Axiomatic Semantics

Maria João Frade

HASLab - INESC TEC
Departamento de Informática, Universidade do Minho

2021/2022

Axiomatic Semantics

- Meanings of programs are defined indirectly via de axioms and rules of some program logic.
- Specific properties of the effect of executing the commands are expressed as *assertions*. Thus there may be some aspects of the executions that are ignored.
- It give us methods for reasoning about program properties and to prove its correction w.r.t. its specification.

Hoare Logic

Hoare logic

- Hoare logic (also know as Floyd-Hoare logic) is a method of reasoning mathematically about imperative programs.
  - Robert Floyd, "Assigning meaning to programs", 1967.
  - Tony Hoare, "An axiomatic basis for computer programming", 1969.
- The logic deals with the notion of correction w.r.t. a *specification* that consists of
  - a *precondition* - an assertion that is assumed to hold when the execution of the program starts
  - and a *postcondition* - an assertion that is required to hold when execution stops.

A simple programming language - **While**

A While language whose commands are defined over a set of variables $x \in \mathbf{Var}$

$$\mathbf{Aexp} \ni a ::= \ldots \mid -1 \mid 0 \mid 1 \mid \ldots \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2$$

$$\mathbf{Bexp} \ni b ::= \texttt{true} \mid \texttt{false} \mid \neg b \mid b_1 \wedge b_2 \mid a_1 = a_2 \mid a_1 \leq a_2$$

$$\mathbf{Stm} \ni C ::= \mathbf{skip} \mid C_1 \,;\, C_2 \mid x := a \mid \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2 \mid \mathbf{while}\ b\ \mathbf{do}\ C$$

Assertions about programs

- We need formulas that express properties of particular states of the program.
- Program assertions $\phi, \theta, \psi \in \mathbf{Assert}$ (preconditions and postconditions in particular) are *first-order formulas* of a language obtained as an expansion of $\mathbf{Bexp}$.
- Note that assertions may contain occurrences of functions and predicates provided by the user.

Semantics

- Will consider an *interpretation structure* $\mathcal{M} = (D, I)$ for the vocabulary describing the concrete syntax of program expressions.
- The interpretation of expressions depends on a *state*, which is a function that maps each variable into its value. $\mathbf{State} = \mathbf{Var} \to D$
- For the **While** language the set of states is $\mathbf{State} = \mathbf{Var} \to \mathbf{Z}$
- Expressions are interpreted as functions from states to the corresponding domain of interpretation.
- We are considering that expression evaluation
  - is free of side-effects
  - does not go wrong

Semantics of expressions

Defined inductively as before:

- $\mathcal{A} : \mathbf{Aexp} \to (\mathbf{State} \to \mathbf{Z})$

  $$\mathcal{A}[\![a]\!] : \mathbf{State} \to \mathbf{Z}$$

  or simply $$[\![a]\!] : \mathbf{State} \to \mathbf{Z}$$

- $\mathcal{B} : \mathbf{Bexp} \to (\mathbf{State} \to \mathbf{T})$

  $$\mathcal{B}[\![b]\!] : \mathbf{State} \to \mathbf{T}$$

  or simply $$[\![b]\!] : \mathbf{State} \to \mathbf{T}$$

Assertion semantics

- We take the usual interpretation of first-order formulas, noting two facts:
  - interpretation of assertions also depends on $\mathcal{M}$
  - states from $\mathbf{State}$ can be used as *variable assignments*
- The interpretation of the assertion $\phi \in \mathbf{Assert}$ is then given by

$$[\![\phi]\!] : \mathbf{State} \to \mathbf{T}$$

- Since assertions may also contain occurrences of functions and predicates provided by the user, the semantics of those must also be given axiomatically by the user.
- We will be reasoning in the context of a first-order theory that is specified in part by the semantics of program expressions and in part by user-provided axioms.

