Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document constitue une fraude caractrisée.
Nom, date et signature :

C - *Horn*

RAYNAUD Paul
·
Supervised by : PERIN Michaël

June 2018

missing
Résumé Être capable de vérifier si un programme réalise ce que sa spécification spécifie de manière automatique est quelque chose d'utile, car au lieu de montrer l'absence d'erreurs que nous capturons avec des tests, nous sommes capables de montrer la correction du programme. Nous chercherons donc a prouver a partir d'un programme source si celui ci est capable de satisfaire les propriétés qui lui sont attachées ou non le tout de manière automatique. Nous nous baserons sur des principes déjà connu tel que la Logic de Hoare, et l'étude de graphe de flot de contrôle pour propager les propriétés établis à un endroit du programme à tout le programme, en ayant recours à un SMT solver pour vérifié les formules générés. Mon stage étant sur plus d'un mois (magistère) les résultats actuels se limite à la mise en place de bonnes conditions pour travailler réellement sur le sujet.

Keywords Compcert · Program verification · Automatic Verification · Logic

1 Problème

1.1 mini-intro

Des nos jours une la rencontre de bugs sur des systèmes informatiques est quelque chose de banalisé, mais ces systèmes étant présent partout, et notamment sur des systèmes critiques (santé, embarqué, ...), nous aimerions être capable de prouver qu'un logiciel est capable de répondre aux contraintes décris dans sa spécification.

Paul RAYNAUD
361 allée de Hector Berlioz 38400
Tel. : 06 15 54 38 90
E-mail: paul.raynaud66@hotmail.fr

1.2 precision

Nous souhaiterions réaliser un outil capable de vérifier si dès lors de sa compilation, un programme est capable de satisfaire les propriétés auquel il prétend ; mais comment vérifier cela ? Tout d'abord nous aurons besoin de définir sur quel language est-il le plus judicieux travailler ; transmettre ensuite les proriétés d'une partie du programme à tout le programme ; puis vérifié si ces propriétés sont satisfaisables.

Le language "c" est le language le plus plus utilisé dans les systèmes embarqué, qui sont également nos systèmes critiques, par conséquent nous travaillerons sur un (*"vérificateur"*) du language c.

Les principes de vérification de programme sont connus et étaient établis depuis longtemps, par des personnes comme Hoare et Floyd, et il est naturellement plus simple d'utilisé ces principes sur des graphes de flot de contrôle pour générer les formules logiques, représentant les propriétés, lié a chaques noeuds. La manière naturel de représenter ces formules logique sont les clauses de Horn, qui sont par ailleurs l'une des entrées standards utilisés par la majorité des SMT solvers qui eux permettent de résoudre ces formules logiques.

transition : - Le language C étant un (vieux/bas niveau) language il s'avère que sa sémantique est floue sur certains points et diffère d'un compilateur à l'autre,ce qui entraine des miscompilation, ce qui généré des éxécution incorrecte de programme pourtan correct. C'est pour cette raison que le compilateur Compcert a vu le jour, il est (actuellemnt) le seul compilateur prouvé en c, et c'est donc sur sa sémantique que nous nous baserons pour notre outil.

2 Travaux relatés

2.1 accroche

Des outils capable de réaliser la vérification de programmes ont déjà était développés, par exemple SeaHorn. Cependant pour le "récent" (ERTS 2016 : Embedded Real Time Software and Systems, 8th European Congress, Jan 2016, Toulouse, France) compilateur Compcert, avec sa sémantique unique, ce serait un nouvel atout d'être capable de vérifier automatiquement un programme. On pourrait ainsi avoir un programme prouvé sans qu'un compilateur à la sémantique "hasardeuse" vienne intégrer des erreurs lors de la compilation, sur un code dont la spécification est prouvé.

2.2 SeaHorn

http ://seahorn.github.io/ Un outil comme SeaHorn repose sur le mêmes principes que ceux utilisé dans ce stage, cependant à la différence que SeaHorn prend des programme C/C++ (compilable par clang) et utilise le bitecode généré par clang. Clang étant un compilateur différent nous n'avons pas les

mêmes propiétés que celles de Compcert sur le code généré, de ce fait nous ne pouvons pas utilisé SeaHorn car clang et Compcert fournissent2 interprétations du C différentes.

