Semantics-based generation of verification conditions by program specialization

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

PPDP'15 – Siena

Talk Outline

- Partial Correctness properties and Verification Conditions (VCs)
- Constraint Logic Programming as a metalanguage for representing
  - the imperative program
  - the semantics of the imperative language
  - the property to be verified
- CLP program specialization for generating VCs
  - using unfold / fold transformation rules
- Experimental evaluation

Partial Correctness and VCs

Given the partial correctness property (Hoare triple):

$\{n \geq 1\}$ x=0; y=0; **while** (x<n) {x=x+1; y=y+2} $\{y > x\}$

Verification Conditions: formulas whose satisfiability implies correctness

Verification conditions (VCs) as Horn clauses with constraints (CLP)

$n \geq 1 \wedge x=0 \wedge y=0 \rightarrow P(x,y,n)$ (Initialization)

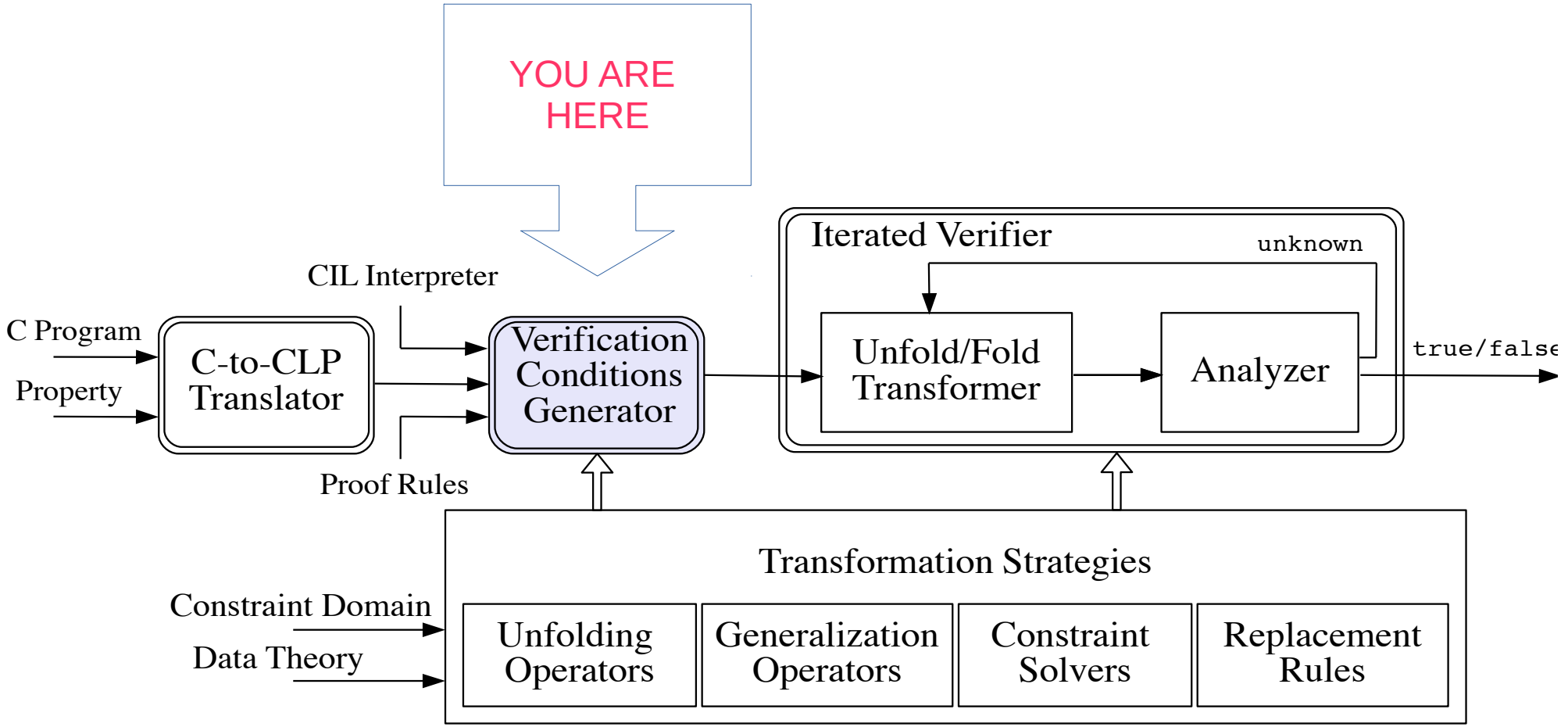
$P(x,y,n) \wedge x<n \rightarrow P(x+1,y+2,n)$ (Loop body execution)

$P(x,y,n) \wedge x \geq n \wedge y \leq x \rightarrow$ false (Loop exit)

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

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

VeriMAP architecture



De Angelis et al.: VeriMAP: A Tool for Verifying Programs through Transformations. (tool paper) TACAS 2014
De Angelis et al.: Program Verification via Iterated Specialization. Science of Computer Programming. 2014.