Program semantics

A natural semantics based on a deterministic evaluation relation

1. $\langle \mathbf{skip}, s\rangle \to s$
2. $\langle x := a, s\rangle \to s[x \mapsto [\![a]\!]s]$
3. if $\langle C_1, s\rangle \to s'$ and $\langle C_2, s'\rangle \to s''$, then $\langle C_1 \,;\, C_2, s\rangle \to s''$
4. if $[\![b]\!]s = \mathbf{tt}$ and $\langle C_t, s\rangle \to s'$, then $\langle \mathbf{if}\ b\ \mathbf{then}\ C_t\ \mathbf{else}\ C_f, s\rangle \to s'$
5. if $[\![b]\!]s = \mathbf{ff}$ and $\langle C_f, s\rangle \to s'$, then $\langle \mathbf{if}\ b\ \mathbf{then}\ C_t\ \mathbf{else}\ C_f, s\rangle \to s'$
6. if $[\![b]\!]s = \mathbf{tt}$, $\langle C, s\rangle \to s'$ and $\langle \mathbf{while}\ b\ \mathbf{do}\ C, s'\rangle \to s''$, then $\langle \mathbf{while}\ b\ \mathbf{do}\ C, s\rangle \to s''$
7. if $[\![b]\!]s = \mathbf{ff}$, then $\langle \mathbf{while}\ b\ \mathbf{do}\ C, s\rangle \to s$

There is no possible *runtime error*, but a program may *diverge*.

Validity

- We assume the existence of "external" means for checking the validity of assertions, in the presence of some *background theory*.
- These tools should additionally allow us to write axioms concerning the uninterpreted functions and predicates.
- Suppose that we wish to encode in the logic a description of what the *factorial* of a number is. The following axioms could be given

$$isfact(0,1)$$
$$\forall n,r.\ n>0 \to isfact(n-1,r) \to isfact(n, n*r)$$

$$\forall n.\ isfact(n, fact(n))$$
$$\forall n,r.\ isfact(n,r) \to r = fact(n)$$

Hoare triples (for partial correction)

- Notation: $\{\phi\}\, C\, \{\psi\}$
  - ϕ is the *precondition*
  - ψ is the *postcondition*
- Denote the *partial correctness* of program C relative to specification (ϕ, ψ)

Intended meaning of $\{\phi\}\, C\, \{\psi\}$

If ϕ holds in a given state and C is executed in that state, then either execution of C does not stop, or *if it does*, ψ will hold in the final state.

Examples

$$\{x=y\}\, x := x+y \,;\, x := 10*x\, \{x = 20*y\}$$

$$\{x=5\}\, \mathbf{while}\ x>0\ \mathbf{do\ skip}\, \{\mathtt{false}\}$$

Hoare triples (for total correction)

- Notation: $[\phi]\, C\, [\psi]$
- Denote the *total correctness* of program C relative to specification (ϕ, ψ)

Intended meaning of $[\phi]\, C\, [\psi]$

If ϕ holds in a given state and C is executed in that state, then execution of C *will stop*, and moreover ψ will hold in the final state of execution.

Examples

$$[x = y]\; x := x + y\,;\; x := 10 * x\; [x = 20 * y]$$

$$[x = 5]\; \textbf{while}\; x > 0\; \textbf{do}\; x := x - 1\; [x = 0]$$

$$[\exists a. x = 10 * a]\; x := x + 18\; [\exists v. x = 2 * v]$$

Semantics of Hoare triples

$\models \{\phi\}\, C\, \{\psi\}$

The Hoare triple $\{\phi\}\, C\, \{\psi\}$ is said to be *valid*, denoted $\models \{\phi\}\, C\, \{\psi\}$, whenever for all $s, s' \in \mathbf{State}$,

$$\text{if } [\![\phi]\!]s = \mathbf{tt} \text{ and } \langle C, s\rangle \to s', \text{ then } [\![\psi]\!]s' = \mathbf{tt}.$$

