ED STIC, Université de Nice-Sophia Antipolis
Laboratoire **I3S**

Le travail en cours sur la localisation des erreurs avec les IIS

Spécialisations : *programmation par contraintes, vérification de programmes, localisation des erreurs.*
Mars 2013

Présenté par :
BEKKOUCHE MOHAMMED
En collaboration avec :
MICHEL RUEHER
HELENE COLLAVIZZA
YAHIA LEBBAH
OLIVER PONSINI

Résumé

- Localisation d'erreurs $\sqsubset$ Débogage logiciel $\sqsubset$ Génie logiciel
- Un contre-exemple -> une trace d'exécution

  L'espace de recherche est limité à l'ensemble d'instructions dans la trace du contre-exemple
- Le formalisme de la programmation par contraintes

  Pourquoi ?
  - Pour modéliser le problème,
  - Et pour le résoudre.

L'objectif du travail

- Localiser les fautes dans les programmes impératifs
- Pour lesquels nous avons un contre-exemple

Introduction, la problématique et les hypothèses

- Un programme peut contenir des erreurs
- Ces erreur peuvent nuire au bon fonctionnement du programme
- Le processus de logiciel de débogage est inévitable
  - La détection d'erreurs, **la localisation des fautes**, la correction des fautes
- Un programme avec des erreurs :
  - Un outil pour model-checking (e.g. CPBPV, CBMC) pour obtenir un contre-exemple
  - Le contre-exemple $\rightarrow$ Trace d'exécution
- Le problème :
  - La trace d'exécution du contre-exemple est souvent long et difficile à comprendre
  - La raison pour laquelle le problème de la localisation d'erreurs est difficile

Notre idée :

- Contre-exemple $+$ la trace du contre-exemple $+$ La postcondition $\rightarrow$ Un ensemble de contraintes infaisable $\rightarrow$ **Un ensemble de contraintes représentant un conflit minimal** (**IIS**)

Introduction, la problématique et les hypothèses

- Nous considérons un ensemble d'hypothèses :
  - Un programme avec une seule instruction d'affectation erronée
  - Un contre-exemple fourni par un outil de model-checking
- Dans ce contexte, nous étudions le cas où :
  - Le chemin est correct,
  - Le chemin est incorrect.

```
1  class program{
2
3   /*@ ensures
4     @ (c >= d+e);
5     @*/
6  void foo(int a,int b){
7    int c;
8    int d;
9    int e;
10   int f;
11   if (a>=0){
12     ...
13   }
14   else{
15     c=b; /* l'erreur */
16     d=1;
17     e=-a;
18     if (a>b){
19       f=b+e+a;
20       d=d+4;
21     }
22     else{
23       ...
24     }
25   }
26   c=c+d+e;
27 }
28 }
```

