Removing Unnecessary Variables from Horn Clause Verification Conditions

E. De Angelis (1), F. Fioravanti (1)
A. Pettorossi (2), M. Proietti (3)

(1) DEC, University ”G. d’Annunzio” of Chieti-Pescara, Italy
(2) DICII, University of Rome Tor Vergata, Roma, Italy
(3) CNR-IASI, Roma, Italy

HCVS'16 – Eindhoven

Talk Outline

- Partial Correctness properties
- Verification Conditions Generation
  - using specialization of Constrained Horn Clauses (CHC) a.k.a. Constraint Logic Programs (CLP)
- Removing unnecessary variables from CHC
  - Non-Linking variables Removal strategy
    - call dependent
  - Constrained FAR algorithm
    - call independent
    - Variable liveness analysis
- Experimental evaluation

Partial Correctness and VCs

Given the partial correctness property (Hoare triple)

$\{x \geq 0\}$

```
int  x,y;
main () {
     int z=x+1;
     while (z<=9) {z=z+1;}
     y=z;
}
```

$\{y > 0\}$

Verification Conditions: formulas whose satisfiability implies correctness

….. as constrained Horn clauses

```
incorrect  :-   X1>=0,  newp1(X1,Y1,  X2,Y2), Y2=<0.
newp1(X1,Y1,X2,Z2) :- Z1=X1+1, newp2(X1,Y1,Z1,X2,Y2,Z2).
newp2(X1,Y1,Z1,X2,Y2,Z3) :- Z1=<9, Z2=Z1+1,newp2(X1,Y1,Z2,X2,Y2,Z3)
newp2(X1,Y1,Z1,X1,Y1,Z1) :- Z1>=10.
```

VCs satisfiability can (possibly) be checked by using Horn solvers and Satisfiability Modulo Theory (SMT) solvers like

- CHA (Gallagher et al.), Duality (McMillan), Eldarica (Ruemmer et al.), MathSAT (Cimatti et al.), QARMC/HSF (Rybalchenko et al.), SeaHorn (Gurfinkel et al.), TRACER (Jaffar et al.), VeriMAP (De Angelis et al.), **Z3** (Bjorner & De Moura),

VCs GENeration

Standard approach

- VCGEN algorithm is tailored to the syntax and the semantics of the imperative programming language
- **Cons**: changing the programming language or its semantics usually requires **rewriting** the VCGEN algorithm

Semantics-based approach

[Cousot SAS'97, Gallagher et al. SAS'98, J Strother Moore CHARME'03, Rosu et al '14]

- VCGEN algorithm is **parametric** wrt programming language semantics
- **Pro**: use the same VCGEN algorithm for different programming languages and semantics

Our semantics-based approach

- uses CHC encoding of program, semantics and logic
- VCs generated by CHC specialization
  - correctness of VC generation follows from correctness of the rules
- Parametricity wrt programming language and class of properties
- Flexibility and efficiency

Encoding Imperative Programs

- Imperative language: subset of CIL (C Intermediate Language)
  - assignments, conditionals, jumps, recursive function calls, abort
  - loops translated to conditionals and jumps
- Commands encoded as facts: **at(Label, Cmd)**

Program *Prog*

```
int x, y;

void main() {
    int z=x+1;          l1
    while (z<=9) {      l2
        z=z+1;          l3
    }                   l4
    y=z;                l5

}
```

CLP encoding of *Prog*

```
fun(main,[],[],1).
at(1,asgn(z,plus(x,1))).
at(2,ite( lteq(z,9),3,5)).
at(3,asgn(z,plus(z,1))).
at(4,goto(2)).
at(5,asgn(y,z)).
at(h,halt).
```

Encoding the Operational Semantics

Configurations: **cf(LC, Env)** program execution state

- **LC** labeled command: a term of the form **cmd(L,C)**
  - **L** label, **C** command
- **Env** environment: a pair **(D,S)**
  - **D** global enviroment, **S** local environment
  - Environments as lists of pairs **[(x,X),(y,Y),(z,Z)]**

Operational semantics: transition relation **tr** between configurations

tr(cf(LC1,E1), cf(LC2,E2))

Multiple steps reachability (reflexive, transitive closure of **tr**)

```
reach(C,C).
reach(C,C2) :- tr(C,C1), reach(C1,C2).
```

Encoding the Operational Semantics

<u>assignment</u> x=e;

```
tr( cf(cmd(L,  asgn(X,expr(E))), (D,S)),            source configuration
     cf(cmd(L1,C),                (D1,S1))) :-      target configuration
      eval(E,(D,S),V),                              evaluate expression
      update((D,S),X,V,(D1,S1)),                    update environment
      nextlab(L,L1),                                next label
      at(L1,C).                                     next command
```

Encoding Partial (In)Correctness

Partial correctness property

$$\{x \geq 0\}\ \ \mathit{Prog}\ \ \{y > 0\}$$

CHC encoding of (in)correctness. program I

```
incorrect :- initConf(Cf), reach(Cf,Cf1), errorConf(Cf1).
                              …
  initConf(cf(C, [(x,X),(y,Y)])) :- at(1,C), X>=0.
errorConf(cf(C, [(x,X),(y,Y)])) :- at(h,C), Y=< 0.
```

Thm. Correctness of CLP Encoding

property does not hold iff incorrect $\in$ M(I)

where: M(I) least LIA model of the CLP program I

Undecidable problem. Even if decidable, very hard to check.