La raison pour laquelle nous ne pouvons réutiliser les outils déjà existant faisant de la vérification de programme est lié au fait que Compcert est un compilateur "récent" et que ces outils n'ont pas étaient developpé son interprétation du C.

transition : Yang et al (2011) ont réalisé une étude visant a montré les miscompilation des différents compilateurs C, le résultat montré est que tous les compilateurs qu'ils ont testé ont généré des code incorrect pour certains tests (incluant compcert), rappelons qe Compcert n'était pas terminé en 2011, mais il y avait déjà des résultats significatifs "The striking thing about our CompCert results is that the middle- end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying : we have devoted about six CPU-years to the task. The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users".

2.3 CompCert

Compcert est projet démarré en 2008 ? visant à créer le premier compilateur C prouvé. C'est un projet à la structure complexe entièrement prouvé et majoritairement développé en coq, a partir du quel il est possible d'extraire du caml éxecutable.

3 Objectif global

3.1 mini-intro

L'objectif est d'intégrer cet outil à Compcert, de manière a ce qu'il vérifie lors de la compilation si le programme est cohérent avec les propriété qui lui sont attachés.

3.2 Stage M1

3.3 quelque mot

4 Contributioin / Travail rélaisé

4.1 théorique

Nous nous somme intégré dans compcert au premier endroit ou compcert créé un CFG, de manière à propager les propriétés de manière simple, les principes des propagations des propriétés sont des choses connues depuis un moment cependant (logique de Hoare), pour les mettre en application sur compcert il faudra tout de même les réécrire pour les adapter à la syntaxe du language que nous exploitons.

4.2 pratique

Premièrement il faut d'abord se rattacher à compcert, un projet important, sans gener ce que le compilateur réalise. Ainsi nous nous somme intégrer de manière indépendante, cependant les structures de données manipulées par compcert n'était pas directement accessible. Pour les récupérer nous avons du utiliser une librairie ocaml (ppx_deriving.show) particulière pour récupérer le code de compcert. Le principe de la librairie est de générer automatiquement des fonctions capable d'afficher un type, qu'il soit un type somme, enregistrement ou autre, pour cela nous avions besoin de modifier les fichiers où étaient définis ces types que nous voulions afficher, ainsi que les fichier où l'on definissait les types de bases.

```
type test =
        | A of int
  [@@deriving show]
;;

type tree =
  | U of tree
  | B of tree * tree
  | L of int
        | T of test
  [@@deriving show]
;;
```

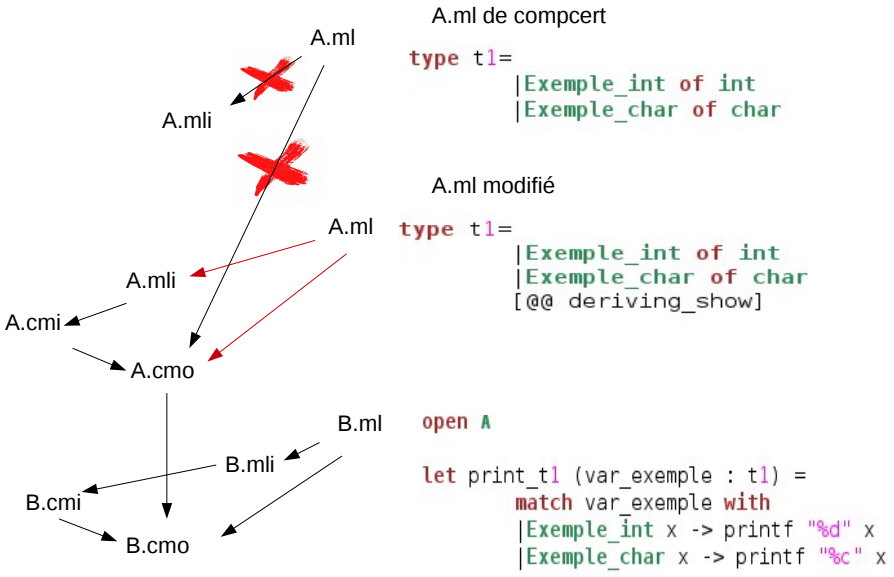
```
open Tree

let output_ocaml_decl_of_tree: string -> tree -> unit
=
  fun name tree ->
    (String.concat "\n" [ "open Tree" ; "let " ^ name
^ " =" ; show_tree tree ; ";;"])
    |> print_string
;;

let t = B(U(L 1), B (L 2, B( (L 3), (T(A 3)) ) )  ) ;;
let _ =
       output_ocaml_decl_of_tree "tree1" t ;;
```

```
sule:/local/raynaudp/TER/projet/CC2HC/ressource/generic_printer$ ocamlfind oca
mlc -c -package ppx_deriving.show tree.ml
sule:/local/raynaudp/TER/projet/CC2HC/ressource/generic_printer$ ocamlfind oca
mlc -o exec -linkpkg -package ppx_deriving.show tree.cmo main.ml
sule:/local/raynaudp/TER/projet/CC2HC/ressource/generic_printer$ ./exec
open Tree
let tree1 =
(Tree.B ((Tree.U (Tree.L 1)),
   (Tree.B ((Tree.L 2), (Tree.B ((Tree.L 3), (Tree.T (Tree.A 3)))))))
```