$\models [\phi]\, C\, [\psi]$

The Hoare triple $[\phi]\, C\, [\psi]$ is said to be *valid*, denoted $\models [\phi]\, C\, [\psi]$, whenever for all $s \in \mathbf{State}$,

$$\text{if } [\![\phi]\!]s = \mathbf{tt}, \text{ then } \exists s' \in \mathbf{State}.\, \langle C, s\rangle \to s' \text{ and } [\![\psi]\!]s' = \mathbf{tt}.$$

Hoare logic as an Axiomatic Semantics (system H)

(skip) $$\frac{}{\{\phi\}\, \mathbf{skip}\, \{\phi\}}$$

(ass) $$\frac{}{\{\psi[e/x]\}\, x := e\, \{\psi\}}$$

(comp) $$\frac{\{\phi\}\, C_1\, \{\theta\} \qquad \{\theta\}\, C_2\, \{\psi\}}{\{\phi\}\, C_1\,;\, C_2\, \{\psi\}}$$

(if) $$\frac{\{\phi \wedge b\}\, C_1\, \{\psi\} \qquad \{\phi \wedge \neg b\}\, C_2\, \{\psi\}}{\{\phi\}\, \mathbf{if}\; b\; \mathbf{then}\; C_1\; \mathbf{else}\; C_2\, \{\psi\}}$$

(while) $$\frac{\{\theta \wedge b\}\, C\, \{\theta\}}{\{\theta\}\, \mathbf{while}\; b\; \mathbf{do}\; C\, \{\theta \wedge \neg b\}}$$

(cons) $$\frac{\{\phi\}\, C\, \{\psi\}}{\{\phi'\}\, C\, \{\psi'\}} \quad \text{if } \phi' \to \phi \text{ and } \psi \to \psi'$$

Loop invariants

- We call *loop invariant* to any property whose validity is preserved by executions of the loop's body.
- Since these executions may only take place when the loop condition is true, an invariant of the loop $\mathbf{while}\; b\; \mathbf{do}\; C$ is any assertion θ such that $\{\theta \wedge b\}\, C\, \{\theta\}$ is valid, in which case of course it also holds that $\{\theta\}\, \mathbf{while}\; b\; \mathbf{do}\; C\, \{\theta \wedge \neg b\}$ is valid.

Warning

Find an adequate loop invariant may be a major difficulty!

Loop variants

- However the validity of $[\theta \wedge b]\, C\, [\theta]$ does not imply the validity of $[\theta]\, \textbf{while}\ b\ \textbf{do}\ C\, [\theta \wedge \neg b]$ *(why?)*

- The required notion here is a *loop variant*: any program expression (or more generally some function on the state) whose value strictly decreases with each iteration, with respect to some well-founded relation.

- The natural choice in our language is to use *non-negative integer* expressions with strictly decreasing values.

$$(\text{while})\quad \frac{[\theta \wedge b \wedge V = v_0]\, C\, [\theta \wedge V < v_0]}{[\theta]\, \textbf{while}\ b\ \textbf{do}\ C\, [\theta \wedge \neg b]} \quad \text{if} \quad \theta \wedge b \to V \geq 0$$

Proof trees

$\vdash_{\mathsf{H}} \{\phi\}\, C\, \{\psi\}$ denote the fact that $\{\phi\}\, C\, \{\psi\}$ is derivable in system H.

