Université Pierre et Marie Curie
Master Informatique Spécialité SESI
2014-2015

TP de MASSOC

Vérification formelle

Lucien Goubet
Emmanuelle Encrenaz

09 janvier 2015

Exercice 1 : Bonne année !

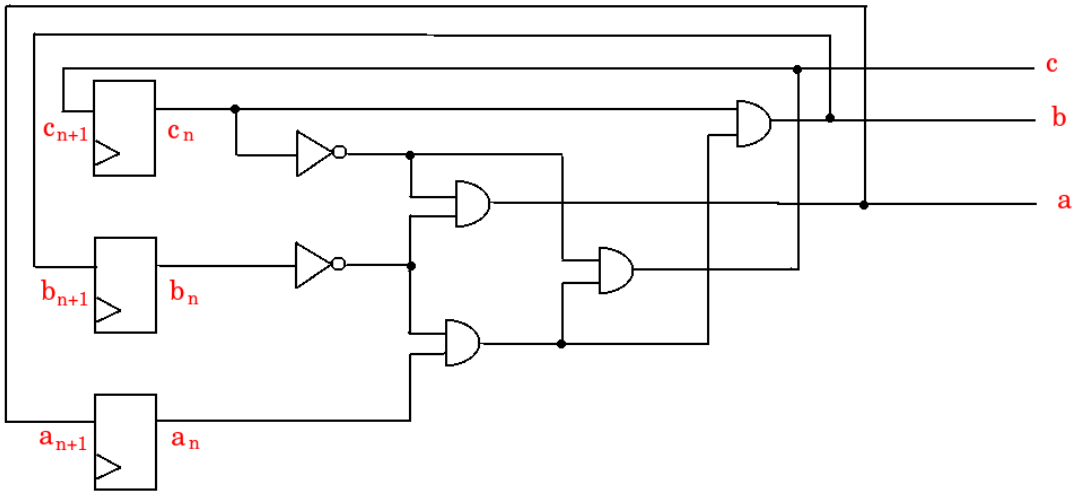


FIGURE 1 – Circuit mystère

Le but de cet exercice est de déterminer le comportement du circuit mystère représenté par la figure1. Pour ce faire nous allons modéliser ce circuit en une formule booléenne afin de prouver certaines propriétés à l'aide d'un SAT-solver.

1. Donner l'expression booléenne des sorties du circuit mystère

$$a_{n+1} =$$
$$b_{n+1} =$$
$$c_{n+1} =$$

2. Modélisation SAT

Les registres *(c,b,a)* possèdent initialement les valeurs *(0,1,0)* respectivement. Nous cherchons à prouver que *(c,b,a) = (0,0,0)* au cycle 1 à l'aide du SAT-solver dans CVC3 [1] :

1. *http : //www.cs.nyu.edu/acsys/cvc3/doc/user_doc.html*

– Modélisez le problème sous la forme : $circuit_0.circuit_1.\overline{prop}$ sachant que $circuit_i$ représente l'état du circuit au cycle i et *prop* la propriété que nous cherchons à vérifier.
– Mettez sous forme conjonctive
– Donnez la formule CNF obtenue au SAT-solver. Qu'en déduisez-vous ?

Le descriptif de CVC3 est donné dans le répertoire $TP\_MASSOC/document\_utiles/cvc3.pdf$ et quelques directives de prise en main sont données page 8 de cet énoncé.

3. Modélisation SMT

CVC3 est un outil permettant de résoudre des problèmes SMT. À la différence des problèmes SAT,
– les problèmes exprimés ne sont pas obligatoirement sous forme CNF
– nous pouvons lui donner des variables autres que booléennes telles que des entiers ou des tableaux de bits
– nous pouvons lui donner des opérateurs complexes tel que la multiplication ou la division.

Sans passer par la forme CNF :
– Prouvez que *(c,b,a) = (0,0,0)* au cycle 1
– Prouvez que *(c,b,a) = (0,0,1)* au cycle 2
– Prouvez que *(c,b,a) = (1,0,1)* au cycle 3
– Prouvez que *(c,b,a) = (0,1,0)* au cycle 4
– Déduisez-en la machine à états du circuit
La machine à état du circuit est cyclique. Sachant que (c,b,a) représente le codage binaire d'un chiffre, quel est le nombre composé par la concaténation des chiffres obtenues aux cycles 0, 1, 2, 3 ?