exemple utilisation ppx_deriving.show

Cependant les fichiers caml contenant les types (les structures) de bases de la structure d'un programme, sont extrait/généré lors de la compilation (de compcert), plus clairement les fichier .ml étaient généré à la compilation de compcert, ce qui effacait alors toutes nos modifications à chaque recompilation de compcert.

Nous avons alors décidé de sauvegarder nos fichiers modifié (.ml) dans un répertoire particulier, les compiler de manière indépendantes pour générer leurs interfaces et les fichiers intermédiaires (.mli, cmi), car le .cmi dépendait des .mli et ces-mêmes .mli étaient générés par compcert au même moment que les .ml, cependant si un .ml n'était pas cohérent avec son interface (.mli) alors le fichier était rejeté a la compilation.Puis nous les avons réinjecté au moment de la compilation de compcert pour que celui ci créé lui même les .cmo et que l'on soit capable d'afficher la structure que nous voulions (en modifiant légérement un printer déja existant).



exemple simple de la compilation

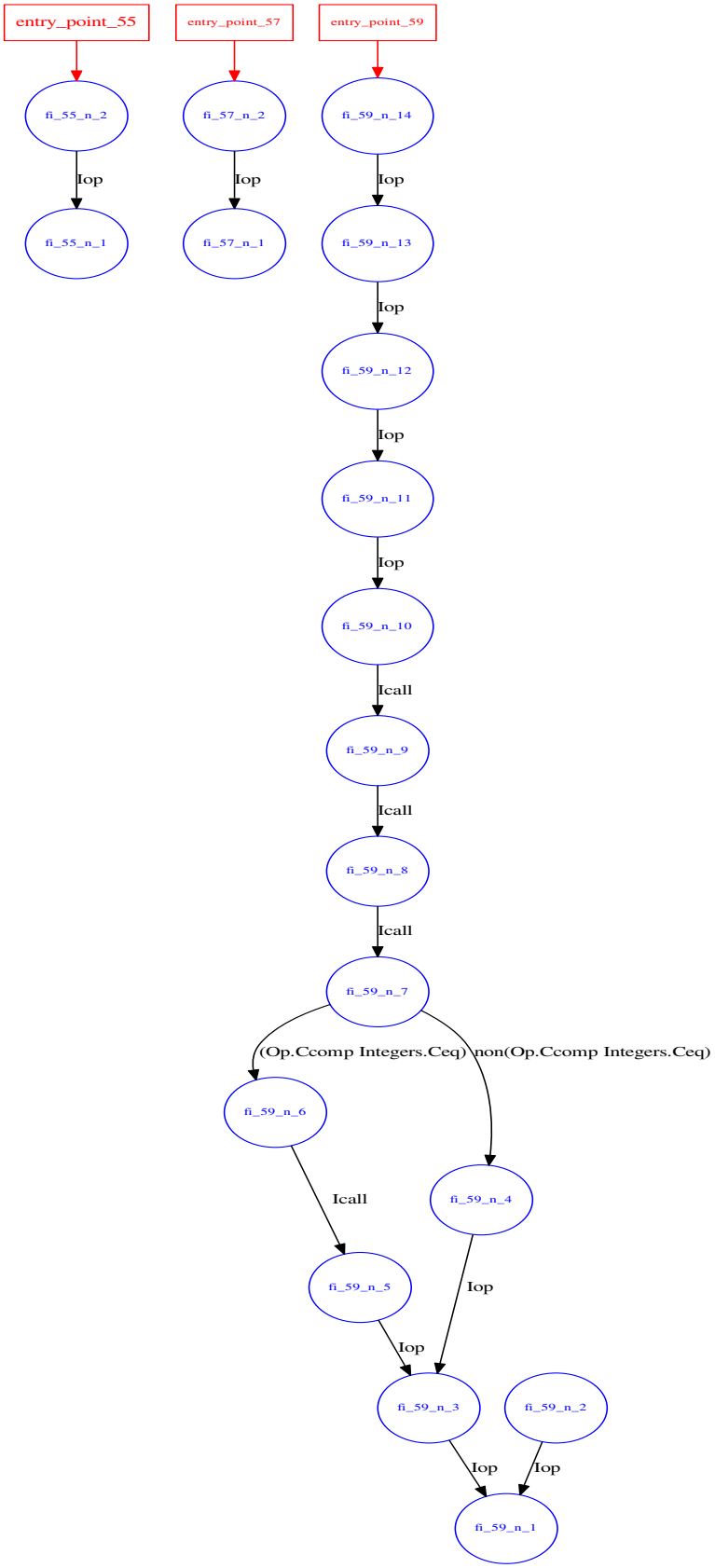
Une fois la structure du code compilé par compcert extraite, nous obtenions une liste de definition de fonctions, ou chacune d'elles représentaient leur code par un arbre d'instructions.

