme (LocFaults)

- Calcul des **MCS** sur le **chemin** du CE
- Exploration DFS du CFG en propageant CE et **en déviant** au plus k instructions conditionnelles $c_1, .., c_k$
  - P: contraintes de propagation issues du CE (de la forme $variable = constante$)
  - C: contraintes du **chemin** jusqu'à c_k
  - Si $\underline{P \text{ satisfait } POST}$:
    - * $\{\neg c_1, .., \neg c_k\}$ est une **correction**,
    - * Les **MCS** de $C \cup \{\neg c_1, .., \neg c_k\}$ sont des **corrections**
- Borne pour les **MCS** calculés et les conditions **déviées**

Approche

Algorithme (LocFaults)

- Calcul des **MCS** sur le **chemin** du CE
- Exploration DFS du CFG en propageant CE et **en déviant** au plus k instructions conditionnelles $c_1, .., c_k$
  - P: contraintes de propagation issues du CE (de la forme $variable = constante$)
  - C: contraintes du **chemin** jusqu'à c_k
  - Si $\underline{P \text{ satisfait } POST}$:
    - * $\{\neg c_1, .., \neg c_k\}$ est une **correction**,
    - * Les **MCS** de $C \cup \{\neg c_1, .., \neg c_k\}$ sont des **corrections**
- Borne pour les **MCS** calculés et les conditions **déviées**

Exemple

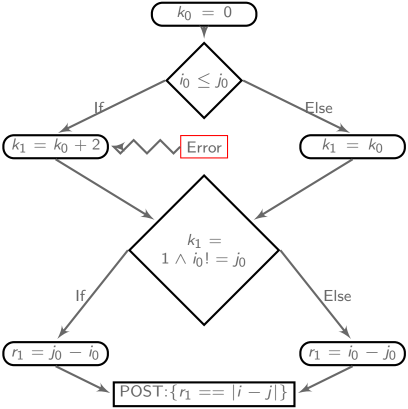
Calcul de la valeur absolue de i-j

```
class AbsMinus {
/*returns|i-j|,the absolute value of i minus j*/
/*@ ensures
  @ (result==|i-j|);
  @*/
  int AbsMinus (int i, int j) {
    int result;
    int k = 0;
    if (i <= j) {
        k = k+2;    // error : k = k+2 instead of
             k=k+1
    }
    if (k == 1 && i != j) {
        result = j-i;
    }
    else {
        result = i-j;
    }
    return result;
  }
}
```

Exemple

Calcul de la valeur absolue de i-j

```
class AbsMinus {
/*returns|i-j|,the absolute value of i minus j*/
/*@ ensures
  @ (result==|i-j|);
  @*/
  int AbsMinus (int i, int j) {
    int result;
    int k = 0;
    if (i <= j) {
       k = k+2;    // error : k = k+2 instead of k=k+1
    }
    if (k == 1 && i != j) {
        result = j-i;
    }
    else {
        result = i-j;
    }
    return result;
  }
}
```



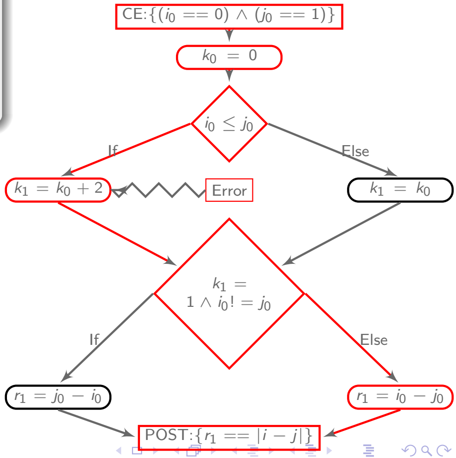
Exemple

Le **chemin** du contre-exemple

POST:$\{r_1 == |i - j|\}$

$\{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = k_0 + 2, r_1 = i_0 - j_0, r_1 = |i - j|\}$ **est inconsistant**