Cycle	**0**	**1**	**2**	**3**
(c,b,a)	(0, 1, 0)	.....	.....	.....
Chiffre	2	.....	.....	.....

Exercice 2 : Optimisation de codes

Dans cette partie, nous utiliserons les preuves formelles pour optimiser du code assembleur. Le descriptif du jeu d'instructions ARM est donné dans le répertoire $TP\_MASSOC/documents\_utiles/armRefer$
Soit le code suivant :

```
orr x, a, b
and y, a, b
eor s, x, y
```

On suppose que les registres a, b, x, y et z sont tous sur quatre bits. À l'aide de CVC3, prouvez que ce code assembleur est équivalent à

```
eor s, a, b
```

Soit le code suivant :

```
orr x, a, b
and y, a, b
add s, x, y
```

On suppose que les registres a, b, x, y et z sont tous sur quatre bits. À l'aide de CVC3, prouvez que ce code assembleur est équivalent à

```
add s, a, b
```

Exercice 3 : Sécurité

Les attaques physiques permettent de casser des systèmes contenant des informations sensibles. Une attaque physique possible est l'injection de faute par impulsions électromagnétiques. Un modèle de faute possible de cette attaque est le remplacement de l'instruction en cour d'exécution par un nop [2]. Dans cet exercice nous verrons comment outrepasser une sécurité avec ce type d'attaque et comment s'en défendre.

```
if (codeSaisi==codeSecret)
        deverrouiller=1;
else
        deverrouiller=0;
```

Le code C ci-dessus permet de protéger l'accès à des données sensibles (codes bancaires, carte d'accès). Les variables qui y sont utilisées sont des entiers **(vecteur de bits) sur 16 bits**.

1. À l'aide de CVC3, prouvez que le code respecte les propriétés de sécurité suivantes :

1. $(codeSaisi = codeSecret) \Rightarrow deverrouiller = 1$
2. $(codeSaisi \neq codeSecret) \Rightarrow deverrouiller \neq 1$

2. Prouvez que le code suivant respecte également les mêmes propriétés :
$deverrouiller = (\overline{codeSaisi \oplus codeSecret}) + 2$

3. En supposant que le registre r12 contient la variable *deverrouiller*, r11 *codeSaisie* et r10 *codeSecret*, le code assembleur correspondant s'écrit :

```
eor r12, r11, r10
mvn r12, r12
add r12, r12, #2
```

Si la deuxième instruction assembleur est remplacée par un *nop* à cause d'une attaque, est-ce que les propriétés de sécurité sont toujours garanties ? Comment un attaquant peut-il donc déverrouiller le système sans détenir le bon mot de passe ?