EXEMPLE du code prendre une instruction simple et la détaillé.

En retriant les instructions de l'arbre, et en les modifiant nous avons réaliser une première nouvelle structure qui sied mieux a la representation d'un CFG une structure par triplet.

EXPLICATION DE LA STRUCTURE

Une fois notre arbre de départ transformé en un CFG plus simple a comprendre nous avons developpé un printer permettant de créer un fichier .dot et d'afficher sous forme de graphe visuel notre structure de CFG en utilisant le logiciel graphviz.



control flow graph (CFG) EXEMPLE DE GRAPH avec le code c a cote

En developpant un peu plus nous avons réalisé une seconde structure plus en accord avec la logique de Hoare qui se veut plut "remonter" l'arbre en partant du bas quadrulet CFG.

EXPLICATION DE LA SECONDE STRUCTURE : ajout des predecesseur

Maintenant que nous pouvons demarrer sur des bases plus solides nous devrions attaquer la transmission de propriétés à travers l'arbre.

5 Travaux futures

5.1 mini-intro

Le but de ce stage est de commencer ce compilateur de C vers clauses de Horn, c'est a dire que le lors de ce stage nous devons developper un premier outil capable de générer des clauses pour un sous-ensemble (plutôt simple) des instructions de C. Mais ce n'est pas en soit une finalité.

5.2 aller plus loin 1

Une seconde étape serait de compléter cet outil pour être capable de traiter tout le C, comme ce que réalise déjà les autres "verifieur" (SeaHorn, Boogie ...), car si on ne supporte pas toutes les instructions supporté par Compcert l'outil serait inutilisable.

5.3 aller plus loin 2

Une fois le C traité par Compcert traité au complet il faudrait rester dans la même logique que Compcert, il faudrait prouver ce que nous avons réaliser jusque là, ce qui d'après Mr. Périn serait le travail d'une thèse.

Si le projet arrivait à son terme, alors un programme compilé par Compcert (et notre outil) pourrait alors

Text with citations [2] and [1].

as required. Don't forget to give each section and subsection a unique label (see Sect. 5).

Paragraph headings Use paragraph headings as needed.

$$a^2 + b^2 = c^2 \tag{1}$$

Références

1. Author, Article title, Journal, Volume, page numbers (year)
2. Author, Book title, page numbers. Publisher, place (year)