Unique MCS dans le chemin: $\{r_1 = i_0 - j_0\}$



Exemple

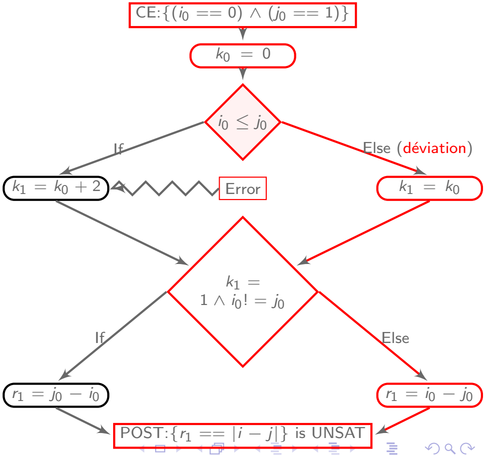
Le **chemin** obtenu **en déviant** la condition $i_0 \leq j_0$

Condition **déviée**: $\{i_0 \leq j_0\}$

$P = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = 0, r_1 = -1\}$

$P \cup \{r_1 = |i - j|\}$ **est inconsistant**

La déviation $\{i_0 \leq j_0\}$ **ne corrige pas le programme**



Exemple

Le **chemin en déviant** la condition $k_1 = 1 \wedge i_0 ! = j_0$

Condition **déviée**: $\{(k_1 = 1 \wedge i_0 ! = j_0)\}$

$P = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = 2, r_1 = 1\}$

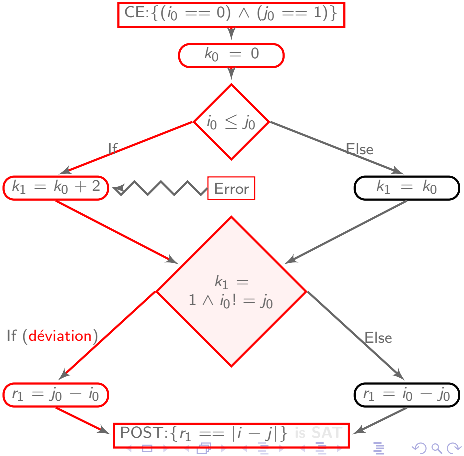
$P \cup \{r_1 = |i - j|\}$ **est consistant**

La déviation $\{(k_1 = 1 \wedge i_0 ! = j_0)\}$ **corrige** le programme

$C = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = k_0 + 2, \neg(k_1 = 1 \wedge i_0 ! = j_0)\}$

C **est inconsistant**

MCS dans le chemin: $\{k_0 = 0\}, \{k_1 = k_0 + 2\}$



Exemple

Le **chemin en déviant** la condition $k_1 = 1 \wedge i_0! = j_0$

Condition **déviée**: $\{(k_1 = 1 \wedge i_0! = j_0)\}$

$P = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = 2, r_1 = 1\}$

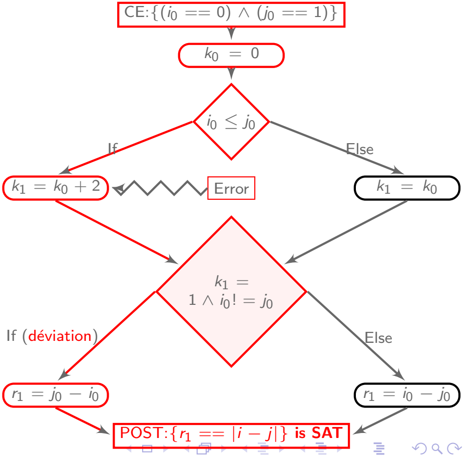
$P \cup \{r_1 = |i - j|\}$ **est consistant**

