Interactive Software Verification

Spring Term 2013

Holger Gast

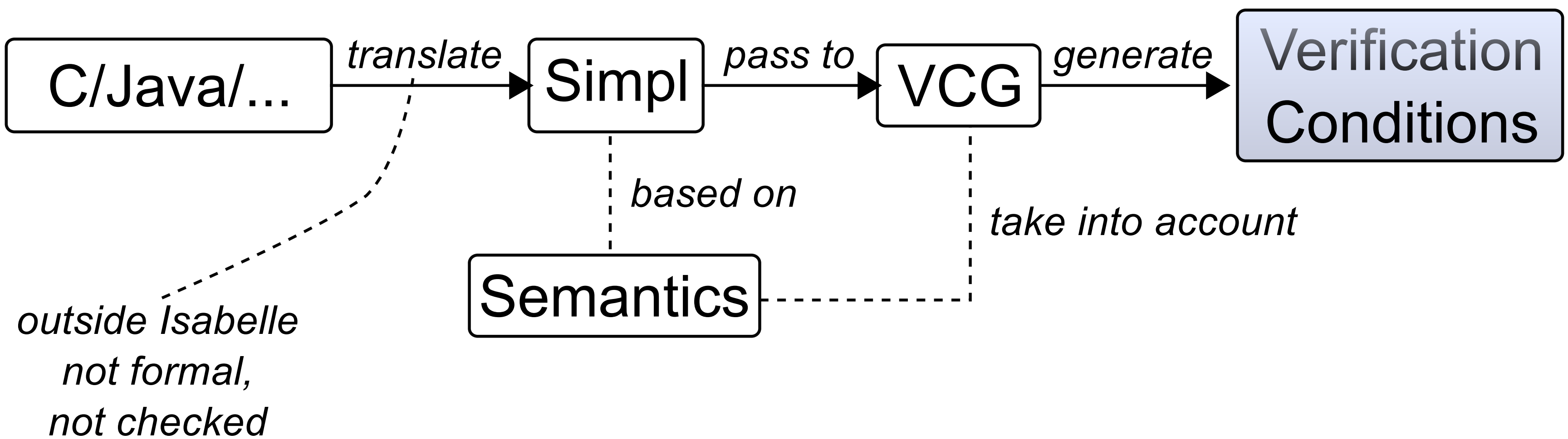
gasth@in.tum.de

28.05.2013

Recap: The Big Picture

Correctness of Programs
Solve verification conditions Arguments about application domain
Hoare Logic
Verification rules for language constructs Generator for verification conditions
Semantics
Define meaning of programs Describe behaviour of programs

Recap: The Plan



Recap: The Semantics of Simpl

- Distinguish between different execution modes/outcomes

  datatype xstate = Normal state | Abrupt state | Fault | Stuck

- Execution as inductively defined predicate

  inductive exec :: "[body,com,xstate,xstate] ⇒ bool" ("-⊢ ⟨-,-⟩ ⇒ -" ⋯)

- Example: sequence

  $\llbracket\ \Gamma \vdash \langle c_1, \text{Normal}\, s \rangle \Rightarrow s';$
  $\Gamma \vdash \langle c_2, s' \rangle \Rightarrow t$
  $\rrbracket \Longrightarrow \Gamma \vdash \langle \text{Seq}\, c_1\, c_2, \text{Normal}\, s \rangle \Rightarrow t$

- Basic: shallow embedding of state updates

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

- Conditional

  $\llbracket s \in b;\ \Gamma \vdash \langle c_1, \text{Normal}\, s \rangle \Rightarrow t \rrbracket \Longrightarrow \Gamma \vdash \langle \text{Cond}\, b\, c_1\, c_2, \text{Normal}\, s \rangle \Rightarrow t$
  $\llbracket s \notin b;\ \Gamma \vdash \langle c_2, \text{Normal}\, s \rangle \Rightarrow t \rrbracket \Longrightarrow \Gamma \vdash \langle \text{Cond}\, b\, c_1\, c_2, \text{Normal}\, s \rangle \Rightarrow t$

