Interactive Software Verification

Spring Term 2013

Holger Gast
gast@in.tum.de

28.05.2013
## Recap: The Big Picture

<table>
<thead>
<tr>
<th>Correctness of Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve verification conditions</td>
</tr>
<tr>
<td>Arguments about application domain</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hoare Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verification rules for language constructs</td>
</tr>
<tr>
<td>Generator for verification conditions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define meaning of programs</td>
</tr>
<tr>
<td>Describe behaviour of programs</td>
</tr>
</tbody>
</table>
Recap: The Plan

C/Java/... \(\xrightarrow{\text{translate}}\) Simpl \(\xrightarrow{\text{pass to}}\) VCG \(\xrightarrow{\text{generate}}\) Verification Conditions

Semantics

\(\xrightarrow{\text{based on}}\) take into account

outside Isabelle
not formal, not checked
Recap: The Semantics of Simpl

- Distinguish between different execution modes/outcomes
  
  \[\text{datatype} \ xstate = \text{Normal state} \mid \text{Abrupt state} \mid \text{Fault} \mid \text{Stuck}\]

- Execution as inductively defined predicate

  \[\text{inductive} \ \text{exec} :: "[body, com, xstate, xstate] \Rightarrow \text{bool}" \quad ("\vdash \langle-,-\rangle \Rightarrow -" \cdots)\]

- Example: sequence

  \[
  \begin{array}{c}
  [\Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow s';
  
  \Gamma \vdash \langle c_2, s' \rangle \Rightarrow t
  
  \] \implies \Gamma \vdash \langle \text{Seq } c_1 c_2, \text{Normal } s \rangle \Rightarrow t
  
  \end{array}
  \]

- Basic: shallow embedding of state updates

  \[\Gamma \vdash \langle \text{Basic } f, \text{Normal } s \rangle \Rightarrow \text{Normal } (fs)\]

- Conditional

  \[
  \begin{array}{c}
  [s \in b; \Gamma \vdash \langle c_1, \text{Normal } s \rangle \Rightarrow t] \implies \Gamma \vdash \langle \text{Cond } b \ c_1 c_2, \text{Normal } s \rangle \Rightarrow t
  
  [s \notin b; \Gamma \vdash \langle c_2, \text{Normal } s \rangle \Rightarrow t] \implies \Gamma \vdash \langle \text{Cond } b \ c_1 c_2, \text{Normal } s \rangle \Rightarrow t
  
  \end{array}
  \]
Today

- Today: the VCG of Simpl
  - Definition of “correctness of programs”
  - Relationship between correctness and semantics
  - A Hoare logic for Simpl

- Based on [4]

- Material taken from [5] & simplified
Constructing the VCG
Idea of “correctness”

- Goal: Prove statements about a program’s behaviour

- $P$ and $Q$ are predicates on states

- Reasoning pattern
  - Check that $P$ holds for $s$ & start the program
  - Conclude that $Q$ holds for $s'$
  - Interpret content of $s'$ as desired value described by $Q$
Formulating Correctness

- **Assertion**: statement about states $\leftrightarrow$ a set of states
  
  **type-synonym** $\text{'s assn} = \text{" set"}$

- Construct (Hoare-) triple with **precondition** & **postcondition**
  
  $\{ P \} \text{sm} \{ Q \}$

- For **Simpl** need quadruple:
  
  - xstate enables different types of outcomes
  - Stuck / Fault: internal execution error
  - Normal / Abrupt: termination of program
  
  $\Rightarrow$ Alternative postcondition for abrupt termination
A Simple Example in Simpl

**lemma** (in ex) simple-assign:
"Γ ⊢ { `x > 0 ∧ `y > 0 } `z := `x + `y { `z > 0 }"

- **Notes**
  - Γ is formally necessary context of procedure definition
  - Braces are `<lbrace>` and `<rbrace>`; in jEdit they look like bold curly braces; type lbrace/rbrace and autocomplete.
  - apply vcg yields proof obligation
    \( \forall x y. [0 < x; 0 < y] \implies 0 < x + y \)
  - If we prove this (by simp), the program is correct.
A Few Remarks on Notation

• Already seen: \( s \in P \) instead of predicate \( P s \)

• The image of a set under a function (write as backtick)
  \[
  f' A = \{ y. \exists x \in A. y = f x \}
  \]