La déviation $\{(k_1 = 1 \wedge i_0! = j_0)\}$ **corrige** le programme

$C = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = k_0 + 2, \neg(k_1 = 1 \wedge i_0! = j_0)\}$

C **est inconsistant**

MCS dans le chemin: $\{k_0 = 0\}$, $\{k_1 = k_0 + 2\}$



État de l'art

Approches basées sur le test systématique

Tarantula, Delta Debugging

- Classement des instructions suspectes détectées lors de l'exécution d'une batterie de tests

+ Approches simples

− Besoin de **beaucoup** de cas de tests

Des approches qui requièrent l'existence d'un oracle

→ Décider si le résultat de **dizaines de milliers** de test est **juste**

Notre cadre est moins exigeant

→ Bounded Model Checking

État de l'art

Approches basées sur le test systématique

Tarantula, Delta Debugging

- Classement des instructions suspectes détectées lors de l'exécution d'une batterie de tests

\+ Approches simples

– Besoin de **beaucoup** de cas de tests

Des approches qui requièrent l'existence d'un oracle

→ Décider si le résultat de **dizaines de milliers** de test est **juste**

Notre cadre est moins exigeant

→ Bounded Model Checking

État de l'art

Approches basées sur le test systématique

Tarantula, Delta Debugging

- Classement des instructions suspectes détectées lors de l'exécution d'une batterie de tests

+ Approches simples

– Besoin de **beaucoup** de cas de tests
Des approches qui requièrent l'existence d'un oracle
→ Décider si le résultat de **dizaines de milliers** de test est **juste**

Notre cadre est moins exigeant
→ Bounded Model Checking

État de l'art

Méthodes basées sur la traduction booléenne du programme

BugAssist

- Une méthode de BMC, comme la notre
- **Différences majeures:**

→ Elle transforme la totalité du programme en une formule SAT

→ Elle est basée sur l'utilisation des solveurs MaxSAT

+ Approche globale

− Le complément de MaxSAT ne correspond pas nécessairement à des instructions sur un même **chemin**

→ Affichage de l'union de ces compléments

État de l'art

Méthodes basées sur la traduction booléenne du programme

BugAssist

- Une méthode de BMC, comme la notre
- **Différences majeures:**

→ Elle transforme la totalité du programme en une formule SAT

→ Elle est basée sur l'utilisation des solveurs MaxSAT

\+ Approche globale

− Le complément de MaxSAT ne correspond pas nécessairement à des instructions sur un même **chemin**

→ Affichage de l'union de ces compléments

Évaluation expérimentale

Outils utilisés

- **LocFaults**: notre implémentation
  - → Le solveur MIP de IBM ILOG CPLEX
  - → L'outil CPBPV pour générer le CFG et CE
  - → Benchmarks: les programmes Java
- **BugAssist**: l'outil de localisation d'erreurs pour l'approche `BugAssist`
  - → Le solveur MaxSAT MSUnCore2
  - → Benchmarks: les programmes ANSI-C

Évaluation expérimentale

Les programmes construits

- **Programmes simples**
  → AbsMinus, Minmax, Mid (programmes illustratifs)
  → Maxmin6var (programme sans points de jonction)
  → Tritype, TriPerimetre (programmes avec points de jonction)
  → Plusieurs versions erronées pour chaque programme
  Exemple: TriPerimetre → TriPerimetreKO, TriPerimetreKO2, TriPerimetreKO3
- **Un benchmark réaliste TCAS** (Traffic Collision Avoidance System)
  → 1608 cas de tests , sauf les cas de débordement du tableau *PositiveRAAltThresh*
  → TcasKO ... TcasKO41

Évaluation expérimentale

Les programmes construits

- **Programmes simples**
  → AbsMinus, Minmax, Mid (programmes illustratifs)
  → Maxmin6var (programme sans points de jonction)
  → Tritype, TriPerimetre (programmes avec points de jonction)
  → Plusieurs versions erronées pour chaque programme
  Exemple: TriPerimetre → TriPerimetreKO, TriPerimetreKO2, TriPerimetreKO3
- **Un benchmark réaliste TCAS** (Traffic Collision Avoidance System)
  → 1608 cas de tests , sauf les cas de débordement du tableau *PositiveRAAltThresh*
  → TcasKO ... TcasKO41

