SORBONNE UNIVERSITÉ

Automne 2020

TP Preuve d'équivalence de circuits logiques

TP 1

Objectifs

- ✯ Comprendre l'équivalence checking de circuit combinatoire
- ✯ Maitriser le format CNF et le format DIMACS
- ✯ Comprendre la transformation circuit vers CNF

Rendu

- ✏ Le fichier C ou les fichiers C++.
- ✏ Un document qui explique les transformations et qui résument vos expériences avec les bug que vous auriez pu trouver dans le benchmark.

1 Problème à résoudre

L'objectif de ce TP est d'implémenter un *equivalence checker* pour des circuits combinatoires. Votre programme prendra deux fichiers en paramètres contenant des circuits. Il permettra de générer une représentation DIMACS du problème d'équivalence entre ces 2 circuits qui pourra être fournit à un SAT solver pour résoudre le problème.

Pour tester votre programme, vous utiliserez le benchmark ISCAS 85. C'est un benchmark standard qui contient un ensemble de circuits combinatoires. Bien que ce benchmark soit très vieux il est toujours utilisé dans des articles même récent. Les circuits ont des noms de la forme "c1908". Vous utiliserez votre programme pour vérifier la version des circuits du répertoire `DATA` avec une version optimiser du même circuit dans le répertoire `NONREDUN`. Ces circuit sont censées être les mêmes. Dans le même répertoire vous trouverez une version plus ancienne qui était censé être équivalente mais qui contenait des bugs.

La documention explique comment ce lit le format des fichiers du benchmark. Le fichiers qui vous sont fournis contiennent déjà la lecture (parsing) pour ce format de fichier.

Pendant ce TP vous aller écrire la transformation d'un circuit donner sous la forme de ce benchmark en format DIMACS et coder le problème d'équivalence, un SAT solver sera utiliser pour résoudre l'équivalence :

- si l'instance est SAT les circuits sont différents et la configuration trouver vous donne une configuration des entrées qui démontre la non-équivalence ;
- si l'instance est UNSAT les deux circuits sont équivalents.

Vous allez avoir besoin d'un SAT solver pour terminer le TP. Vous pouvez utiliser les SAT solver utiliser lors de la dernière SAT compétition (plingeling et glucose, s'installe facilement. Un sat-solver parallèle painless vient du LIP6).

Vous pouvez choisir de faire ce TP en C ou en C++.

Code en C Un squelette de programme (skeleton.c) vous est fournit. Il sait :

- lire les fichiers ISCAS85,
- remplir une structure de données correspondant à un circuit en mémoire,
- Générer les clauses CNF pour un sous-ensemble des opérateurs.

Code en C++ Le code fournit les structures pour représenter un circuit sous la forme de listes d'entrées/sorties, de signaux internes et un ensemble de portes accessible via leur signal de sortie (le clé du map). Le code fournit aussi :

— Un parser pour lire les fichiers ISCAS85,
— Le parser remplit une structure de données correspondant à un circuit en mémoire,
— Générer les clauses CNF pour un sous-ensemble des opérateurs.

Le générateur de clauses est le visiteur `CNFvisitor`.

Exercice 1 Forme normale conjonctive CNF

L'objectif ici est de compléter le squelette pour pouvoir générer un CNF sous format DIMACS.

Question 1
Quelles sont les transformations Tseytin pour les portes NAND, OR, NOR, XOR et NOT ?

Question 2
Implémentez les parties du code qui implémentent ces transformations et génère le format DIMACS. Le CNF sera afficher sur la sortie standard.

Question 3
Tester votre code avec les petits circuits des portes fournit ainsi que le `small.circ`

Exercice 2 Satisfaisabilité du CNF

Question 1
Tester votre code avec un petit circuit comme par exemple un *adder*.

Question 2
Récupérer le code DIMACS généré dans un fichier et utiliser un SAT solver pour obtenir une solution de votre CNF.

Question 3
A quoi correspond la solution ?

Exercice 3 Preuve d'équivalence

Pour obtenir la preuve d'équivalence nous devons construire la structure *miter* qui met en parallèle les deux circuits et ajoute la comparaison des sorties. Au lieu de simuler ce circuit pour toutes les entrées possible (ce qui prendra beaucoup de temps), on utilise la transformation Tseytin et un SAT solver. La question d'équivalence devient donc :

Est-ce qu'il existe une affectation des entrées de notre miter tel que la seule sortie du circuit prend la valeur 1 ?

Si elle existe, il s'agit d'un contrexemple qui démontre la non-équivalence. Sinon, les circuits sont vraiment équivalent.

Question 1
Écrire une fonction qui check que les nombres d'entrées et si leur nom sont identiques pour les deux circuits.

Question 2
Écrire une fonction qui check les sorties

Question 3
Écrire une fonction qui connecte les entrées entre elles en CNF..

Question 4
Ecrire la logique de comparaison du miter en CNF.

Question 5
Ajouter la contrainte sur la sortie pour la forcer en 1 en CNF.

Question 6
Testez votre programme à l'aide d'un SAT solver avec le benchmark ISCAS85 comme expliqué au début du TP.

Question 7
(BONUS) Intégrer l'appel au SAT solver dans votre code.