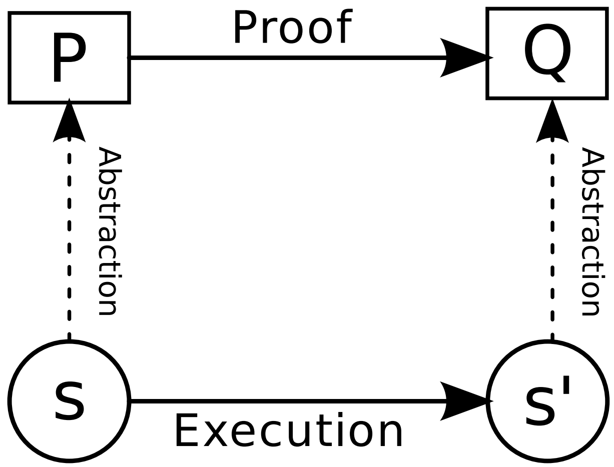
Today

- 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 's assn = " 's set"

- Construct (Hoare-) triple with precondition & postcondition

$$\{\ P\ \}\ sm\ \{\ Q\ \}$$

- 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:
$''\Gamma \vdash \{\!| \acute{}x > 0 \wedge \acute{}y > 0 |\!\} \acute{}z :== \acute{}x + \acute{}y \{\!| \acute{}z > 0 |\!\}''$

- 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

$$\bigwedge x\, y.\ [\![0 < x; 0 < y]\!] \Longrightarrow 0 < x + y$$

- 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. ∃ x∈A. y = f x}

⇒ Can help hiding an existential

- Application: “the normal states satisfying P”

  (s ∈ Normal ‘ P) ⟷ (∃ n. s = Normal n ∧ n ∈ P)

  - Take the desired states P
  - Lift them into `xstate` as `Normal` states
  - Check that s is in that set

Defining Partial Correctness

- Definition correctness

  $\Gamma \models P\ c\ Q{,}A \equiv$
  $\forall s\ t.\ s \in \mathrm{Normal}\ `\ P \longrightarrow \Gamma \vdash \langle c{,}s\rangle \Rightarrow t \longrightarrow t \in \mathrm{Normal}\ `\ Q \cup \mathrm{Abrupt}\ `\ A$

- 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 contollers)

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

  $\Gamma,\Theta \vdash P\ c\ Q,A$

  - Note: Different symbol (from logic: provability vs. validity)
  - P, Q, A have same intended meaning
  - Defined along structure of c ⇨ 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

$$\Gamma,\Theta \vdash \mathrm{P\,c\,Q,A} \quad \Longrightarrow \quad \Gamma,\Theta \models \mathrm{P\,c\,Q,A}$$

  - 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

$$\frac{\{\,P'\,\}\,c\,\{\,Q\,\} \quad P \implies P'}{\{\,P\,\}\,c\,\{\,Q\,\}}$$

- Weakening the post-condition

$$\frac{\{\,P\,\}\,c\,\{\,Q'\,\} \quad Q' \implies Q}{\{\,P\,\}\,c\,\{\,Q\,\}}$$

- 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 Skip Q,A

- Check that sensible: reading in semantics
  - If Q already holds before the execution
  - Then is holds after the execution

- Alternative: backward reading
  - If Q is to hold after the execution
  - Then it must hold already before

Rule for Seq

- Rule for Seq

  $\llbracket \Gamma, \Theta \vdash P\ c_1\ R, A;$
  $\Gamma, \Theta \vdash R\ c_2\ Q, A$
  $\rrbracket \Longrightarrow \Gamma, \Theta \vdash P\ (\mathrm{Seq}\ c_1\ c_2)\ Q, A$

- Check: forward reasoning
  - If P holds before the execution
  - And we can prove that R holds after c_1
  - And we can prove that Q holds after c_2
  - Then Q finally holds
- Backward reasoning
  - If Q must hold after c_2
  - Then R must hold before c_2
  - And P must hold before c_1, i.e. at the start

Conditionals

- Rule for `if`

$$\llbracket \Gamma,\Theta \vdash (P \cap b)\, c_1\, Q,A;$$
$$\Gamma,\Theta \vdash (P \cap \text{-}\, b)\, c_2\, Q,A$$
$$\rrbracket \Longrightarrow \Gamma,\Theta \vdash P\, (\mathrm{Cond}\, b\, c_1\, c_2)\, Q,A$$