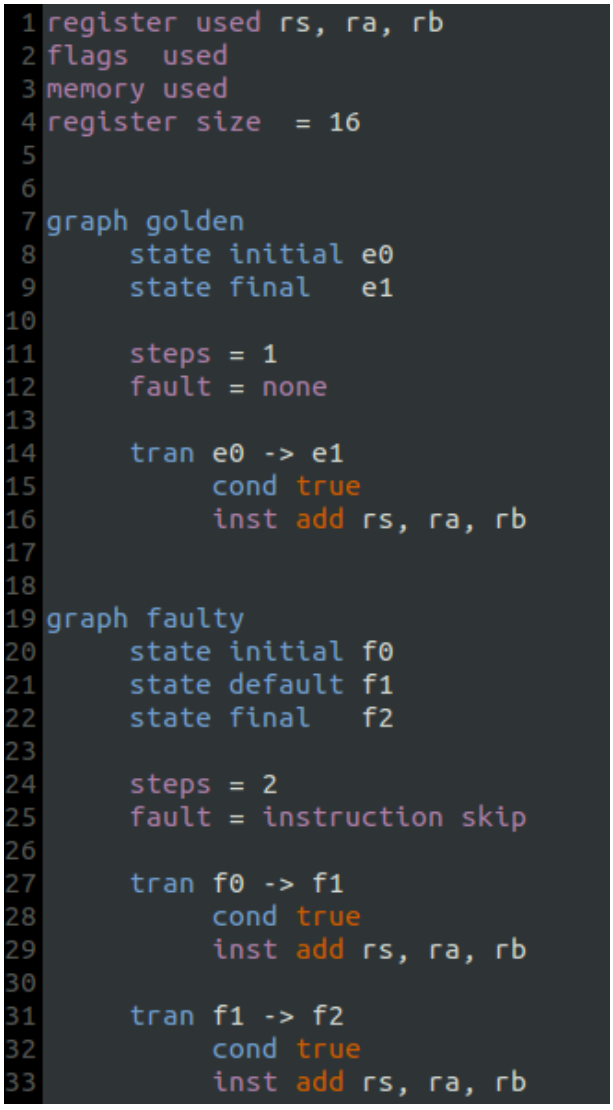
2. No OPeration

4. Il est difficile d'introduire deux fautes dans un intervalle d'exécution très petit. Nous supposons donc que l'attaquant n'est pas capable de fauter deux instructions successives.
En doublant une instruction, on est donc sûr qu'au moins l'un des deux s'exécutera. Il faut néanmoins faire attention à ce que le sens du code ne soit pas modifié.

– Proposez un code assembleur équivalent à celui fourni en **3.** et qui tolère une faute de type *saut d'instruction assembleur* (nop). Vous pourrez utilement vous inspirer de la méthode décrite dans $TP\_MASSOC/documents\_utiles/contremesureProof\_JCEN2014.pdf$

5. MAE[3] est un petit langage permettant de modéliser facilement le problème d'équivalence-checking entre deux codes assembleur. Il a été conçu dans le but de prouver l'efficacité de contre-mesures logicielles contre les fautes de types *saut d'une instruction assembleur* (nop). Pour ce faire les deux codes assembleurs sont modélisés comme des machines à états, ensuite le parseur MAE traduit cette description en un problème SMT pour CVC3. La figure 2 est un exemple de description avec MAE. Vous trouverez une description de ce langage dans $TP\_MASSOC/documents\_utiles/contremesuresASM\_GOUBET2014.pdf$ ainsi que dans $TP\_MASSOC/documents\_utiles/MAE\_instructions\_implementees.pdf$

3. Machine À États



```
register used rs, ra, rb
flags  used
memory used
register size  = 16


graph golden
     state initial e0
     state final   e1

     steps = 1
     fault = none

     tran e0 -> e1
          cond true
          inst add rs, ra, rb


graph faulty
     state initial f0
     state default f1
     state final   f2

     steps = 2
     fault = instruction skip

     tran f0 -> f1
          cond true
          inst add rs, ra, rb

     tran f1 -> f2
          cond true
          inst add rs, ra, rb
```

FIGURE 2 – Fichier de description d'automates MAE pour l'instruction add idempotente

– Prouvez l'efficacité de votre contre-mesure en utilisant le langage MAE

Annexe : Prise en main de CVC3

Prise en main de cvc3

Lancement de l'application

– mode interactif :
  `cvc3 +interactive`
  `CVC> <saisir une ligne de commande>`
  chaque ligne de commande est interprétée avant de saisir la suivante
– mode noninteractif :
  `cvc3 nom_fichier_texte`
  le ficher texte contient la suite des lignes de commandes

Exemple interactif :

```
cvc3 +interactif
CVC> % declarations de variables
CVC> x : BOOLEAN;
CVC> y : BOOLEAN;
CVC> % declaration des contraintes
CVC> ASSERT (x<=>y);
CVC> % recherche solution SAT
CVC> CHECKSAT;
Satisfiable.
CVC> COUNTERMODEL;
Current scope level is 9.
%Satisfiable  Variable Assignment: %
ASSERT NOT x;
ASSERT NOT y;
```

Utilisation des vecteurs de bits :

```
v: BITVECTOR(4);
ASSERT BVMULT(4, v, 0bin0010) /= BVSHL(v, 0bin0001);
CHECKSAT;
```

Les opérateurs relatifs aux vecteurs de bits sont décrits dans les pages 10-11 du manuel utilisateur de cvc ($TP\_MASSOC/documents\_utiles/cvc3\_manual.pdf$)