⇒ Can help hiding an existential

• Application: “the normal states satisfying \( P \)”
  \[
  (s \in \text{Normal } ' P) \iff (\exists n. s = \text{Normal } n \land n \in P)
  \]

• Take the desired states \( P \)
• Lift them into \textit{xstate} as Normal states
• Check that \( s \) is in that set
Defining Partial Correctness

- Definition correctness
  \[ \Gamma \models P_c Q, A \equiv \forall s.t. s \in \text{Normal} \implies P \rightarrow \Gamma \vdash \langle c, s \rangle \Rightarrow t \rightarrow t \in \text{Normal} \cup \text{Abrupt} \]

- Partial correctness
  - If execution starts in a Normal state \( s \)
  - In which the precondition \( P \) holds
  - And if execution terminates
  - Then no error has occurred (Fault/Stuck)
  - And
    - Postcondition \( Q \) holds in case of normal termination
    - Postcondition \( A \) holds in case of abrupt termination
What is “partial” here?

- “Partial” as in “partial function”
  - If execution does not terminate, it yields not result
  - We cannot make a statement about the result

⇒ Consider statements as partial functions to result state

- We have no assertions about
  - Whether the program terminates
  - The program’s behaviour if non-terminating
  - Intermediate states of the execution

⇒ Partial correctness may not be “safe enough” (e.g. for embedded controllers)
Hoare Logic

• Correctness is a semantic notion, based on execution & states
• Goal: reason about the source code, not the semantics
• Idea: have independent rules for this reasoning
• Idea by Hoare [3], Floyd [2], Dijkstra [1]

• In Isabelle: another inductively-defined relation
  \( \Gamma, \Theta \vdash P \circ Q, A \)
  • Note: Different symbol (from logic: provability vs. validity)
  • \( P, Q, A \) have same intended meaning
  • Defined along structure of \( c \Rightarrow \) proof by rule application
Soundness of the Hoare Logic

- Essential: connect Hoare Logic to correctness
  - Reason within Hoare logic
  - Derive relation pre-/postcondition
  - Conclude that this relation holds for the actual execution
- The Hoare logic is sound if this reasoning is justified

\[ \Gamma, \Theta \vdash P \circ Q, A \quad \Rightarrow \quad \Gamma, \Theta \models P \circ Q, A \]

- If we can prove that \( c \) obeys the pre-/postcondition relation (according to the definition of the Hoare logic)
- Then its execution actually does obey this relation (according to the definition of correctness)
- Prove soundness in Isabelle to be really sure
Structural Rules

• Hoare Rules: (a) for program constructs, (b) about triples

•Assertion $P$ is stronger than $P'$ iff for any state $s$ we have $P_s \implies P'_s$

• Symmetrically: weaker assertions

• Strengthening the pre-condition

$$\begin{align*}
\{ P' \} \ c \ \{ Q \} \quad & P \implies P' \\
\{ P \} \ c \ \{ Q \} \quad & \ \{ P' \} \ c \ \{ Q \}
\end{align*}$$

• Weakening the post-condition

$$\begin{align*}
\{ P \} \ c \ \{ Q' \} \quad & Q' \implies Q \\
\{ P \} \ c \ \{ Q \} \quad & \ \{ P \} \ c \ \{ Q \}
\end{align*}$$

• Note: the Simpl consequence rules subsumes those but is more complex since it has to treat auxiliary variables.
Hoare Rule for Skip

- Skip does nothing
  \[ \Gamma, \Theta \vdash Q \text{Skip} \Omega, A \]

- Check that sensible: reading in semantics
  - If \( Q \) already holds before the execution
  - Then it holds after the execution

- Alternative: backward reading
  - If \( Q \) is to hold after the execution
  - Then it must hold already before
Rule for \textit{Seq}

- Rule for \textit{Seq}
  \[ \begin{array}{c}
  \Gamma, \Theta \vdash P \mathbf{c}_1 R, A; \\
  \Gamma, \Theta \vdash R \mathbf{c}_2 Q, A \\
  \end{array} \] \quad \implies \quad \Gamma, \Theta \vdash P (\text{Seq } \mathbf{c}_1 \mathbf{c}_2) Q, A