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 requires **rewriting** the VCGEN algorithm

Semantics-based approach

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

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

Our semantics-based approach

- uses CLP program specialization
- Correctness of VC generation follows from correctness of unfold/fold transformation rules
- 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 *gcd*

```
int x, y;

int sub(int a, int b) {
      int r = a-b;                l1
      return r;                   l2
}
void main() {
      while (x!=y)                l3
            if (x>y)              l4
                  x = sub(x,y);   l5
            else
                  y = sub(y,x);   l7
}
```

CLP encoding of *gcd*

```
fun(sub,[a,b],[r],1).
at(1,asgn(r,minus(a,b))).    % l1
at(2,return(r)).             % l2

fun(main,[],[],3).
at(3,ite(neq(x,y),4,h)).
at(4,ite(gt(x,y),5,7)).
at(5,asgn(x,call(sub,[x,y]))).
at(6,goto(3)).
at(7,asgn(y,call(sub,[y,x]))).
at(8,goto(3)).
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

assignment 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 the Operational Semantics

function call x=f(e1,...,en); "return" case

```
tr(cf(cmd(L,asgn(X,call(F,Es))), (D,S)),          source configuration
   cf(cmd(L2,C2),                 (D2,S2))) :-    target configuration
     eval_list(Es,D,S,Vs),          evaluate function parameters
     build_funenv(F,Vs,FEnv),       build function environment
     firstlab(F,FL),  at(FL,C),     first label and command function def
     reach( cf(cmd(FL,C),          (D,FEnv)),     function execution
            cf(cmd(LR,return(E)),(D1,S1))),      return
     eval(E,(D1,S1),V),              evaluate returned expression
     update((D1,S),X,V,(D2,S2)),   update caller environment
     nextlab(L,L2), at(L2,C2).      next label and command
```

Multi-Step Operational Semantics

Encoding Partial (In)Correctness

Partial correctness property P

$$\{x \geq 1 \wedge y \geq 1\} \; gcd \; \{x \geq 0\}$$

CLP encoding of (in)correctness. CLP program I

```
incorrect :- initConf(Cf), reach(Cf,Cf1), errorConf(Cf1).
                                  …
   initConf(cf(C, [(x,X),(y,Y)])) :- at(3,C), X>=1, Y>=1.
errorConf(cf(C, [(x,X),(y,Y)])) :- at(h,C), X=< -1.
```

Thm. Correctness of CLP Encoding

property P 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”.

Unfold/Fold program specialization

incorrect :- initConf(C),reach(C,C1),errorConf(C1).

- **UNFOLDING** (replace initConf(C) with the body of its definition)

  incorrect :- X>=1,Y>=1, reach(cf(cmd(3,ite(neq(x,y)),4,h), [(x,X),(y,Y)],[]), C1),
  errorConf(C1).

- **UNFOLDING** (wrt errorConf(C1))

  incorrect :- X>=1,Y>=1, X1=<-1, reach(cf(cmd(3,ite(neq(x,y)),4,h),[(x,X),(y,Y)],[]),
  cf(cmd(h,halt),[(x,X1),(y,Y1)],[])).

- **DEFINITION-INTRODUCTION** (with constraint generalization)

  new3(X,Y, X1,Y1) :- reach(cf(cmd(3,ite(neq(x,y)),4,h),[(x,X),(y,Y)],[]),
  cf(cmd(h,halt),[(x,X1),(y,Y1)],[])).

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

  incorrect :- X>=1,Y>=1,X1=<-1, new3(X,Y, X1,Y1).

VCG strategy

Input: program I (incorrect, at, tr, reach, …)
Output: VCs (new predicates)

VCs := ∅;
Defs := {incorrect :- initConf(C),reach(C,C1),errorConf(C1)};

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

Thm. Termination and correctness of the VCG strategy

(i) the VCG strategy terminates
(ii) incorrect ∈ M(I) iff incorrect ∈ M(VCs)

VCs Multi-Step Semantics