Le programme Foo

Exemple de motivation

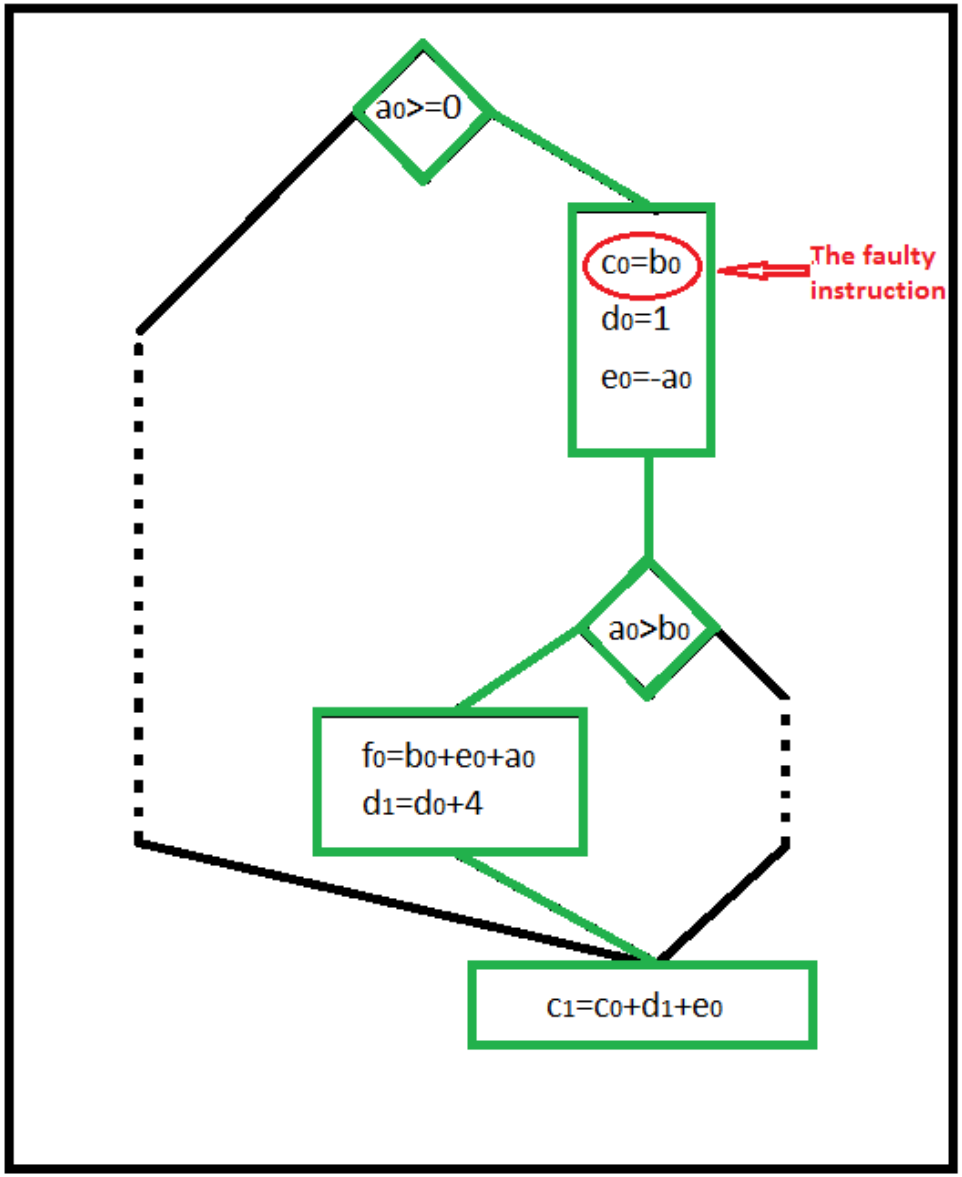

FIGURE: The control flow graph of the SSA form of the foo program

Exemple de motivation

Description de l'exemple :

- Le programmes est écrit en JAVA
- Il est annoté avec une spécification JML (Java Modeling Language)
- L'erreur injectée est sur une instruction d'affectation : $c = d$
- L'instruction erronée est en dépendance de données avec les variables de la postcondition
- Notre but :
  - Trouver l'ensemble minimal **d'instructions suspectes** dans le programme
  - Qui couvre l'instruction erronée insérée

Exemple de motivation

Notre approche pour localiser les fautes :

- Par l'utilisation d'un outil de BMC (Bounded Model Cheking), on obtient le contre-exemple suivant :
  $CE_{PROG}\ (a_0 = -1, b_0 = -2)$
- On génère l'ensemble de contraintes qui correspond à la trace du contre-exemple :
  $C_{TCE} = \{c_0 = b_0, d_0 = 1, e_0 = -a_0, a_0 > b_0, f_0 = b_0 + e_0 + a_0, d_1 = d_1 + 4, c_1 = c_0 + d_1 + e_0\}$
- On génère les contraintes qui correspondent à la postcondition :
  $C_{POST} = \{c_1 >= d_1 + e_0\}$
- On génère aussi les contraintes du contre-exemple :
  $C_{CE_{PROG}} = \{a_0 = -1, b_0 = -2\}$

Exemple de motivation

Notre approche pour localiser les fautes :

- L'identification des contraintes fautives :
  - $C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST}$ is infeasible
    Il comporte au moins un sous-système de contraintes irréductibles infaisable (**IIS**)
  - $C_{CE_{PROG}} \cup C_{LOC} \cup C_{POST}$ doit être infaisable et C_{LOC} minimal
    $C_{LOC} = \{c_0 = b_0, c_1 = c_0 + d_1 + e_0\}$
    - $\{a_0 = -1, b_0 = -2\} \cup \{c_0 = b_0, c_1 = c_0 + d_1 + e_0\} \cup \{c_1 >= d_1 + e_0\}$ est infaisable
    - $\{a_0 = -1, b_0 = -2\} \cup \{c_0 = b_0\} \cup \{c_1 >= d_1 + e_0\}$ est infaisable
      $\{a_0 = -1, b_0 = -2\} \cup \{c_1 = c_0 + d_1 + e_0\} \cup \{c_1 >= d_1 + e_0\}$ est infaisable
  - $C' = C_{CE_{PROG}} \cup C_{TCE} \backslash c_i \cup C_{POST}$ est infaisable
    $(c_i \in C_{LOC})$
    $\rightarrow$ Parce que le système infaisable d'entrée a un seul **IIS**
- $LOC = \{$***ligne 15***, *ligne* 26$\}$