$\vdash_{\mathsf{H}} \{x = y\}\, x := x + y\, ;\, x := 10 * x\, \{x = 20 * y\}$

$$\dfrac{\dfrac{\dfrac{}{\{10*(x+y) = 20*y\}\, x := x+y\, \{10*x = 20*y\}}\,(\text{ass})}{\{x = y\}\, x := x+y\, \{10*x = 20*y\}}\,(\text{cons})^{*} \qquad \dfrac{}{\{10*x = 20*y\}\, x := 10*x\, \{x = 20*y\}}\,(\text{ass})}{\{x = y\}\, x := x+y\, ;\, x := 10*x\, \{x = 20*y\}}\,(\text{comp})$$

(cons)* since $x = y \to 10*(x+y) = 20*y$

Alternative display:

$\vdash_{\mathsf{H}} \{x = y\}\, x := x + y\, ;\, x := 10 * x\, \{x = 20 * y\}$

$\{x = y\}\, x := x+y\, ;\, x := 10*x\, \{x = 20*y\}$	(comp)	
1. $\{x = y\}\, x := x+y\, \{10*x = 20*y\}$	(cons)	$x = y \to 10*(x+y) = 20*y$
1.1. $\{10*(x+y) = 20*y\}\, x := x+y\, \{10*x = 20*y\}$	(ass)	
2. $\{10*x = 20*y\}\, x := 10*x\, \{x = 20*y\}$	(ass)	

Soundness

System H is sound w.r.t. the semantics of Hoare triples

If $\vdash_{\mathsf{H}} \{\phi\}\, C\, \{\psi\}$, then $\models \{\phi\}\, C\, \{\psi\}$.

Proof: By induction on the derivation of $\vdash_{\mathsf{H}} \{\phi\}\, C\, \{\psi\}$.

Completeness

- Two major difficulties for proving a program:
  - guess the appropriate intermediate formulas (for sequence, for the loop invariant)
  - prove the logical premises of consequence rule

- System H is complete as long as the assertion language is *sufficiently expressive* to grant the existence of intermediate assertions for reasoning.

System H is complete w.r.t. the semantics of Hoare triples

With $\mathbf{Assert}$ expressive in the above sense, if $\models \{\phi\}\, C\, \{\psi\}$ then $\vdash_{\mathsf{H}} \{\phi\}\, C\, \{\psi\}$.

- This is usually called *relative completeness* [Cook, 1978]

Auxiliary variables

- How to specify formally what the following program does?

$$a := x\,;\, x := y\,;\, y := a$$

- Employ *auxiliary variables*, forbidden to occur in the program, to record initial values of variables.

$$\{x = x_0 \wedge y = y_0\}\, a := x\,;\, x := y\,;\, y := a\, \{x = y_0 \wedge y = x_0\}$$

- In fact, auxiliary variables are required in every specification, to avoid trivial solutions.
  - For instance, an inappropriate specification of factorial would be $(n \geq 0, f = fact(n))$ *(Give some solutions!)*

Exercises

- Prove the validity of the following Hoare triple

$$\{x = x_0 \wedge y = y_0\}\, a := x\,;\, x := y\,;\, y := a\, \{x = y_0 \wedge y = x_0\}$$

- How to specify formally what the following program does?

$$\textbf{if } x < 0 \textbf{ then } x := -x \textbf{ else skip}$$

  Prove its correction w.r.t. the specification proposed.

- Consider the following $\mathbf{While}$ program for calculating x^e

$$\begin{array}{l} r := 1\,; \\ \textbf{while } e > 0 \textbf{ do } \{ \\ \quad r := r * x\,; \\ \quad e := e - 1 \\ \} \end{array}$$

  Specify formally what the following program does and prove its correction w.r.t. the specification proposed.

Annotated programs

- We are interested in automated verification
  - invariants are notoriously difficult to infer automatically
  - in practice loop invariants are typically given by the programmer as an input to the program verification process
- The syntactic class of *annotated programs*

$$\mathbf{AStm} \ni C \;::=\; \textbf{skip} \mid x := e \mid C_1\,;\, C_2 \mid \textbf{if } b \textbf{ then } C_1 \textbf{ else } C_2 \mid \textbf{while } b \textbf{ do } \{\theta\}\, C$$

- Annotations do not affect the operational semantics.
- The (while) rule

$$\frac{\{\theta \wedge b\}\, C\, \{\theta\}}{\{\theta\}\, \textbf{while } b \textbf{ do } \{\theta\}\, C\, \{\theta \wedge \neg b\}}$$

Annotated programs

- Whereas in the standard presentation a program can be proved correct with respect to a specification if there exists adequate invariants for proving it, with annotated loops a program can only be proved correct if it is *correctly annotated*.
- Soundness is preserved.
- Completeness does not hold, since the annotated invariants may be inadequate for deriving the triple.

