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):

\[
\{n \geq 1\} \quad \text{x=0; y=0;} \quad \{y > x\} \\
\text{while} \ (x<n) \ \{x=x+1; \ y=y+2\}
\]

Verification Conditions: formulas whose satisfiability implies correctness

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

\[
\begin{align*}
n \geq 1 & \land x=0 \land y=0 \rightarrow P(x,y,n) & \quad \text{(Initialization)} \\
P(x,y,n) \land x<n & \rightarrow P(x+1,y+2,n) & \quad \text{(Loop body execution)} \\
P(x,y,n) \land x\geq n \land y \leq x & \rightarrow \text{false} & \quad \text{(Loop exit)}
\end{align*}
\]

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

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

C-to-CLP Translator

Verification Conditions Generator

Unfold/Fold Transformer

Analyzer

Iterated Verifier

Transformation Strategies

Unfolding Operators

Generalization Operators

Constraint Solvers

Replacement Rules

YOU ARE HERE

De Angelis et al.: VeriMAP: A Tool for Verifying Programs through Transformations. (tool paper) TACAS 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: \texttt{at(Label, Cmd)}

\begin{itemize}
\item \textbf{Program \textit{gcd}}
\begin{verbatim}
int x, y;

int \textbf{sub}(int a, int b) {
  int r = a-b;
  return r;
}

\textbf{void} main() {
  \textbf{while} (x!=y) {
    \textbf{if} (x>y) {
      x = sub(x,y);
    } \textbf{else} {
      y = sub(y,x);
    }
  }
}
\end{verbatim}
\end{itemize}

\begin{itemize}
\item \textbf{CLP encoding of \textit{gcd}}
\begin{verbatim}
fun\(\text{sub},[a,b],[r],1)\).
\texttt{at}(1,\texttt{asgn}(r,\texttt{minus}(a,b))). \% l1
\texttt{at}(2,\texttt{return}(r)). \% l2

fun\(\text{main},[],[],3)\).
\texttt{at}(3,\texttt{ite}(\texttt{neq}(x,y),4,h)).
\texttt{at}(4,\texttt{ite}(\texttt{gt}(x,y),5,7)).
\texttt{at}(5,\texttt{asgn}(x,\texttt{call}(\texttt{sub},[x,y]))).
\texttt{at}(6,\texttt{goto}(3)).
\texttt{at}(7,\texttt{asgn}(y,\texttt{call}(\texttt{sub},[y,x]))).
\texttt{at}(8,\texttt{goto}(3)).
\texttt{at}(h,\texttt{halt}).
\end{verbatim}
\end{itemize}
Encoding the Operational Semantics

Configurations: \( \text{cf}(\text{LC}, \text{Env}) \)  program execution state
- \text{LC} labeled command: a term of the form \( \text{cmd}(L,C) \)
  - \( L \) label, \( C \) command
- \text{Env} environment: a pair \( (D,S) \)
  - \( D \) global environment, \( S \) local environment
  - Environments as lists of pairs \([x,X),(y,Y),(z,Z)]\)

Operational semantics: transition relation \( \text{tr} \) between configurations
\[
\text{tr}( \text{cf}(\text{LC1},E1), \text{cf}(\text{LC2},E2))
\]

Multiple steps reachability (reflexive, transitive closure of \( \text{tr} \))
\[
\text{reach}(C,C).
\text{reach}(C,C2) :- \text{tr}(C,C1), \text{reach}(C1,C2).
\]
Encoding the Operational Semantics

**assignment**  \( x = e; \)

\[
\text{tr( cf(cmd(L, asgn(X,expr(E))), (D,S)), cf(cmd(L1,C), (D1,S1))) :- source configuration target configuration} \\
\text{eval(E,(D,S),V), evaluate expression} \\
\text{update((D,S),X,V,(D1,S1)), update environment} \\
\text{nextlab(L,L1), next label} \\
\text{at(L1,C). next command}
\]
function call  \( x = f(e_1, \ldots, e_n); \)  

\[
\text{tr}(\text{cf}(\text{cmd}(L, \text{asgn}(X, \text{call}(F, Es))))), (D, S)), \\
\text{cf}(\text{cmd}(L_2, C_2)), (D_2, S_2)) :- \\
\text{eval_list}(Es, D, S, Vs), \\
\text{build_funenv}(F, Vs, FEnv), \\
\text{firstlab}(F, FL), \text{ at}(FL, C), \\
\text{reach}( \text{cf}(\text{cmd}(FL, C)), (D, FEnv)), \\
\text{cf}(\text{cmd}(LR, \text{return}(E)), (D_1, S_1)), \\
\text{eval}(E, (D_1, S_1), V), \\
\text{update}((D_1, S), X, V, (D_2, S_2)), \\
\text{nextlab}(L, L_2), \text{ at}(L_2, C_2).
\]