```
incorrect :- X>=1,Y>=1,X1=<-1, new3(X,Y, X1,Y1).
new3(X,Y, X1,Y1):- X+1=<Y, new4(X,Y, X1,Y1).        % loop execution
new3(X,Y, X1,Y1):- X>=Y+1, new4(X,Y, X1,Y1).        % loop execution
new3(X,Y, X,Y):- X=Y.                               % loop exit
new4(X,Y, X3,Y3):- X>=Y+1, A=X, B=Y, X2=R1,         % then branch
                   new6(X,Y,A,B,R,  X1,Y1,A1,B1,R1),
                   new3(X2,Y1, X3,Y3).
new4(X,Y, X3,Y3):- X=<Y,     A=Y, B=X, Y2=R1,       % else branch
                    new6(X,Y,A,B,R,  X1,Y1,A1,B1,R1),
                   new3(X1,Y2, X3,Y3).
new6(X,Y,A,B,R,  X,Y,A,B,R1) :- R1=A-B.             %   sub  function call
```

VCs generated by using the multi-step semantics

- Non linear recursive: multiple atoms in the body (e.g. new6, new3)
- Predicate arity is even (variables for source and target configurations)

Small-Step Semantics

We need to keep a stack of activation frames

Function call: push an element on top of the stack

```
tr(cf(cmd(L,asgn(X,call(F,Es))),D,T),
   cf(cmd(FL,C),                    D,[frame(L1,X,FEnv)|T])) :-
            nextlab(L,L1),
            loc_env(T,S),  eval_list(Es,D,S,Vs),
            build_funenv(F,Vs,FEnv),
             firstlab(F,FL), at(FL,C).
```

L1 label where to jump after returning
X stores the value returned by the function call
FEnv local environment used during the execution of the function call

Function return: pop an element from the stack

```
tr(cf(cmd(L,return(E)),D,[frame(L1,X,S) |T]),
   cf(cmd(L1,C),         D1,T1))                    :-
            eval(E,D,S,V),
            update((D,T),X,V,(D1,T1)),
             at(L1,C).
```

Small-Step Semantics

Encoding correctness when using the Small-Step semantics

```
incorrect :- initConf(C), reach(C).
reach(C) :- tr(C,C1), reach(C1).
reach(C) :- errorConf(C).
```

VCs generated by using the Small-Step semantics

```
incorrect :- X>=1, Y>=1, new3(X,Y).
new3(X,Y) :- X=<-1, Y=X.
new3(X,Y) :- X+1=<Y, new4(X,Y).
new3(X,Y) :- X>=1+Y, new4(X,Y).
new4(X,Y) :- X>=Y+1, new6(X,Y).
new4(X,Y) :- X=<Y, new7(X,Y).
new6(X,Y) :- A=X, B=Y, new11(X,Y,A,B,R).
new7(X,Y) :- A=Y, B=X, new8(X,Y,A,B,R).
new8(X,Y,A,B,R) :- R1=A-B, new9(X,Y,A,B,R1).
new9(X,Y,A,B,R) :- Y1=R, new3(X,Y1).
new11(X,Y,A,B,R) :- R1=A-B, new12(X,Y,A,B,R1).
new12(X,Y,A,B,R) :- X1=R, new3(X1,Y).
```

- Linear recursive (at most one atom in the body)
- More predicates and clauses than in Multi-Step semantics VCs
  Multiple predicates for the calls to the sub function (e.g. new11 and new8)
- Half the variables w.r.t. MS semantics VCs

Semantics variations

- Side-effect free functions
  - new6(X,Y,A,B,R, ~~X1,Y1,~~ A,B,R1) :- R1=A-B. % sub
- Undefined functions and assertions
- Stack traces in case of aborted execution
- Output commands
- Mapping the VCs to source code
- Tuning the unfolding strategy