The factorial example

The following is an example of a correctly annotated program w.r.t. the specification

$$(n \geq 0, f = \mathit{fact}(n))$$

Let **fact** be

$$
\begin{array}{l}
f := 1\,;\, i := 1\,; \\
\textbf{while}\ i \leq n\ \textbf{do}\, \{f = \mathit{fact}(i-1) \wedge i \leq n+1\}\, \{ \\
\quad f := f * i\,; \\
\quad i := i + 1 \\
\}
\end{array}
$$

A proof of $\{n \geq 0\}$ **fact** $\{f = \mathit{fact}(n)\}$ will be given later.

Mechanising Hoare Logic
(extra)

Mechanising Hoare logic

- In H system two desirable properties for backward proof construction are missing:
  - sub-formula property
  - unambiguous choice of rule

$$\frac{\{\phi\}\, C_1\, \{\theta\} \qquad \{\theta\}\, C_2\, \{\psi\}}{\{\phi\}\, C_1\, ;\, C_2\, \{\psi\}} \qquad\qquad \frac{\{\phi\}\, C\, \{\psi\}}{\{\phi'\}\, C\, \{\psi'\}}\ \text{if}\ \ \phi' \to \phi\ \ \text{and}\ \ \psi \to \psi'$$

- The consequence rule causes ambiguity. Its presence is however necessary to make possible the application of rules for skip, assignment, and while, as well as reuse.
- An alternative is to distribute the side conditions among the different rules.

Hg a goal-directed system

$$\text{(skip)} \quad \frac{}{\{\phi\}\, \textbf{skip}\, \{\psi\}}\ \text{if}\ \phi \to \psi$$

$$\text{(ass)} \quad \frac{}{\{\phi\}\, x := e\, \{\psi\}}\ \text{if}\ \phi \to \psi[e/x]$$

$$\text{(comp)} \quad \frac{\{\phi\}\, C_1\, \{\theta\} \qquad \{\theta\}\, C_2\, \{\psi\}}{\{\phi\}\, C_1\, ;\, C_2\, \{\psi\}}$$

$$\text{(if)} \quad \frac{\{\phi \wedge b\}\, C_1\, \{\psi\} \qquad \{\phi \wedge \neg b\}\, C_2\, \{\psi\}}{\{\phi\}\, \textbf{if}\ b\ \textbf{then}\ C_1\ \textbf{else}\ C_2\, \{\psi\}}$$

$$\text{(while)} \quad \frac{\{\theta \wedge b\}\, C\, \{\theta\}}{\{\phi\}\, \textbf{while}\ b\ \textbf{do}\, \{\theta\}\, C\, \{\psi\}}\ \text{if}\ \begin{array}{l} \phi \to \theta\ \ \text{and} \\ \theta \wedge \neg b \to \psi \end{array}$$

Hg properties

Admissibility of the consequence rule in Hg

If $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$, $\models \phi' \to \phi$, and $\models \psi \to \psi'$, then $\vdash_{\mathsf{Hg}} \{\phi'\}\, C\, \{\psi'\}$.

Proof: By induction on the derivation of $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$.

Let $\lfloor\cdot\rfloor : \mathbf{AStm} \to \mathbf{Stm}$ be a function that erases all annotations from a program (defined in the obvious way).

Soundness of Hg

If $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$, then $\vdash_{\mathsf{H}} \{\phi\}\, \lfloor C\rfloor\, \{\psi\}$.

Proof: By induction on the derivation of $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$.