Évaluation expérimentale

Résultats (**MCS** identifiés)

Programme	Contre-exemple	LocFaults = 0	LocFaults ≤ 1	LocFaults ≤ 2	LocFaults ≤ 3	BugAssist
AbsMinusKO3	$\{i = 0, j = 1\}$	{20}	{16},{14},{12} {20}	{16},{14},{12} {20}	{16},{14},{12} {20}	{16, 20}
MinmaxKO	$\{in_1 = 2, in_2 = 1, in_3 = 3\}$	{10},{19}	{18},{10} {10},{19}	{18},{10} {10},{19}	{18},{10} {10},{19}	{18, 19, 22}
MidKO	$\{a = 2, b = 1, c = 3\}$	{19}	{19}	{19}	{14, 23, 26} {19}	{14, 19, 30}
Maxmin6varKO4	$\{a = 1, b = -3, c = -4, d = -2, e = -1, f = -2\}$	{116}	{116}	{116}	{12, 15, 19} {116}	{12, 166}
TritypeKO2	$\{i = 2, j = 2, k = 4\}$	{54}	{21} {26} {35},{27},{25} {53},{25},{27} {54}	{21} {26} {29, 57},{30},{27},{25} {32, 44},{33},{25},{27} {35},{27},{25} {53},{25},{27} {54}	{21} {26} {29, 57},{30},{27},{25} {32, 44},{33},{25},{27} {35},{27},{25} {53},{25},{27} {54}	{21, 26, 27, 29, 30, 32, 33, 35, 36, 53, 68}
TritypeKO4	$\{i = 2, j = 3, k = 3\}$	{46}	{45},{33},{25} {46}	{26, 32} {29, 32} {45},{33},{25} {46}	{26, 32} {29, 32} {32, 35, 49},{25} {32, 35, 53},{25} {32, 35, 57},{25} {45},{33},{25} {46}	{26, 27, 29, 30, 32, 33, 35, 45, 49, 68}
TriPerimetreKO	$\{i = 2, j = 1, k = 2\}$	{58}	{31} {37},{32},{27} {58}	{31} {37},{32},{27} {58}	{31} {37},{32},{27} {58}	{28, 29, 31, 32, 35, 37, 65, 72}

LocFaults fournit une localisation plus informative et plus explicative

Évaluation expérimentale

Résultats (Temps de calcul)

Programme	LocFaults					BugAssist	
	P	L				P	L
		$=0$	≤ 1	≤ 2	≤ 3		
AbsMinusKO3	$0,479s$	$0,076s$	$0,113s$	$0,357s$	$0,336s$	$0,02s$	$0,04s$
MinmaxKO	$0,528s$	$0,243s$	$0,318s$	$0,965s$	$1,016s$	$0,01s$	$0,09s$
MidKO	$0,524s$	$0,065s$	$0,078s$	$0,052s$	$0,329s$	$0,02s$	$0,08s$
Maxmin6varKO4	$0,538s$	$0,06s$	$0,07s$	$0,075s$	$0,56s$	$0,04s$	$0,78s$
TritypeKO2	$0,51s$	$0,023s$	$0,25s$	$2,083s$	$3,864s$	$0,02s$	$0,69s$
TritypeKO4	$0,497s$	$0,023s$	$0,095s$	$0,295s$	$5,127s$	$0,02s$	$0,21s$
TriPerimetreKO	$0,518s$	$0,047s$	$0,126s$	$1,096s$	$2,389s$	$0,03s$	$0,64s$