Unfold/Fold program specialization for “removing the interpreter” and producing VCs.

Partial Correctness and VCs

Given the partial correctness property (Hoare triple)

{x ≥ 0}

```
int  x,y;
main () {
      int z=x+1;
      while (z<=9) {z=z+1;}
      y=z;
}
```

{y > 0}

Verification Conditions as constrained Horn clauses

incorrect :- X1>=0, newp1(X1,Y1, X2,Y2), Y2=<0. — program execution (call to the main() function)

newp1(X1,Y1,X2,Z2) :- Z1=X1+1, newp2(X1,Y1,Z1,X2,Y2,Z2). — loop initialization

newp2(X1,Y1,Z1,X2,Y2,Z2) :- Z1=<9, Z3=Z1+1,newp2(X1,Y1,Z3,X2,Y2,Z2) — loop iteration

newp2(X1,Y1,Z1,X1,Y1,Z1) :- Z1>=10. — loop exit

Unnecessary variables

- It is well-known that transformational approaches may produce unnecessary variables

- Two solutions from LP (adapted to CHC) for removing (some) unnecessary variables
  - Non-linking variables strategy
    - call dependent
  - Constrained FAR algorithm
    - call independent
    - variable liveness analysis

Non-Linking variables Removal

Let C be a clause of the form H :- c, L, B , R

A variable occurring in B is **non-linking** in C if it does not occur in the rest of the clause

Non-linking variables can be removed from the call

Verification Conditions after VCG

incorrect :- X1>=0, newp1(X1,Y1, X2,Y2), Y2=<0.
newp1(X1,Y1,X2,Y2) :- Z1=X1+1, newp2(X1,Y1,Z1,X2, Y2, Z2).
newp2(X1,Y1,Z1,X2,Y2,Z2) :- Z1=<9, Z3=Z1+1,newp2(X1,Y1,Z3,X2,Y2,Z2)
newp2(X1,Y1,Z1,X1,Y1,Z1) :- Z1>=10.

Verification Conditions after application of the NLR strategy

incorrect$_{NLR}$:- X1>=0, newp3(X1,Y2), Y2=<0.
newp3(X1,Z2) :- Z1=X1+1, newp4(X1,Z1,Z2).
newp4(X1,Z1,Z2) :- Z1=<9, Z3=Z1+1,newp4(X1,Z3,Z2)
newp4(X1,Z1,Z1) :- Z1>=10.

NLR strategy

```
Input: a set VC of CHCs
Output: VC_NLR

VC_NLR := ∅;
Defs := {incorrect_NLR :- incorrect };

while  there exists d in Defs to be processed  do
   Cls = UNFOLDING(d,VC);
   Defs = Defs  U   DEFINITION-INTRODUCTION(Cls);
   VC_NLR = VC_NLR  U   FOLDING(Cls, Defs);
   mark d as processed;
done
```

Thm. Termination and correctness of the NLR strategy

(i) the NLR strategy terminates

(ii) incorrect $\in$ M(VC) iff incorrect$_{NLR}$ $\in$ M(VC$_{NLR}$)

NLR strategy in action

incorrect$_{NLR}$:- incorrect

- **UNFOLDING** (replace leftmost atom incorrect with the body of its definition)

  incorrect$_{NLR}$:- X1>=0, newp1(X1,Y1, X2,Y2), Y2=<0.

- **DEFINITION-INTRODUCTION** (add a clause with a new head predicate and linking vars)

  d1: newp3(X1,Y2) :- newp1(X1,Y1, X2,Y2)

- **FOLDING** (replace an instance of the body of a definition by its head)

  incorrect$_{NLR}$:- X1>=0, newp3(X1,Y2), Y2=<0.

- **UNFOLDING** (of d1)

  newp3(X1,Z2) :- Z1=X1+1, newp2(X1,Y1,Z1,X2, Y2, Z2).

- **DEFINITION-INTRODUCTION**

  d2: newp4(X1,Z1,Z2) :- newp2(X1,Y1,Z1,X2, Y2, Z2).

- **FOLDING**

  newp3(X1,Z2) :- Z1=X1+1, newp4(X1,Z1,Z2). …. continues ...

NLR strategy in action

- **UNFOLDING**

  newp4(X1,Z1,Z2) :- Z1=<9, Z3=Z1+1, newp2(X1,Y1,Z3, X2, Y2, Z2).

  newp4(X1,Z1,Z1) :- Z1>=10.

- **DEFINITION-INTRODUCTION** (no new definition, **reuse** already introduced definition)

  d2: newp4(X1,Z1,Z2) :- newp2(X1,Y1,Z1,X2, Y2, Z2).