```
incorrect :- X>=1,Y>=1,X1=<-1, new3(X,Y ,X1,Y1).
new3(X,Y, X2,Y2) :-  X1=X-Y, X>=Y+1, new3(X1,Y, X2,Y2).
new3(X,Y, X2,Y2) :-  Y1=Y-X, X+1=<Y, new3(X,Y1, X2,Y2).
new3(X,Y, X,Y) :- X=Y.
```

Experimental evaluation

- 320 verification problems written in the C language
  - from TACAS SV-COMP, other public benchmarks
- Performance improvements wrt previous VCG strategy
  - More efficient constraint satisfiability checker (rows 2,4)
  - Replace sequences of unfolding steps with Prolog calls (3,4)

		n	t_{VCG}	t'_{VCG}
Small-step	1. SS_o^p	216	159.45	159.45
	2. SS_o^s	320	1235.96	35.58
	3. SS_f^p	317	4254.71	34.71
	4. SS_f^s	320	218.57	11.25
Multi-step	5. MS	318	364.43	7.70

Table 3. Times (in seconds) taken for the VC generation using different language semantics and settings. The time limit is five minutes. n is the number of programs out of 320, for which the VCs were generated.

Experimental evaluation

- Checking the satisfiability of the VCs
  - QARMC, Z3 (PDR), MathSAT (IC3), HSF+QARMC

		Small-step(SS_f^s)			Multi-step (MS)			HSF + QARMC
		QARMC	Z3	MSATIC3	QARMC	Z3	MSATIC3	
c	Correct answers	221	209	212	212	196	178	189
s	safe problems	164	150	160	161	143	149	158
u	unsafe problems	57	59	52	51	53	29	31
i	Incorrect answers	5	0	2	3	0	0	12
f	false alarms	3	0	0	1	0	0	3
m	missed bugs	2	0	2	2	0	0	9
to	Timed-out problems	94	111	106	103	122	140	119
n	Total problems	320	320	320	318	318	318	320
t_{VCG}	VCG time	218.57	218.57	218.57	364.43	364.43	364.43	N/A
st	Solving time	3423.75	3446.84	3008.81	2924.62	2618.21	1616.81	N/A
tt	Total time	3642.32	3665.41	3227.38	3289.05	2982.64	1981.24	631.11
at	Average Time	16.12	17.54	15.08	15.30	15.30	11.13	3.14

Table 4. Verification results using QARMC, Z3, MSATIC3, and HSF+QARMC. The time limit is five minutes. Times are in seconds.

Demo

Hope it works... :-)

Conclusions

- Semantics-based VC generation
  - Semantics of the programming language as a parameter
  - Flexible and quite efficient
  - Viable from a practical point of view
- Future work
  - More languages
  - More properties
- Benchmarks, VCs and tool at http://map.uniroma2.it/vcgen/

The end

Thank you!

```
1.  tr(cf(cmd(L,asgn(X,expr(E))), (D,S)), cf(cmd(L1,C), (D1,S1))) :-
        eval(E,(D,S),V), update((D,S),X,V,(D1,S1)), nextlab(L,L1), at(L1,C).
2a. tr(cf(cmd(L,asgn(X,call(F,Es))),(D,S)), cf(cmd(LA,abort),(bot,D1,S))) :-
        eval_list(Es,D,S,Vs), build_funenv(F,Vs,FEnv), firstlab(F,FL), at(FL,C),
        reach(cf(cmd(FL,C),(D,FEnv)), cf(cmd(LA,abort),(bot,D1,S1))).
2r. tr(cf(cmd(L,asgn(X,call(F,Es))),(D,S)), cf(cmd(L2,C2),(D2,S2))) :-
        eval_list(Es,D,S,Vs), build_funenv(F,Vs,FEnv), firstlab(F,FL), at(FL,C),
        reach(cf(cmd(FL,C),(D,FEnv)), cf(cmd(LR,return(E)),(D1,S1))),
        eval(E,(D1,S1),V), update((D1,S),X,V,(D2,S2)), nextlab(L,L2), at(L2,C2).
3.  tr(cf(cmd(L,abort),(D,S)), cf(cmd(L,abort),(bot,D,S))).
4t. tr(cf(cmd(L,ite(E,L1,L2)),(D,S)), cf(cmd(L1,C),(D,S))) :- beval(E,(D,S)), at(L1,C).
4f. tr(cf(cmd(L,ite(E,L1,L2)),(D,S)), cf(cmd(L2,C),(D,S))) :- beval(not(E),(D,S)), at(L2,C).
5.  tr(cf(cmd(L,goto(L1)),(D,S)), cf(cmd(L1,C),(D,S))) :- at(L1,C).
6.  update((D,S),X,V,(D1,S)) :- global(X), update_glocal(D,X,V,D1).
7.  update((D,S),X,V,(D,S1)) :- local(X), update_glocal(S,X,V,S1).
8.  reach(C,C).
9.  reach(C,C2) :- tr(C,C1), reach(C1,C2).
```

Table 2. The CLP interpreter for the multi-step operational semantics *MS*: the clauses for `tr` and `reach`.