Les temps de LocFaults sont proches des temps de BugAssist

Évaluation expérimentale

Résultats (Nombre d'erreurs localisées pour TCAS)

Programme	Nb_E	Nb_CE	LF ≤ 1	LF ≤ 2	BA
TcasKO	1	131	131	131	131
TcasKO2	2	67	67	134	67
TcasKO3	1	23	2	2	23
TcasKO4	1	20	16	20	20
TcasKO5	1	10	10	10	10
TcasKO6	3	12	36	36	24
TcasKO7	1	36	23	36	0
TcasKO8	1	1	1	1	0
TcasKO9	1	7	7	7	7
TcasKO10	6	14	16	65	84
TcasKO11	6	14	16	34	46
TcasKO12	1	70	52	52	70
TcasKO13	1	4	3	3	4
TcasKO14	1	50	6	6	51
TcasKO16	1	70	22	70	0
TcasKO17	1	35	22	35	0
TcasKO18	1	29	21	28	0
TcasKO19	1	19	13	19	0
TcasKO20	1	18	18	18	18
TcasKO21	1	16	16	16	16
TcasKO22	1	11	11	11	11
TcasKO23	1	41	41	41	41
TcasKO24	1	7	7	7	7
TcasKO25	1	3	0	3	3
TcasKO26	1	11	11	11	11
TcasKO27	1	10	10	10	10
TcasKO28	2	75	74	148	121
TcasKO29	2	18	17	35	0
TcasKO30	2	57	57	114	0
TcasKO34	1	77	77	77	77
TcasKO35	4	75	74	148	115
TcasKO36	1	122	120	120	0
TcasKO37	4	94	110	235	236
TcasKO39	1	3	0	3	3
TcasKO40	2	122	0	103	120
TcasKO41	1	20	17	20	20

LocFaults se compare très favorablement à BugAssist

Conclusion

- Approche de localisation d'erreurs bornée
  - Exploration **DFS** bornée
  - Calcul des **MCS** borné

    → Pour empêcher l'explosion combinatoire
- Basée sur l'utilisation des solveurs de contraintes
- `LocFaults` fournit des résultats plus précis et plus pertinents par rapport à `BugAssist`

  → Les temps des deux outils sont similaires
- `LocFaults` localise fréquemment les erreurs pour les programmes TCAS

Conclusion

- Approche de localisation d'erreurs bornée
  - Exploration **DFS** bornée
  - Calcul des **MCS** borné
    → Pour empêcher l'explosion combinatoire
- Basée sur l'utilisation des solveurs de contraintes
- `LocFaults` fournit des résultats plus précis et plus pertinents par rapport à `BugAssist`
  → Les temps des deux outils sont similaires
- `LocFaults` localise fréquemment les erreurs pour les programmes TCAS

Conclusion

- Approche de localisation d'erreurs bornée
  - Exploration **DFS** bornée
  - Calcul des **MCS** borné

    → Pour empêcher l'explosion combinatoire
- Basée sur l'utilisation des solveurs de contraintes
- `LocFaults` fournit des résultats plus précis et plus pertinents par rapport à `BugAssist`

  → Les temps des deux outils sont similaires
- `LocFaults` localise fréquemment les erreurs pour les programmes TCAS

Conclusion

- Approche de localisation d'erreurs bornée
  - Exploration **DFS** bornée
  - Calcul des **MCS** borné

    → Pour empêcher l'explosion combinatoire
- Basée sur l'utilisation des solveurs de contraintes
- `LocFaults` fournit des résultats plus précis et plus pertinents par rapport à `BugAssist`

  → Les temps des deux outils sont similaires
- `LocFaults` localise fréquemment les erreurs pour les programmes TCAS

Perspectives

- Nouvelle version incrémentale de l'algorithme
- Traiter des programmes avec boucles
- Étendre notre implémentation pour supporter les instructions non-linéaires plus complexes par l'usage des **solveurs spécialisés**
- Expérimenter plus de programmes réels
- Traiter les instructions avec calcul sur les flottants