Notations et définitions

- CSP

  $\mathcal{P} = < X, D, C >$

- La fonction *Sol*

  $\Delta = D_{x_1} \times D_{x_2} \times ... \times D_{x_n}$

  $Sol : C \times D \longrightarrow \Delta$

- IS
  - $IS \subseteq C$.
  - $Sol(IS, D) = \emptyset$.
- MIN-UNCSP
  - $Sol(C \backslash MUC, D) \neq \emptyset$.
  - $\nexists\ MUC' \subset MUC \mid Sol(C \backslash MUC', D) \neq \emptyset$.
- IIS
  - S est un IS.
  - $\forall\ S' \subset S . Sol(S', D) \neq \emptyset$.
- MIN-IIS
  - MS est un IIS.
  - $\forall\ S \in \Sigma_{IIS} . |MS| \leq |S|$

    (Σ_{IIS} représente l'ensemble contenant tous les IISs inclus dans C).

Notations et définitions

- IIS-COVER
  - $\forall\, S \in \Sigma_{IIS}, \exists\, c \in SC$ such that $c \in S$
    (Σ_{IIS} est l'ensemble de tous les IIS dans C).
- MIN-IIS-COVER
  - MSC est un IIS-COVER.
  - $\forall\, SC \in \Sigma_{SC}.|MSC| \leq |SC|$
    (Σ_{SC} est l'ensemble de tous les IISs dans C).
- MIN-UNCSP $\equiv$ MIN-IIS-COVER

Notations et définitions

Exemple Soit $\mathcal{P} = < X, D, C >$, avec $C = \{C_1, C_2, ..., C_{16}\}$.

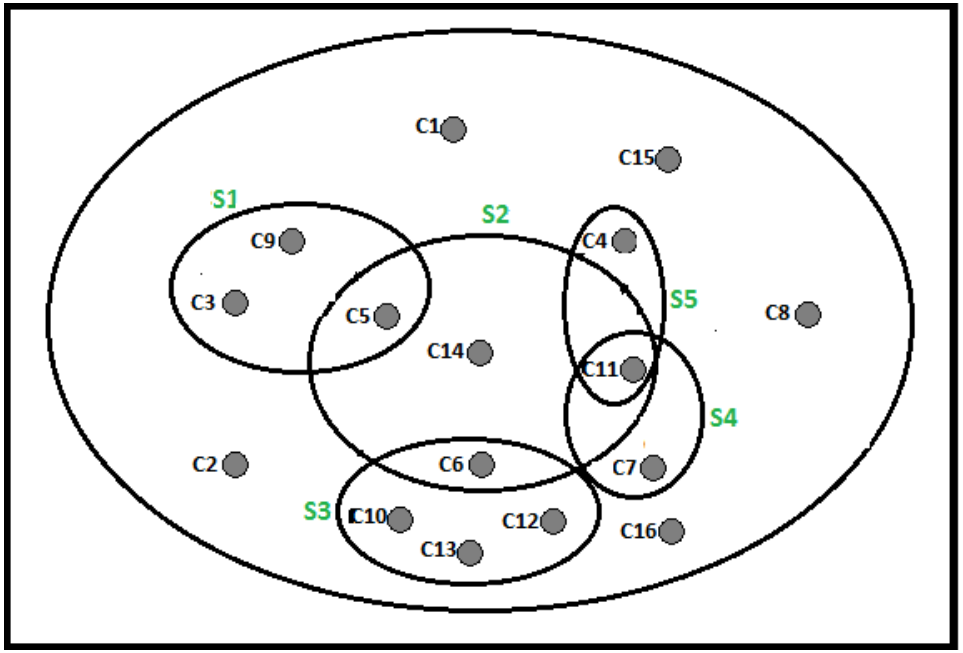

FIGURE: Un système de contraintes avec cinq IISs

$\Sigma_{IIS} = \{S_1, S_2, S_3, S_4, S_5\}$.
From the set Σ_{IIS}, on peut calculer :