Correctly-annotated program

We say that C is *correctly-annotated* w.r.t. (ϕ, ψ) if $\vdash_{\mathsf{H}} \{\phi\}\, \lfloor C\rfloor\, \{\psi\}$ implies $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$.

A strategy for proofs

- Focus on the command and postcondition; guess an appropriate precondition that guarantees the given postcondition.
- In the rules for skip, assignment, and while, the precondition is determined by looking at the side condition and choosing the weakest condition that satisfies it.
- In the sequence rule, we obtain the intermediate condition by propagating the postcondition.

A strategy for proofs

- $\{\phi\}\, x := e_1\, ;\, y := e_2\, ;\, z := e_3\, \{\psi\}$

  $\boxed{1.}$ $\{\phi\}\, x := e_1\, ;\, y := e_2\, \{\theta\}$
  $\boxed{2.}$ $\{\theta\}\, z := e_3\, \{\psi\}$

- Now the second sub-goal is an assignment, which means that the corresponding axiom can be applied by simply taking the precondition to be the one that trivially satisfies the side condition, i.e. $\theta = \psi[e_3/z]$. Now of course this can be substituted globally in the current proof construction

- $\{\phi\}\, x := e_1\, ;\, y := e_2\, ;\, z := e_3\, \{\psi\}$

  $\boxed{1.}$ $\{\phi\}\, x := e_1\, ;\, y := e_2\, \{\psi[e_3/z]\}$
  2. $\{\psi[e_3/z]\}\, z := e_3\, \{\psi\}$

A strategy for proofs

- $\{\phi\}\, x := e_1\, ;\, y := e_2\, ;\, z := e_3\, \{\psi\}$

  $\boxed{1.}$ $\{\phi\}\, x := e_1\, ;\, y := e_2\, \{\psi[e_3/z]\}$
  $\boxed{1.1.}$ $\{\phi\}\, x := e_1\, \{\psi[e_3/z][e_2/y]\}$
  1.2. $\{\psi[e_3/z][e_2/y]\}\, y := e_2\, \{\psi[e_3/z]\}$
  2. $\{\psi[e_3/z]\}\, z := e_3\, \{\psi\}$

- $\{\phi\}\, x := e_1\, ;\, y := e_2\, ;\, z := e_3\, \{\psi\}$

  1. $\{\phi\}\, x := e_1\, ;\, y := e_2\, \{\psi[e_3/z]\}$
  1.1. $\{\phi\}\, x := e_1\, \{\psi[e_3/z][e_2/y]\}$,
  1.2. $\{\psi[e_3/z][e_2/y]\}\, y := e_2\, \{\psi[e_3/z]\}$
  2. $\{\psi[e_3/z]\}\, z := e_3\, \{\psi\}$

- In step 1.1 we were not free to choose the precondition for the assignment since this is now the first command in the sequence. Thus the side condition $\phi \to \psi[e_3/z][e_2/y][e_1/x]$ is introduced.

Using the weakest precondition strategy to verify **fact**

$\{n \geq 0\}$ **fact** $\{f = fact(n)\}$

1. $\{n \geq 0\}\ f := 1\,;\ i := 1\ \{n \geq 0 \wedge f = 1 \wedge i = 1\}$
   - 1.1. $\{n \geq 0\}\ f := 1\ \{n \geq 0 \wedge f = 1\}$
   - 1.2. $\{n \geq 0 \wedge f = 1\}\ i := 1\ \{n \geq 0 \wedge f = 1 \wedge i = 1\}$
2. $\{n \geq 0 \wedge f = 1 \wedge i = 1\}$
   $\textbf{while}\ i \leq n\ \textbf{do}\ \{f = fact(i-1) \wedge i \leq n+1\}\ C_w$
   $\{f = fact(n)\}$
   - 2.1. $\{f = fact(i-1) \wedge i \leq n+1 \wedge i \leq n\}\ C_w\ \{f = fact(i-1) \wedge i \leq n+1\}$
     - 2.1.1. $\{f = fact(i-1) \wedge i \leq n+1 \wedge i \leq n\}\ f := f * i\ \{f = fact(i-1) * i \wedge i \leq n\}$
     - 2.1.2. $\{f = fact(i-1) * i \wedge i \leq n\}\ i := i+1\ \{f = fact(i-1) \wedge i \leq n+1\}$

where C_w represents the command $f := f * i\,;\ i := i+1$.