Multi-Step Operational Semantics
Encoding Partial (In)Correctness

Partial correctness property $P$

$$\{ x \geq 1 \land y \geq 1 \} \quad \text{gcd} \quad \{ x \geq 0 \}$$

CLP encoding of (in)correctness.

\[
\text{incorrect} :- \text{initConf}(Cf), \text{reach}(Cf,Cf1), \text{errorConf}(Cf1).
\]

\[
\text{initConf}(cf(C, [(x,X),(y,Y)])) :- \text{at}(3,C), X\geq1, Y\geq1.
\]

\[
\text{errorConf}(cf(C, [(x,X),(y,Y)])) :- \text{at}(h,C), X\leq -1.
\]

**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 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

\[
\begin{align*}
& \text{tr(cf(cmd(L, asgn(X, call(F, Es))), D, T),} \\
& \quad \text{cf(cmd(FL, C),} \quad D, [\text{frame}(L1, X, FEnv) | T]) \quad :- \\
& \quad \text{nextlab(L, L1),} \\
& \quad \text{loc_env(T, S), eval_list(Es, D, S, Vs),} \\
& \quad \text{build_funenv(F, Vs, FEnv),} \\
& \quad \text{firstlab(F, FL), at(FL, C).}
\end{align*}
\]

- **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

\[
\begin{align*}
& \text{tr(cf(cmd(L, return(E)), D, [frame(L1, X, S) | T]),} \\
& \quad \text{cf(cmd(L1, C),} \quad D1, T1)) \quad :- \\
& \quad \text{eval(E, D, S, V),} \\
& \quad \text{update((D, T), X, V, (D1, T1)),} \\
& \quad \text{at(L1, C).}
\end{align*}
\]
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 \texttt{sub} function (e.g. new11 and new8)
- Half the variables \emph{w.r.t.} MS semantics VCs
Semantics variations

- Side-effect free functions
  - new6(X,Y,A,B,R, X4,Y4, 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)

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>$t_{VCG}$</th>
<th>$t'_{VCG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small-step</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. $SS^p_o$</td>
<td>216</td>
<td>159.45</td>
<td>159.45</td>
</tr>
<tr>
<td>2. $SS^s_o$</td>
<td>320</td>
<td>1235.96</td>
<td>35.58</td>
</tr>
<tr>
<td>3. $SS^p_f$</td>
<td>317</td>
<td>4254.71</td>
<td>34.71</td>
</tr>
<tr>
<td>4. $SS^s_f$</td>
<td>320</td>
<td>218.57</td>
<td>11.25</td>
</tr>
<tr>
<td>Multi-step</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5. $MS$</td>
<td>318</td>
<td>364.43</td>
<td>7.70</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Small-step ($SS_f^+$)</th>
<th>Multi-step ($MS$)</th>
<th>HSF + QARMC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QARMC</td>
<td>Z3</td>
<td>MSATIC3</td>
</tr>
<tr>
<td>$c$</td>
<td>Correct answers</td>
<td>221</td>
<td>209</td>
</tr>
<tr>
<td>$s$</td>
<td>Safe problems</td>
<td>164</td>
<td>150</td>
</tr>
<tr>
<td>$u$</td>
<td>Unsafe problems</td>
<td>57</td>
<td>59</td>
</tr>
<tr>
<td>$i$</td>
<td>Incorrect answers</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$f$</td>
<td>False alarms</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$m$</td>
<td>Missed bugs</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$t_1$</td>
<td>Timed-out problems</td>
<td>94</td>
<td>111</td>
</tr>
<tr>
<td>$n$</td>
<td>Total problems</td>
<td>320</td>
<td>320</td>
</tr>
<tr>
<td>$tv_C$</td>
<td>VCG time</td>
<td>218.57</td>
<td>218.57</td>
</tr>
<tr>
<td>$st$</td>
<td>Solving time</td>
<td>3423.75</td>
<td>3446.84</td>
</tr>
<tr>
<td>$tt$</td>
<td>Total time</td>
<td>3642.32</td>
<td>3665.41</td>
</tr>
<tr>
<td>$at$</td>
<td>Average Time</td>
<td>16.12</td>
<td>17.54</td>
</tr>
</tbody>
</table>

Table 4. Verification results using QARMC, Z3, MSATIC3, and HSF+QARMC. The time limit is five minutes. Times are in seconds.
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!
Table 2. The CLP interpreter for the multi-step operational semantics $MS$: the clauses for \texttt{tr} and \texttt{reach}.