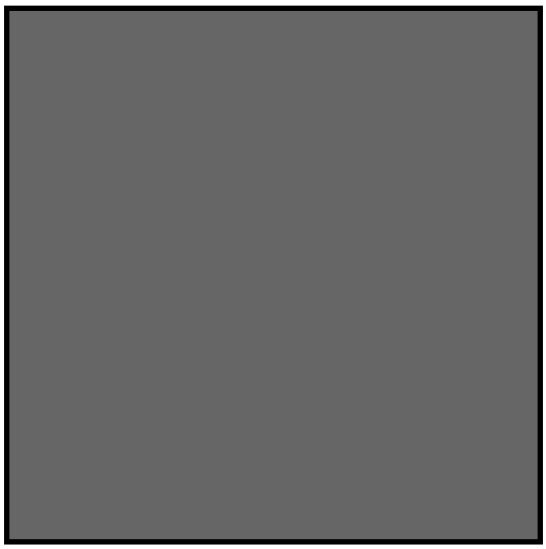
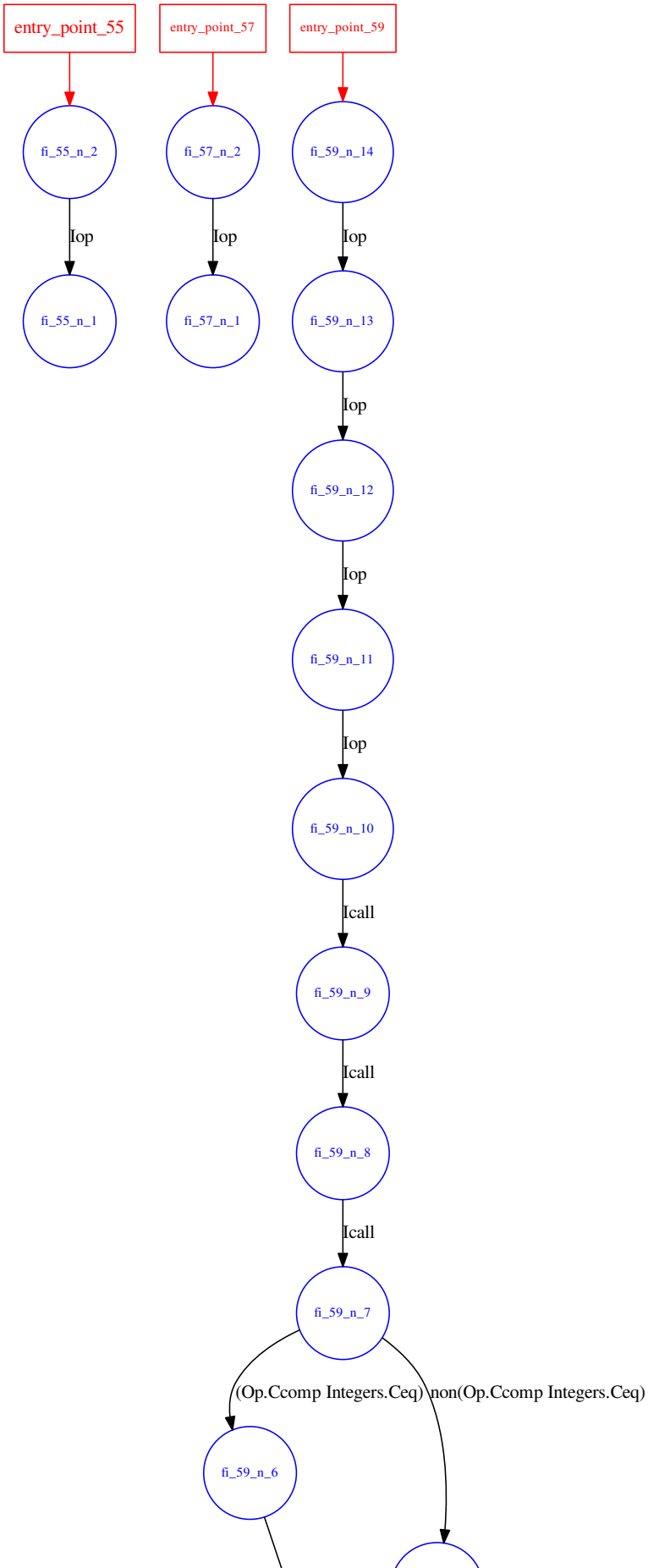


Figure 1 Please write your figure caption here

Table 1 Please write your table caption here

first	second	third
number	number	number
number	number	number


entry_point_55
entry_point_57
entry_point_59
fi_55_n_2
fi_57_n_2
fi_59_n_14
Iop
Iop
Iop
fi_55_n_1
fi_57_n_1
fi_59_n_13
Iop
fi_59_n_12
Iop
fi_59_n_11
Iop
fi_59_n_10
Icall
fi_59_n_9
Icall
fi_59_n_8
Icall
fi_59_n_7
(Op.Ccomp Integers.Ceq)
non(Op.Ccomp Integers.Ceq)
fi_59_n_6