Using the weakest precondition strategy to verify **fact**

- The following side conditions are required for each node of the tree:

  1.1 $n \geq 0 \rightarrow (n \geq 0 \wedge f = 1)[1/f]$
  1.2 $n \geq 0 \wedge f = 1 \rightarrow (n \geq 0 \wedge f = 1 \wedge i = 1)[1/i]$
  2. $n \geq 0 \wedge f = 1 \wedge i = 1 \rightarrow f = fact(i-1) \wedge i \leq n+1$ and
  $f = fact(i-1) \wedge i \leq n+1 \wedge \neg(i \leq n) \rightarrow f = fact(n)$
  2.1.1. $f = fact(i-1) \wedge i \leq n+1 \wedge i \leq n \rightarrow (f = fact(i-1) * i \wedge i \leq n)[f * i/f]$
  2.1.2. $f = fact(i-1) * i \wedge i \leq n \rightarrow (f = fact(i-1) \wedge i \leq n+1)[i+1/i]$

- The validity of these conditions is fairly obvious in the current theory.

An architecture for program verification

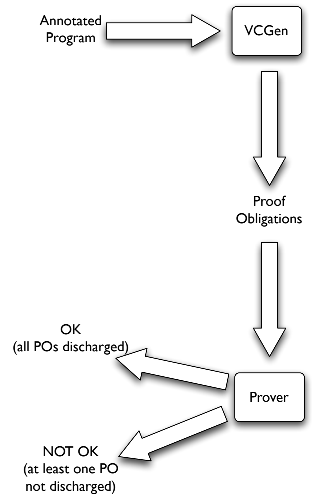
At this point we may outline a method for program verification as follows.

1. Mechanically produce a derivation with $\{\phi\}\,C\,\{\psi\}$ as conclusion, assuming that all the side conditions created in this process hold. The side conditions are called *Verification Conditions (VCs)* or *Proof Obligations (POs)*
2. Send the VCs generated in step 1 to some proof tool in order to be checked.
3. If all VCs are shown to be valid by a proof tool, then $\{\phi\}\,C\,\{\psi\}$ is valid.

Verification Conditions Generator

The mechanisation of the construction of the proof tree following the weakest precondition strategy does not even explicitly construct the proof tree; it just outputs the set of verification conditions.
This algorithm is called a *Verification Conditions Generator (VCGen)*.

An architecture for program verification



Discharging the VCs

- VCs are first-order formulas whose validity is to be checked w.r.t. a *background theory*.
- The VCs are discharged using proof tools.
- *Automated proof tools* (such as SMT-solvers) are usually the first choice.
  - It is possible to use a multi-prover approach (as can be seen with Frama-C/Why3)
- If no conclusive answer is given (recall FOL is semi-decidable) one must use a *proof assistant*.
- If the automated prover finds a counter-example (or if the interactive proof does not succeed), then we do not have a proof tree for the Hoare triple. That means the verification of the program has *failed!*

Warning

This may be due to errors in the *program*, *specification* or *annotations*!