- Check by semantic reading
  - If $P \wedge b$ holds before execution
  - And after c_1 we have Q, then finally Q
  - If $P \wedge \neg b$ 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
  - $P \wedge b$ before c_1 or
  - $P \wedge \neg b$ before c_2

The Case of Abrupt Termination

- `Skip` justifies any assertion A (since it never terminates abruptly)

  $\Gamma,\Theta \vdash Q\ \mathsf{Skip}\ Q,A$

- `Seq` justifies A if both c_1 and c_2 justify it

  $$\llbracket \Gamma,\Theta \vdash P\ c_1\ R,A;\ \Gamma,\Theta \vdash R\ c_2\ Q,A \rrbracket \Longrightarrow \Gamma,\Theta \vdash P\ (\mathsf{Seq}\ c_1\ c_2)\ Q,A$$

  - If c_1 terminates with an exception, it must guarantee A
  - If c_2 termiantes with an exception, it must guarantee A

- Same reasoning for `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 $\{\ x > 0 \wedge y > 0\ \}\ z := x + y\ \{\ z > 0\ \}$
  - Strengthen: $\{\ ?P\ \}\ z := x + y\ \{\ z > 0\ \} \wedge$
    $(x > 0 \wedge y > 0 \implies ?P)$
  - Assign: set $?P\ =\ (z > 0)[(x + y)/z]\ =\ x + y > 0$

  $\Rightarrow$ Prove $x > 0 \wedge y > 0 \implies x + y > 0$

$\Rightarrow$ Remove program constructs, prove implications instead

The Rule for `Basic`

- Simpl uses shallow embedding for state updates
- The rule is very short

  $\Gamma,\Theta \vdash \{s.\, f\, s \in Q\}\ (\mathsf{Basic}\, f)\ Q,A$

- 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

$$\frac{\begin{array}{c} P \implies I \\ \{\ I \wedge t\ \}\ b\ \{\ I\ \} \\ I \wedge \neg t \implies Q \end{array}}{\{\ P\ \}\ \textsf{while}\ (t)\ b\ \{\ Q\ \}}$$

  - Invarint must hold before execution
  - After succesful 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 ⇨ rule is

  Γ,Θ⊢ (P ∩ b) c P,A
  $\Longrightarrow \Gamma$,Θ⊢ P (While b c) (P ∩ - b),A

  - Set $P = I$
  - Use $P \wedge \neg b$ for postcondition

- Introduce "hole" for I into abstract syntax

  whileAnno b I c = While b c

- And derive a rule for the new constant

  ⟦ P ⊆ I;
  Γ,Θ⊢ (I ∩ b) c I,A;
  I ∩ -b ⊆ Q
  ⟧ $\Longrightarrow \Gamma$,Θ⊢ P (whileAnno b I c) Q,A

Dealing with Abrupt Termination

- `Throw` causes abrupt termination

  $\Gamma, \Theta \vdash$ A Throw Q,A

  - If A must hold after abrupt termination
  - Then it must already have held before `Throw`
  - Any Q is ok, because `Throw` never terminates normally
- `Catch` finishes abrupt execution

  $[\![\Gamma, \Theta \vdash \mathsf{P}\ \mathsf{c}_1\ \mathsf{Q},\mathsf{R};$
  $\Gamma, \Theta \vdash \mathsf{R}\ \mathsf{c}_2\ \mathsf{Q},\mathsf{A}$
  $]\!] \Longrightarrow \Gamma, \Theta \vdash \mathsf{P}\ \mathsf{Catch}\ \mathsf{c}_1\ \mathsf{c}_2\ \mathsf{Q},\mathsf{A}$

  - 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'

$\Rightarrow$ 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)

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

- Prove specification

  lemma (in mult-impl) mult-spec:
  "$\forall$ A B. $\Gamma \vdash$ ⦃ ´a = A $\wedge$ ´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 Θ