  → Utiliser des **solveurs spécialisés** sur les flottants

Perspectives

- Nouvelle version incrémentale de l'algorithme
- Traiter des programmes avec boucles
- Étendre notre implémentation pour supporter les instructions non-linéaires plus complexes par l'usage des **solveurs spécialisés**
- Expérimenter plus de programmes réels
- Traiter les instructions avec calcul sur les flottants

  $\rightarrow$ Utiliser des **solveurs spécialisés** sur les flottants

Perspectives

- Nouvelle version incrémentale de l'algorithme
- Traiter des programmes avec boucles
- Étendre notre implémentation pour supporter les instructions non-linéaires plus complexes par l'usage des **solveurs spécialisés**
- Expérimenter plus de programmes réels
- Traiter les instructions avec calcul sur les flottants

  → Utiliser des **solveurs spécialisés** sur les flottants

Perspectives

- Nouvelle version incrémentale de l'algorithme
- Traiter des programmes avec boucles
- Étendre notre implémentation pour supporter les instructions non-linéaires plus complexes par l'usage des **solveurs spécialisés**
- Expérimenter plus de programmes réels
- Traiter les instructions avec calcul sur les flottants

  → Utiliser des **solveurs spécialisés** sur les flottants

Perspectives

- Nouvelle version incrémentale de l'algorithme
- Traiter des programmes avec boucles
- Étendre notre implémentation pour supporter les instructions non-linéaires plus complexes par l'usage des **solveurs spécialisés**
- Expérimenter plus de programmes réels
- Traiter les instructions avec calcul sur les flottants

  $\rightarrow$ Utiliser des **solveurs spécialisés** sur les flottants

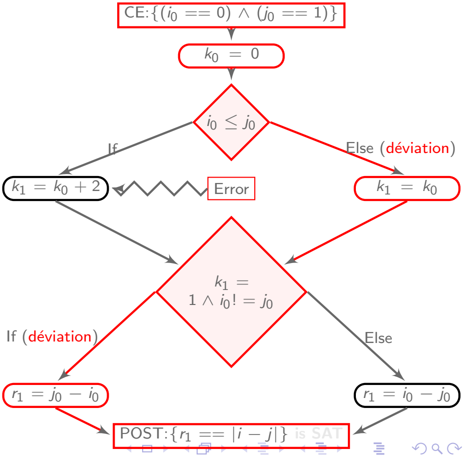
Merci pour votre attention.

Quelles sont vos questions ?

Exemple

Le chemin d'une déviation non minimale: $\{i_0 \leq j_0, k_1 = 1 \wedge i_0! = j_0\}$

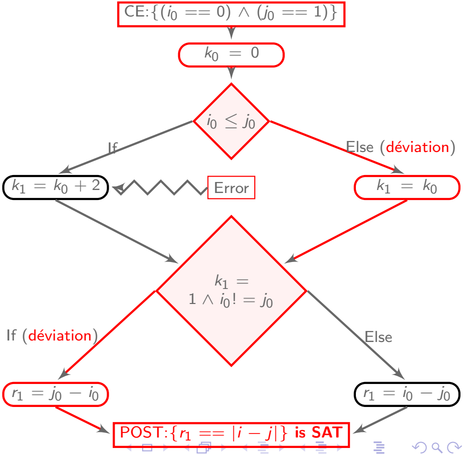
Conditions **déviées**:
$\{i_0 \leq j_0, (k_1 = 1 \wedge i_0! = j_0)\}$
$P = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = 0, r_1 = 1\}$
$P \cup \{r_1 = |i - j|\}$ **est consistant**
La déviation n'est pas minimale



Exemple

Le **chemin** d'**une déviation** non minimale: $\{i_0 \leq j_0, k_1 = 1 \wedge i_0 != j_0\}$