Weakest liberal precondition

[Dijkstra, 1975]

Given a command C and a postcondition ψ, $\mathsf{wlp}(C,\psi)$ should return the minimal precondition ϕ that validates the triple $\{\phi\}\,C\,\{\psi\}$.

$$\begin{array}{rcl}
\mathsf{wlp}(\mathbf{skip},\psi) & = & \psi \\
\mathsf{wlp}(x := e,\psi) & = & \psi[e/x] \\
\mathsf{wlp}(C_1;C_2,\psi) & = & \mathsf{wlp}(C_1,\mathsf{wlp}(C_2,\psi)) \\
\mathsf{wlp}(\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,\psi) & = & (b \to \mathsf{wlp}(C_1,\psi)) \wedge (\neg b \to \mathsf{wlp}(C_2,\psi)) \\
\mathsf{wlp}(\mathbf{while}\ b\ \mathbf{do}\ \{\theta\}\,C,\psi) & = & \theta
\end{array}$$

VCGen algorithm

VC produces a set of verification conditions from a program and a postcondition

$$\begin{array}{rcl}
\mathsf{VC}(\mathbf{skip},\psi) & = & \emptyset \\
\mathsf{VC}(x := e,\psi) & = & \emptyset \\
\mathsf{VC}(C_1;C_2,\psi) & = & \mathsf{VC}(C_1,\mathsf{wlp}(C_2,\psi)) \ \cup\ \mathsf{VC}(C_2,\psi) \\
\mathsf{VC}(\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2,\psi) & = & \mathsf{VC}(C_1,\psi) \ \cup\ \mathsf{VC}(C_2,\psi) \\
\mathsf{VC}(\mathbf{while}\ b\ \mathbf{do}\ \{\theta\}\,C,\psi) & = & \{(\theta\wedge b) \to \mathsf{wlp}(C,\theta), (\theta \wedge \neg b) \to \psi\} \\
 & & \cup\ \mathsf{VC}(C,\theta)
\end{array}$$

$$\mathsf{VCG}(\{\phi\}\,C\,\{\psi\}) \ = \ \{\phi \to \mathsf{wlp}(C,\psi)\} \ \cup\ \mathsf{VC}(C,\psi)$$

VCGen algorithm

Some observations:

- The function VC simply follows the structure of the rules of system Hg to collect the union of all sets of verification conditions.
- According to the weakest precondition strategy the side conditions generated are trivially satisfied (so we do not collect them).
- In fact, only the loop rule actually introduces verification conditions that need to be checked.
- To understand the clause for loops, it may help to observe that this clause is just an expansion of

$$\mathsf{VC}(\mathbf{while}\ b\ \mathbf{do}\ \{\theta\}\,C,\psi) = \{(\theta\wedge\neg b) \to \psi\} \cup \mathsf{VCG}(\{\theta\wedge b\}\,C\,\{\theta\})$$

Properties of VCGen

Soundness

If $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$, then

1. $\vdash_{\mathsf{Hg}} \{\mathsf{wlp}(C,\psi)\}\, C\, \{\psi\}$
2. $\models \phi \to \mathsf{wlp}(C,\psi)$

Proof: By induction on the structure of C.

Adequacy of VCGen

$$\models \mathsf{VCG}(\{\phi\}\, C\, \{\psi\}) \quad \text{iff} \quad \vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$$

Proof:

$\Rightarrow$) By induction on the structure of C.

$\Leftarrow$) By induction on the derivation of $\vdash_{\mathsf{Hg}} \{\phi\}\, C\, \{\psi\}$.

Applying the VCGen algorithm to **fact**

- Start by calculating $\mathsf{VC}(\mathbf{fact}, f = \mathit{fact}(n))$.
- Then do the calculation of $\mathsf{VCG}(\{n \geq 0\}\, \mathbf{fact}\, \{f = \mathit{fact}(n)\})$.
- The end result should be following set of proof obligations.
  1. $n \geq 0 \to 1 = \mathit{fact}(1-1) \wedge 1 \leq n+1$
  2. $f = \mathit{fact}(i-1) \wedge i \leq n+1 \wedge i \leq n \to f * i = \mathit{fact}(i+1-1) \wedge i+1 \leq n+1$
  3. $f = \mathit{fact}(i-1) \wedge i \leq n+1 \wedge i > n \to f = \mathit{fact}(n)$