- Check: forward reasoning
  - If \( P \) holds before the execution
  - And we can prove that \( R \) holds after \( \mathbf{c}_1 \)
  - And we can prove that \( Q \) holds after \( \mathbf{c}_2 \)
  - Then \( Q \) finally holds

- Backward reasoning
  - If \( Q \) must hold after \( \mathbf{c}_2 \)
  - Then \( R \) must hold before \( \mathbf{c}_2 \)
  - And \( P \) must hold before \( \mathbf{c}_1 \), i.e. at the start
Conditionals

- Rule for if

\[ [\Gamma, \Theta \vdash (P \land b) \ c_1 \ Q, A; \n\Gamma, \Theta \vdash (P \land \neg b) \ c_2 \ Q, A \n] \implies \Gamma, \Theta \vdash P \ (\text{Cond } b \ c_1 \ c_2) \ Q, A \]

- Check by semantic reading
  - If \( P \land b \) holds before execution
  - And after \( c_1 \) we have \( Q \), then finally \( Q \)
  - If \( P \land \neg b \) holds before execution
  - And after \( c_2 \) we have \( Q \), then finally \( Q \)

- Backward reading
  - To have \( Q \) after the conditional, we must either have
    - \( P \land b \) before \( c_1 \) or
    - \( P \land \neg b \) before \( c_2 \)
The Case of Abrupt Termination

- **Skip** justifies any assertion \( A \) (since it never terminates abruptly)
  \[ \Gamma, \Theta \vdash Q \text{Skip} Q, A \]

- **Seq** justifies \( A \) if both \( c_1 \) and \( c_2 \) justify it
  \[
  \begin{align*}
  &\left[ \Gamma, \Theta \vdash P c_1 R, A; \\
  &\quad \Gamma, \Theta \vdash R c_2 Q, A \\
  \right] \implies \Gamma, \Theta \vdash P \left( \text{Seq} c_1 c_2 \right) Q, A
  \end{align*}
  \]

- If \( c_1 \) terminates with an exception, it must guarantee \( A \)
- If \( c_2 \) terminates with an exception, it must guarantee \( A \)

- Same reasoning for \( \text{Cond} \)
State Updates

- Hoare’s classical assignment axiom

\[
\{ Q[e/x] \} x := e \{ Q \}
\]

- Use backward reading
  
  - If \( Q \) is to hold after the assignment, possibly making an assertion about \( x \), then
  
  - \( Q \) must already “hold for \( e \)” before the assignment
  
  - Alternative: \( Q \) must hold if we set \( x \) directly to the value \( e \)
    (rather than waiting for the assignment to happen)

- Note: Rule yields pre-condition for the post-condition

\( \Rightarrow \) Hoare rules are applied backwards to obtain VCs

\( \Rightarrow \) Different formulation weakest preconditions [1]
Examples: Assignment Axiom

• Show \( \{ y > 0 \} \ x := y \ \{ x > 0 \} \)

• By (assign): \( \{ (x > 0)[y/x] \} \ x := y \ \{ x > 0 \} \)

• So: \( \{ y > 0 \} \ x := y \ \{ x > 0 \} \)

• Show \( \{ x > 0 \land y > 0 \} \ z := x + y \ \{ z > 0 \} \)

• Strengthen: \( \{ ?P \} \ z := x + y \ \{ z > 0 \} \land (x > 0 \land y > 0 \implies ?P) \)

• Assign: set \( ?P = (z > 0)[(x + y)/z] = x + y > 0 \)

\( \implies \) Prove \( x > 0 \land y > 0 \implies x + y > 0 \)

\( \implies \) Remove program constructs, prove implications instead
The Rule for Basic

- Simpl uses shallow embedding for state updates
- The rule is very short
  \[ \Gamma, \Theta \vdash \{s. fs \in Q\} (\text{Basic} f) Q, A \]
- Use backward reading
  - If \( Q \) is to hold after the state update via \( f \)
  - Then obviously it must hold in \( f s \)
- Since Basic never terminates abruptly, any \( A \) is justified.
The Rule for While

- Problem: body can be executed many times
- Classical rule with invariant $I$
  
  \[
  \begin{align*}
  P & \implies I \\
  \{ I \land t \} & b \{ I \} \\
  I \land \lnot t & \implies Q \\
  \{ P \} & \text{while } (t) \ b \{ Q \}
  \end{align*}
  \]