- MIN-IIS : $\Sigma_{MS} = \{\{C_7, C_{11}\}, \{C_4, C_{11}\}\}$, $|MS| = 2$.
- L'ensemble qui contient tous les IIS-COVERs :
  $\Sigma_{SC} =$
  $\{C, \{S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5\}, ..., \{C_3, C_{11}, C_{13}\}, \{C_5, C_6, C_{11}\}, ...\}$.
- Le MIN-IIS-COVER (MIN-UNCSP) : Il y a exactement douze ($|S_1| \times |S_3|$) MIN-IIS-COVERs avec une cardinalité égale à trois.
  Exemple : $\{C_3, C_{11}, C_{13}\}$

Notations and définitions

- Deux classes de contraintes
  - C_{HARD}
  - C_{SOFT}
- Conflict Set
  - $CS \subseteq C_{SOFT}$
  - $CS \cup C_{HARD}$ est un IS ($Sol(CS \cup C_{HARD}, D) = \emptyset$)
- Minimal Conflict Set
  - CS est un Conflict Set
  - $\forall\ CS' \subset CS$, CS' n'est pas un Conflict Set

La définition du problème de localisation d'erreurs

- Un programme erroné : *PROG*
- Une postcondition violée : *POST*
- Un contre-exemple : CE_{PROG}
- $\rightarrow$ On peut trouver la trace du contre-exemple *TCE*

Le problème de la localisation d'erreurs dans *TCE*

Quel est l'ensemble minimal à supprimer (ou changer) dans TCE pour atteindre la satisfiabilité de $CE_{PROG} \wedge POST$ *?*

Le problème de la localisation d'erreurs dans *TCE*

Quel est l'ensemble minimal d'instructions (un ou plusieurs ensembles) en contradiction avec $CE_{PROG} \wedge POST$ *?*

La définition du problème de la localisation d'erreurs

- Le problème de localisation d'erreurs dans *TCE* $\longrightarrow$ Problème qui consiste à isoler l'infaisablilité dans P :
  - $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$

L'isolation de l'infaisablité dans P

Quel est l'ensemble minimal de contraintes à retirer de C_{TCE} pour atteindre la faisabilité de $C_{CE_{PROG}} \cup C_{POST}$?

L'isolation de l'infaisablité dans P

Quel est l'ensemble minimal conflictuel (un ou plusieurs ensembles) dans C_{TCE} en contradiction avec $C_{CE_{PROG}} \cup C_{POST}$?

Notre approche

Algorithm 1 Notre algorithme de localisation des erreurs

Input : *PROG* : Un programme ; *PRED* : Une précondition ; *POST* : Une postcondition
Output : *LOC* : Un ensemble d'instructions suspectes dans *PROG*

1: $CE_{PROG} \leftarrow BMC(PROG, PRED, POST)$
2: **if** CE_{PROG} **is** *Nulle* **then**
3: $\quad LOC \leftarrow Nulle$
4: $\quad$ WRITE("Le programme est conforme vis-à-vis de sa spécification")
5: **else**
6: $\quad TCE \leftarrow$ GENERATE_TCE($CE_{PROG}, PROG, POST$)
7: $\quad < X, D, C_{CE} \cup C_{TCE} \cup C_{POST} > \leftarrow$ GENERATE_CSP($CE_{PROG}, TCE, POST$)
8: $\quad C_{LOC} \leftarrow$ ISOLATING-INFEASIBILITY($< X, D, C_{CE} \cup C_{TCE} \cup C_{POST} >$)
9: $\quad LOC \leftarrow Consts\_To\_inst(C_{LOC})$
10: **end if**