Conditions **déviées**:
$\{i_0 \leq j_0, (k_1 = 1 \wedge i_0 != j_0)\}$
$P = \{i_0 = 0, j_0 = 1, k_0 = 0, k_1 = 0, r_1 = 1\}$
$P \cup \{r_1 = |i - j|\}$ **est consistant**
La déviation n'est pas minimale



Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

Fonction MCS(C,MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
  $C' \leftarrow$ ADDYVARS(C); $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
  tant que SAT(C') $\wedge\ k \leq MCS_b$ **faire**
    $C'_k \leftarrow C' \wedge$ ATMOST($\{\neg y_1, \neg y_2, \ldots, \neg y_n\}$,$k$)
    tant que SAT(C'_k) **faire**
      $newMCS \leftarrow \emptyset$
      pour chaque indicateur y_i **faire**
        Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
        si $val(y_i) = 0$ **alors**
          $newMCS \leftarrow newMCS \cup \{c_i\}$.
        fin
      fin
      $MCS.add(newMCS)$.
      $C'_k \leftarrow C'_k \wedge$ BLOCKINGCLAUSE($newMCS$)
      $C' \leftarrow C' \wedge$ BLOCKINGCLAUSE($newMCS$)
    fin
    $k \leftarrow k + 1$
  fin
  retourner MCS
fin

Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

Fonction MCS(C,MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
  $C' \leftarrow \textsc{AddYVars}(C)$; $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
  tant que $\mathrm{SAT}(C') \wedge k \leq MCS_b$ **faire**
    $C'_k \leftarrow C' \wedge \textsc{AtMost}(\{\neg y_1, \neg y_2, \ldots, \neg y_n\}, k)$
    tant que $\mathrm{SAT}(C'_k)$ **faire**
      $newMCS \leftarrow \emptyset$
      pour chaque indicateur y_i **faire**
        Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
        si $val(y_i) = 0$ **alors**
          $newMCS \leftarrow newMCS \cup \{c_i\}$.
        fin
      fin
      $MCS.add(newMCS)$.
      $C'_k \leftarrow C'_k \wedge \textsc{BlockingClause}(newMCS)$
      $C' \leftarrow C' \wedge \textsc{BlockingClause}(newMCS)$
    fin
    $k \leftarrow k + 1$
  fin
  retourner MCS
fin

Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

Fonction MCS(C,MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
  $C' \leftarrow \textsc{AddYVars}(C)$; $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
  tant que $\mathrm{SAT}(C') \wedge k \leq MCS_b$ **faire**
    $C'_k \leftarrow C' \wedge \textsc{AtMost}(\{\neg y_1, \neg y_2, \ldots, \neg y_n\}, k)$
    tant que $\mathrm{SAT}(C'_k)$ **faire**
      $newMCS \leftarrow \emptyset$
      pour chaque indicateur y_i **faire**
        Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
        si $val(y_i) = 0$ **alors**
          $newMCS \leftarrow newMCS \cup \{c_i\}$.
        fin
      fin
      $MCS.add(newMCS)$.
      $C'_k \leftarrow C'_k \wedge \textsc{BlockingClause}(newMCS)$
      $C' \leftarrow C' \wedge \textsc{BlockingClause}(newMCS)$
    fin
    $k \leftarrow k + 1$
  fin
  retourner MCS
fin

Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

Fonction MCS(C,MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
    $C' \leftarrow \textsc{AddYVars}(C)$; $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
    tant que $\mathrm{SAT}(C') \wedge k \leq MCS_b$ **faire**
        $C'_k \leftarrow C' \wedge \textsc{AtMost}(\{\neg y_1, \neg y_2, \ldots, \neg y_n\}, k)$
        tant que $\mathrm{SAT}(C'_k)$ **faire**
            $newMCS \leftarrow \emptyset$
            pour chaque indicateur y_i **faire**
                Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
                si $val(y_i) = 0$ **alors**
                    $newMCS \leftarrow newMCS \cup \{c_i\}$.
                fin
            fin
            $MCS.add(newMCS)$.
            $C'_k \leftarrow C'_k \wedge \textsc{BlockingClause}(newMCS)$
            $C' \leftarrow C' \wedge \textsc{BlockingClause}(newMCS)$
        fin
        $k \leftarrow k + 1$
    fin
    retourner MCS
fin

Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

`Fonction MCS`(C, MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
  $C' \leftarrow \textsc{AddYVars}(C)$; $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
  tant que $\mathrm{SAT}(C') \wedge k \leq MCS_b$ **faire**
    $C'_k \leftarrow C' \wedge \textsc{AtMost}(\{\neg y_1, \neg y_2, \ldots, \neg y_n\}, k)$
    tant que $\mathrm{SAT}(C'_k)$ **faire**
      $newMCS \leftarrow \emptyset$
      pour chaque indicateur y_i **faire**
        Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
        si $val(y_i) = 0$ **alors**
          $newMCS \leftarrow newMCS \cup \{c_i\}$.
        fin
      fin
      $MCS.add(newMCS)$.
      $C'_k \leftarrow C'_k \wedge \textsc{BlockingClause}(newMCS)$
      $C' \leftarrow C' \wedge \textsc{BlockingClause}(newMCS)$
    fin
    $k \leftarrow k + 1$
  fin
  retourner MCS
fin

Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

Fonction MCS(C,MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
    $C' \leftarrow \text{AddYVars}(C)$; $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
    tant que $\text{SAT}(C') \wedge k \leq MCS_b$ **faire**
        $C'_k \leftarrow C' \wedge \text{AtMost}(\{\neg y_1, \neg y_2, \ldots, \neg y_n\}, k)$
        tant que $\text{SAT}(C'_k)$ **faire**
            $newMCS \leftarrow \emptyset$
            pour chaque indicateur y_i **faire**
                Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
                si $val(y_i) = 0$ **alors**
                    $newMCS \leftarrow newMCS \cup \{c_i\}$.
                fin
            fin
            $MCS.add(newMCS)$.
            $C'_k \leftarrow C'_k \wedge \text{BlockingClause}(newMCS)$
            $C' \leftarrow C' \wedge \text{BlockingClause}(newMCS)$
        fin
        $k \leftarrow k + 1$
    fin
    retourner MCS
fin

Approche

Algorithme de Liffiton et Sakallah pour calculer les MCS

Fonction MCS(C,MCS_b)
Entrées: C: Ensemble de contraintes infaisable, MCS_b: Entier
Sorties: MCS: Liste de MCS de C de cardinalité inférieure à MCS_b
début
    $C' \leftarrow \textsc{AddYVars}(C)$; $MCS \leftarrow \emptyset$; $k \leftarrow 1$;
    tant que $\underline{\mathrm{SAT}(C') \wedge k \leq MCS_b}$ **faire**
        $C'_k \leftarrow C' \wedge \textsc{AtMost}(\{\neg y_1, \neg y_2, \ldots, \neg y_n\}, k)$
        tant que $\underline{\mathrm{SAT}(C'_k)}$ **faire**
            $newMCS \leftarrow \emptyset$
            pour chaque indicateur y_i **faire**
                Soit y_i l'indicateur de la contrainte $c_i \in C$, et $val(y_i)$ la valeur de y_i dans la solution calculée de C'_k.
                si $\underline{val(y_i) = 0}$ **alors**
                    $newMCS \leftarrow newMCS \cup \{c_i\}$.
                fin
            fin
            $MCS.add(newMCS)$.
            $C'_k \leftarrow C'_k \wedge \textsc{BlockingClause}(newMCS)$
            $C' \leftarrow C' \wedge \textsc{BlockingClause}(newMCS)$
        fin
        $k \leftarrow k + 1$
    fin
    retourner MCS
fin