- Invariant must hold before execution
- After successful test, the body must preserve the invariant
- In the end, invariant + failed test must imply postcondition
- What is the invariant $I$?
  - Describe intermediate states during iteration
  - Final state is special case of invariant
Simpl’s While Rule

- The raw While statement has no invariant \( \triangleright \) rule is:
  \[
  \Gamma, \Theta \vdash (P \cap b) \ c \ P, A
  \implies \Gamma, \Theta \vdash P (\text{While} \ b \ c) (P \cap -b), A
  \]
- Set \( P = I \)
- Use \( P \land \neg b \) for postcondition
- Introduce "hole" for \( I \) into abstract syntax
  \[
  \text{whileAnno} \ b \ I \ c = \text{While} \ b \ c
  \]
- And derive a rule for the new constant:
  \[
  \begin{align*}
  \left[ &P \subseteq I; \\
  &\Gamma, \Theta \vdash (I \cap b) \ c I, A; \\
  &I \cap -b \subseteq Q \\
  \right] \\
  \implies \Gamma, \Theta \vdash P (\text{whileAnno} \ b \ I \ c) Q, A
  \end{align*}
  \]
Dealing with Abrupt Termination

- **Throw** causes abrupt termination
  \[ \Gamma, \Theta \vdash A \text{Throw} Q, A \]
- If \( A \) must hold after abrupt termination
- Then it must already have held before \text{Throw}
- Any \( Q \) is ok, because \text{Throw} never terminates normally
- **Catch** finishes abrupt execution
  \[
  \left[ \left[ \Gamma, \Theta \vdash P c_1 Q, R; \\
  \Gamma, \Theta \vdash R c_2 Q, A \\
  \right] \Rightarrow \Gamma, \Theta \vdash P \text{Catch} c_1 c_2 Q, A \right]
  \]
- If \( c_1 \) terminates abruptly with state in \( R \),
- Then \( c_2 \) must guarantee normal postcondition \( Q \)
- If \( c_1 \) terminates normally, it must guarantee \( Q \)
- If \( c_2 \) terminates abruptly, it must guarantee \( A \)
The VCG

- Take the input triple \{ P \} sm \{ Q \} from the lemma
- Strengthen precondition to have variable \(?P\) in triple
- Repeatedly apply Hoare rules to fill variable to \(Q'\)

⇒ Have pure implication \(P \implies Q'\)

- Prove this implication to prove the program correct
A Word on Procedures

- Want to separate implementation and specification

⇒ For each procedure \( p \) prove a theorem \( p\_spec \)

- The VCG will look up the theorem by naming conventions

- Example: Multiplication by addition (notation: input/output variables)

```plaintext
procedures
mult (a::nat, b :: nat | s::nat)
where i :: nat in "...

- Prove specification

  lemma (in mult-impl) mult-spec:
  "\∀ A B. Γ ⊢ \{ \`a = A ∧ \`b = B\} \`s ::= PROC mult(\`a, \`b) \{ \`s = A * B\}""

- Used by VCG in verifying calls, e.g.

  procedures square(x :: nat | y :: nat) "\`y ::= CALL mult(\`x, \`x)"
Recursive Procedures

- Recursion: cannot prove correctness spec-lemma beforehand

- Idea: provide their specifications as relation \( \Theta \)

  Cannot use map \( p \mapsto (P, Q, A) \), this prohibits logical variables, see discussion in [4, §3.1.1]

- Definition: partial correctness with context

\[
\Gamma, \Theta \models P \circ Q, A \equiv \\
(\forall (P, p, Q, A) \in \Theta . \Gamma \models P (\text{Call } p) Q, A) \rightarrow \Gamma \models P \circ Q, A
\]

  - Assuming that all specifications
  - . . . are actually obeyed (i.e. all calls are “correct”)
  - Then the given statement must be partially correct

\[ \Rightarrow \text{ In verifying a call, we can assume its specification from } \Theta \]
apply Hoare rules

C/Java/... \textit{translate} Simpl \textit{pass to} VCG \textit{generate} Verification Conditions

\textit{based on} Semantics \textit{take into account}

\textit{outside Isabelle not formal, not checked}

soundness theorem
References