Notre approche

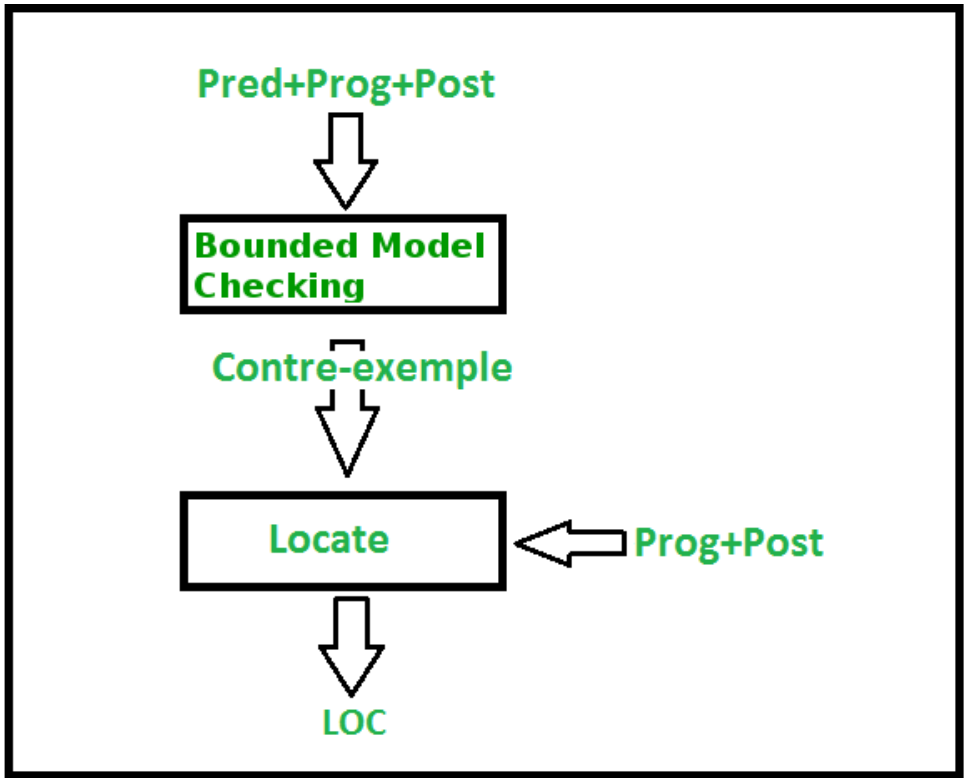

FIGURE: Notre approche pour localiser les erreurs

Modélisation du problème

- Le point de départ est le contre-exemple
  Obtenu par l'usage d'un outil de BMC
- La génération de la trace du contre-exemple
- *CSP* $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$
  - $C_{CE_{PROG}}$ qui correspond à CE_{PROG}.
  - C_{TCE} qui correspond à *TCE*.
  - C_{POST} qui correspond à *POST*.

Algorithme de l'isolation de l'infaisablité basé sur la méthode Deletion Filter

Algorithm 2

Entrée : $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$: Un système de contraintes infaisable.
Sortie : Un conflit minimal dans C_{TCE}.

1: **for** chaque contrainte c_i dans C_{TCE} **do**
2: Temporairement supprimer la contrainte c_i de C_{TCE}.
3: Tester la faisablitÃ© de $C_{CE_{PROG}} \cup (C_{TCE} \setminus c_i) \cup C_{POST}$:
4: **if** feasible **then**
5: retourner La contrainte supprimer à l'ensemble
6: **else**
7: supprimer la contrainte définitivement
8: **end if**
9: Nous prenons l'ensemble des contraintes qui reste dans C_{TCE}
10: **end for**

Algorithme de l'isolation de l'infaisablité basé sur la méthode Additive Method

Algorithm 3

Entrée : $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$: Un système de contraintes infaisable.
Sortie : I est un conflit minimal dans C_{TCE}.

1: $T \leftarrow \emptyset, I \leftarrow \emptyset$.
2: $T \leftarrow C_{CE_{PROG}} \cup C_{POST} \cup I$.
3: **for** chaque contrainte c_i dans C_{TCE} **do**
4: $T \leftarrow T \cup \{c_i\}$.
5: **if** $C_{CE_{PROG}} \cup C_{POST} \cup T$ infaisable **then**
6: $I \leftarrow I \cup \{c_i\}$.
7: Aller à 10.
8: **end if**
9: **end for**
10: **if** $C_{CE_{PROG}} \cup C_{POST} \cup I$ faisable **then**
11: Aller à 2.
12: **end if**

Algorithme de l'isolation de l'infaisablité basé sur la méthode Additive/Deletion method

Algorithm 4

Input : $\mathcal{P} = < X, D, C_{CE_{PROG}} \cup C_{TCE} \cup C_{POST} >$: Un système de contraintes infaisable.
Output : Un conflit minimal dans C_{TCE}.