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

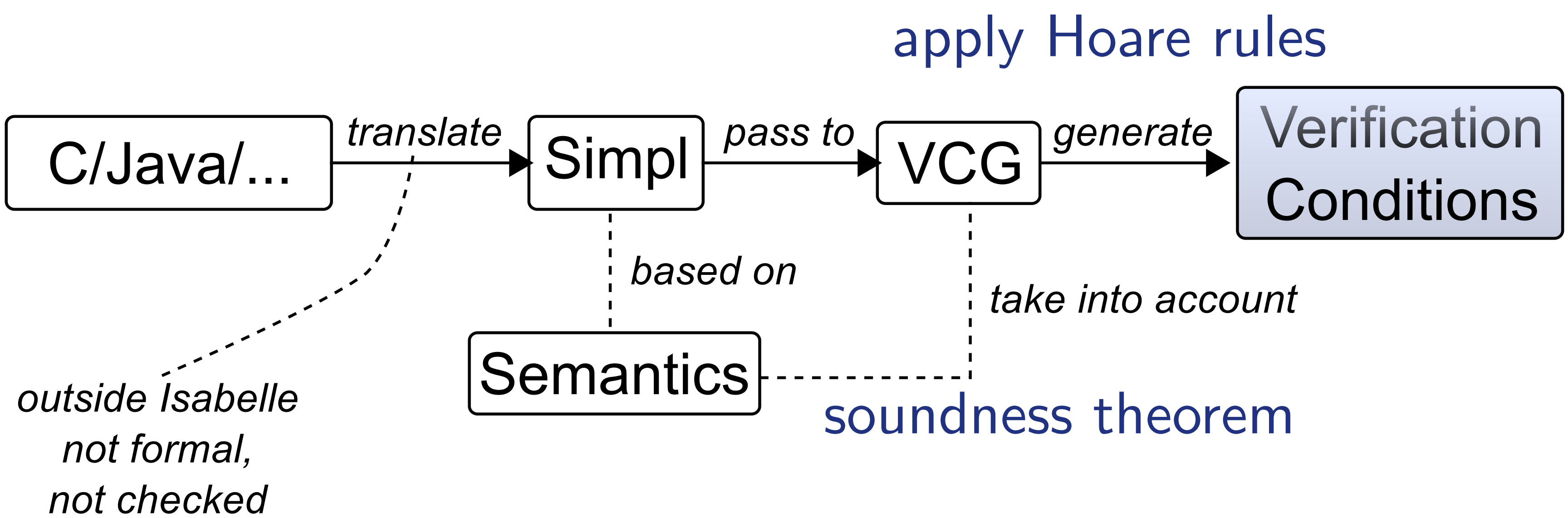
- Definition: partial correctness with context

$$
\begin{aligned}
&\Gamma,\Theta \models \mathsf{P}\,\mathsf{c}\,\mathsf{Q},\mathsf{A} \equiv \\
&\qquad (\forall\,(\mathsf{P},\mathsf{p},\mathsf{Q},\mathsf{A}) \in \Theta.\ \Gamma \models \mathsf{P}\,(\mathsf{Call}\,\mathsf{p})\,\mathsf{Q},\mathsf{A}) \longrightarrow \Gamma \models \mathsf{P}\,\mathsf{c}\,\mathsf{Q},\mathsf{A}
\end{aligned}
$$

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

⇒ In verifying a call, we can assume its specification from Θ

Done!


apply Hoare rules
C/Java/...
translate
Simpl
pass to
VCG
generate
Verification Conditions
based on
take into account
Semantics
soundness theorem
outside Isabelle
not formal,
not checked

References

[1] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18:453–457, August 1975.

[2] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32, Providence, Rhode Island, 1967. American Mathematical Society.

[3] C.A.R Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580,583, October 1969.

[4] Norbert Schirmer. *Verification of Sequential Imperative Programs in Isabelle/HOL*. PhD thesis, Technische Universität München, 2005.

[5] Norbert Schirmer. A sequential imperative programming language syntax, semantics, hoare logics and verification environment. In Gerwin Klein, Tobias Nipkow, and Lawrence Paulson, editors, *The Archive of Formal Proofs*. http://afp.sourceforge.net/entries/Simpl.shtml, February 2008. Formal proof development.