Generating Verification Conditions from Annotated Programs

Deepak D’Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

13 March 2014
Outline

1. Overview of Verification
2. Hoare logic
3. How VCC generates VC’s
Basic Idea of verification technology

- Given a program $P$ with assert, assume, invariant annotations.
- $P$ satisfies annotations if no execution of it “goes wrong”.
  - An execution goes wrong if it violates an assert and passes all assume’s till then.
- Translate it to an acyclic program with goto’s $P'$.
- $P'$ satisfies property that if $P'$ does not go wrong then neither will $P$.
- Generate Verification Conditions (VC’s) $\varphi_{P'}$ from $P'$, such that $\varphi_{P'}$ is valid iff $P'$ does not go wrong.
- Check validity of $\varphi_{P'}$ using an SMT solver like Z3.
Translating $P$ to acyclic program $P'$

```plaintext
int min(int a, int b)
  _(requires \true)
  _(ensures \result <= a && \result <= b) {
    if (a <= b)
      return a;
    else
      return b;
  }
```

```plaintext
int min(int a, int b)
  assume \true
  int \result;
  goto iftrue, iffalse;
  assume a <= b
  \result = a;
  goto endif;
  assume a > b;
  \result = b;
  goto endif;
  assert \result <= a && \result <= b
```

```plaintext
iftrue: assume a <= b
  \result = a;
  goto endif;

iffalse: assume a > b;
  \result = b;
  goto endif;
```

```plaintext
endif: assert \result <= a && \result <= b
```
Translating $P$ to acyclic program $P'$: function calls

int main() {
    int x, y, z;
    z = min(x, y);
    _{assert z <= x}
    return 0;
}

int main() {
    assume \true
    int \result, x, y, z;
    int res;
    assert \true
    assume res <= x && res
    z = res;
    assert z <= x
    \result = 0
    assert \true
}
Translating $P$ to acyclic program $P'$: loops with invariants

void div(unsigned x, unsigned d, unsigned *q, unsigned *r) {
  unsigned lq, lr;
  lq = 0;
  lr = x;
  while(lr >= d) {
    (invariant x == d * lq + lr) {
      lq++;
      lr = lr - d;
    }
    *q = lq;
    *r = lr;
    return;
  }
}

unsigned div(unsigned x, d, *q, *r) {
  assume d > 0 && q != r
  int \result, lq, lr;
  lq = 0; lr = x;
  assert x == lq * d + rq
  unsigned fresh_lq, fresh_lr;
  lq = fresh_lq; lr = fresh_lr;
  assume x == lq * d + lr
  if !(lr >= d) goto loopexit
  lq++;
  lr = lr - d;
  assert x == lq * d + rq
  assume \false
  *q = lq; *r = lr;
  loopexit: *q = lq; *r = lr;
  assert x == (*q) * d + *r && *r < d
Rules for Weakest Preconditions

- Let $WP(L, Q)$, where $L$ is a statement label in program $P$ and $Q$ is a post-condition on the state of $P$, denote the set of states $s$ such that if we execute $P$ starting at label $L$ in state $s$, the execution never goes wrong, and if it terminates it does so in a state satisfying $Q$.

- Let $M$ be the label of the statement following $L$. Below “goto $N, O$” means non-deterministically branch to label $N$ or label $O$. Then
  - $WP(L: \text{assume } A, Q) = A \implies WP(M, Q)$.
  - $WP(L: \text{assert } A, Q) = A \land WP(M, Q)$.
  - $WP(L: x := e, Q) = WP(M, Q)[e/x]$.
  - $WP(L: \text{goto } N, O, Q) = WP(N, Q) \land WP(O, Q)$. 
Label each program statement “L: ...” in $P'$ by $WP(L, true)$:

- Begin from leaf nodes and proceed upwards to label a node if its control successors have been labelled.

Output $\Box \implies \varphi_0$ as the verification condition for $P'$, where $\varphi_0$ is the $WP$ at the start node of $P'$.

Clearly, $P'$ has no execution that goes wrong iff $\varphi_{P'}$ is valid (in other words it negation is unsatisfiable).
Generating VC’s from an acyclic $P'$: min example

```c
int min(int a, int b)
{
    [a <= b ==> (a <= a && a <= b)]
    assume \true
    \&\& [a > b ==> (b <= a && b <= b)]
    [a <= b ==> (a <= a && a <= b)]
    int \result;
    \&\& [a > b ==> (b <= a && b <= b)]
    goto iftrue, iffalse;
    [a <= b ==> (a <= a && a <= b)]
    \&\& [a > b ==> (b <= a && b <= b)]
    a <= b ==> (a <= a && a <= b)
    iftrue: assume a <= b
    a <= a && a <= b
    \result = a;
    \result <= a && \result <= b
    goto endif;
    a > b ==> (b <= a && b <= b)
    iffalse: assume a > b;
    b <= a && b <= b
    \result = b;
    \result <= a && \result <= b
    goto endif;
    \result <= a && \result <= b
    endif: assert \result <= a && \result <= b
}
```
Generating VC’s from an acyclic \( P' \): \textit{min} example

```c
int min(int a, int b)
    [a <= b ==> (a <= a && a <= b)]
    assume \textbf{true}
    [a <= b ==> (a <= a && a <= b)]
    int \textbf{result};
    [a <= b ==> (a <= a && a <= b)]
    \&\& [a > b ==> (b <= a && b <= b)]
    goto iftrue, iffalse;
    [a <= b ==> (a <= a && a <= b)]
    \&\& [a > b ==> (b <= a && b <= b)]
    \textbf{result} = a;
    [a <= b ==> (a <= a && a <= b)]
    a <= a && a <= b
    \textbf{result} <= a && \textbf{result} <= b
    goto endif;
    a > b ==> (b <= a && b <= b)
    a <= b ==> (a <= a && a <= b)
    \textbf{result} = b;
    b <= a && b <= b
    \textbf{result} <= a && \textbf{result} <= b
    goto endif;
    \textbf{result} <= a && \textbf{result} <= b
endif: assert \textbf{result} <= a && \textbf{result} <= b
```

Final formula \( \varphi_{\text{min}} \) generated (\( A \) is axioms known, like \textit{int} \ a):

\[
\forall \ \Rightarrow [a \leq b \Rightarrow (a \leq a \land a \leq b)] \land [a > b \Rightarrow (b \leq a \land b \leq b)]
\]