1: L'ensemble $T \leftarrow \emptyset$.
2: **for** each constraint c_i in C **do**
3: L'ensemble $T \leftarrow T \cup c_i$.
4: **if** $C_{CE_{PROG}} \cup C_{POST} \cup T$ infaisable **then**
5: Aller à 8.
6: **end if**
7: **end for**
8: **for** chaque contrainte t_i dans $t_{|T|-1}$ dans T : **do**
9: Temporairement supprimer la contrainte t_i.
10: Tester la faisablité de $C_{CE_{PROG}} \cup C_{POST} \cup T \setminus t_i$:
11: **if** feasible **then**
12: retourner la contrainte supprimer à l'ensemble T.
13: **else**
14: $T \leftarrow T \setminus t_i$.
15: **end if**
16: **end for**

La résolution du problème

Comparison

- Toutes ces méthodes sont basées sur le principe qui consiste à tester la faisabilité d'un sous-système de contrainte
- La différence entre eux réside dans le nombre de tests de faisabilité effectués
  - La cardinalité de l'ensemble des contraintes de la trace de contre-exemple est n
  - Le cardinal de l'ensemble renvoyé est k
    - Le nombre de tests de faisablité :
    - En utilisant Deletion filter
      Dans tous les cas n
    - En utilisant Additive method
      Dans le pire des cas : $k/2 * (2n - k)$
      In le meilleur des cas : $k/2 * (k + 1)$
    - En utilisant Additive/Deletion method
      Dans le pire des cas : $n + (n - 1)$
      Dans le meilleur des cas : $k + (k - 1)$
    - En utilisant QUICKXPLAIN
      Dans le pire des cas : $2k * log(n/k) + 2k$
      Dans le meilleur des cas : $log(n/k) + 2k$

Implementation

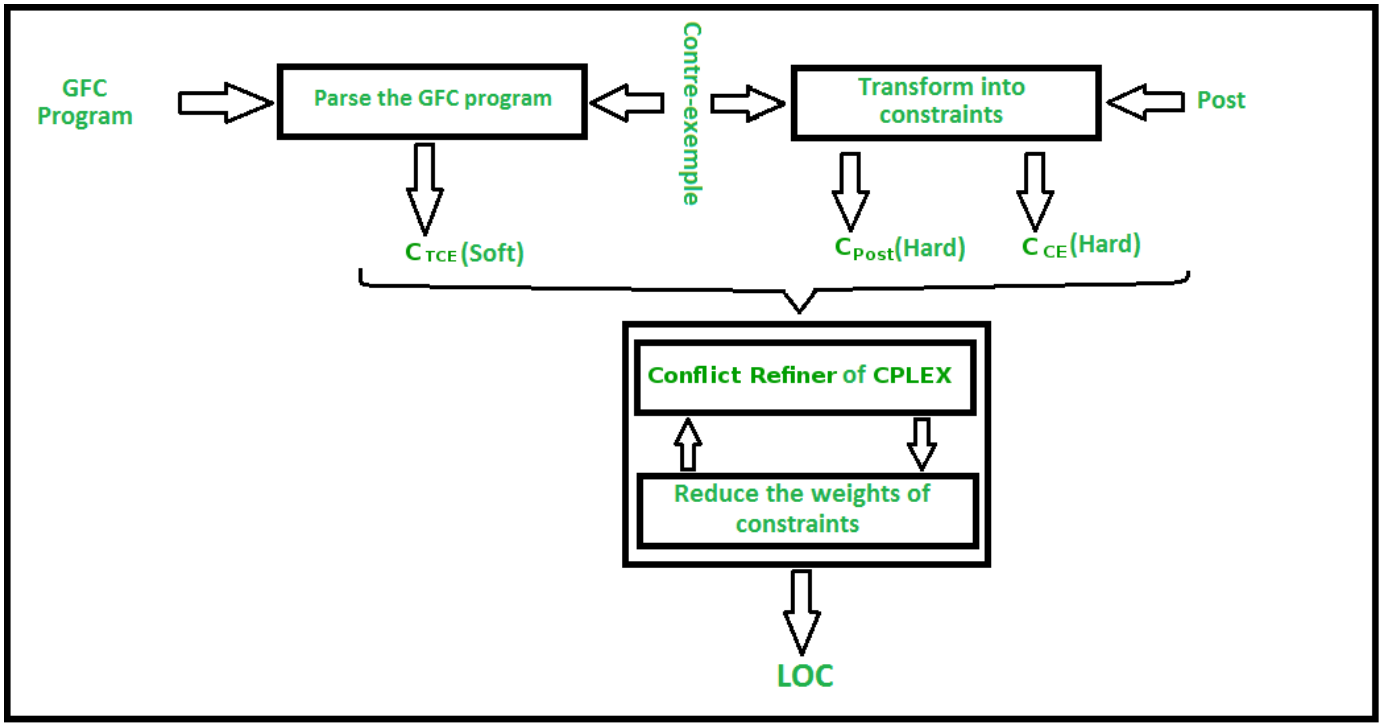

FIGURE: Le processus de localisation d'erreurs

Merci pour votre attention