- **FOLDING**

  newp4(X1,Z1,Z2) :- Z1=<9, Z3=Z1+1, newp4(X1,Z3,Z2).

Verification Conditions after NLR

incorrect$_{NLR}$:- X1>=0, newp3(X1,Y2), Y2=<0.
newp3(X1,Z2) :- Z1=X1+1, newp4(X1,Z1,Z2).
newp4(X1,Z1,Z2) :- Z1=<9, Z3=Z1+1,newp4(X1,Z3,Z2)
newp4(X1,Z1,Z1) :- Z1>=10.

NLR strategy - generalization

- What if there are calls to the same predicate having different sets of linking variables?
  - r(X) :- X>0, p(X,$\underline{Y}$,$\underline{Z}$).   s(Y) :- Y=1, p($\underline{X}$,Y,$\underline{Z}$).
- We could introduce a definition for every different set of variables
  - d1: newp1(X) :- p(X,$\underline{Y}$,$\underline{Z}$).
  - d2: newp2(Y) :- p($\underline{X}$,Y,$\underline{Z}$).

  Risk of exponential increase of the number of definitions !

- Assume that d1 is currently the only definition for p(X,Y,Z)

  instead of introducing d2, we replace d1 with

  d3: newp3(X,Y) :- p(X,Y,$\underline{Z}$).

  intersection of non-linking variables (i.e. union of head variables)

- Thus, VCs after NLR have the same size (number of predicates and clauses) of the input VCs, but hopefully less variables.

Constrained FAR - motivation

Verification Conditions after NLR

...

newp4(X1,Z1,Z2) :- Z1=<9, Z3=Z1+1,newp4(X1,Z3,Z2)
newp4(X1,Z1,Z1) :- Z1>=10.

- variable X1 plays no role in the (model of) newp4
  … it does not occur in the constraints and it does not “change”

newp4(X1,Z1,Z2) holds iff newp4(~~X1,~~ Z1,Z2) holds

- … but X1 could not be removed by NLR

  We extend to CHC the FAR algorithm [Leuschel et al, '96]

Constrained FAR

- An **erasure** E is a set of pairs (p,k) where p is a predicate symbol of arity n and $1 \leq k \leq n$
- Given an erasure E={(p,2), (q,1)} and clause C: r(X,Y,Z) :- X=Z, p(X,Y), q(Z).

  the erased clause C_E: r(X,Y,Z) :- X=Z, p(X), q.

- Erasure E is **safe** for P iff for all (p,k) $\in$ E and for all $p(X_1,\ldots,X_n)$:- c, G in P
  - X_k is a variable and $\mathcal{A} \models \forall X_k \exists Y\, c$ where $Y = vars(c) - \{X_k\}$
  - X_k is not constrained to any other variable in H
  - X_k is not constrained to any variable in G_E

- If E is a safe erasure for program P then for all atoms B

$$B \in M(P) \quad \text{iff} \quad B_E \in M(P_E)$$

Constrained FAR algorithm

Let $E = \{(p,k) \mid p \text{ of arity } n \text{ and } 1 \leq k \leq n\}$ be the full erasure

repeat

if E is an unsafe erasure due to some $(p,k) \in E$

then $E = E - \{(p,k)\}$

until E is a safe erasure

<u>Thm. Termination and correctness of the cFAR algorithm</u>

The cFAR algorithm terminates and

$$incorrect \in M(P) \quad \text{iff} \quad incorrect_E \in M(P_E)$$

NLR vs cFAR

- NLR and cFAR are incomparable in general
- cFAR cannot erase variables that occur multiple times in the head of a clause

```
q(Z) :-  p(X,Y,Z).
p(X,X,Z).
```

… but NLR can

```
newq(Z) :- newp(Z).
newp(Z).
```

Experimental evaluation

- 320 verification problems written in the C language
  - from TACAS SV-COMP, other public benchmarks
- Z3 with default options (slicing on)

		VCG; Z3	VCG; NLR; Z3	VCG; NLR; cFAR; Z3
c	Correct answers	196	7	9
s	safe problems	144	3	7
u	unsafe problems	52	4	2
to	Timeouts	124	117	108
n	Total problems	320	124	117
t_{VCG}	VCG time	40.65	20.48	4.57
t_{NLR}	NLR time	–	58.39	9.53
t_{cFAR}	cFAR time	–	–	304.84
st	Z3 solving time	2704.95	988.15	649.56
tt	Total time	2745.60	1067.02	968.50
at	Average time	14.01	152.43	107.61

Table 1: Verification results obtained by using Z3 on the output generated by applying VCG and the auxiliary transformations NLR and cFAR. The timeout limit time is five minutes. Times are in seconds.

Conclusions

- Removing unnecessary variables may help Horn solvers
- Future work
  - Apply to VCs generated by other tools
  - Experiment with different solvers

- Benchmarks, VCs and tool at
  http://map.uniroma2.it/vcgen/