Hoare Logic
G52DOA - Derivation of Algorithms

Venanzio Capretta

Syntax of programs

We work with a simplified programming language. It has, anyway, all the important features of any complete programming language.

A program is a structured sequence of commands that operates on a *store*. The store contains the values for the variables in the program.

The definition of its syntax is:

$$\begin{array}{rl} \mathit{Program} ::= & \mathsf{skip} \mid \mathit{Program}; \mathit{Program} \mid \mathsf{x} := \mathit{term} \mid \\ & \mathsf{if}\ b\ \mathsf{then}\ \mathit{Program}\ \mathsf{else}\ \mathit{Program} \mid \mathsf{while}\ b\ \mathsf{do}\ \mathit{Program}. \end{array}$$

This means that a program will have one of the following forms:

- skip: This is a program that does nothing, after its execution the store is exactly how it was before;
- $p_1; p_2$: Here p_1 and p_2 are programs and we compose them with the ";" operator, program p_1 is executed first, when it terminates, program p_2 is executed;
- $\mathsf{x} := t$: This command modifies the store by evaluating the term t in the present store and then assigning its value to the variable x;
- if b then p_1 else p_2: This is a conditional command, it evaluates the boolean expression b, if its value is true it executes program p_1, if its value is false it executes p_2;
- while b do p: This command evaluates the boolean expression b, if it is true it executes the program p and then repeats itself, otherwise it terminates.

We also use parentheses to structure the programs. Basically, "(" and ")" have the roles more often covered by "{" and "}" or begin and end in other programming languages. So, for example, the program

$$\mathsf{while}\ \mathsf{x} > 0\ \mathsf{do}\ (\mathsf{x} := \mathsf{x} - 1; \mathsf{y} := \mathsf{x} * \mathsf{y})$$

is different from the program

$$\mathsf{while}\ \mathsf{x} > 0\ \mathsf{do}\ \mathsf{x} := \mathsf{x} - 1; \mathsf{y} := \mathsf{x} * \mathsf{y}$$

because the instruction $\mathsf{y} := \mathsf{x} * \mathsf{y}$ is inside the loop in the first one, outside in the second one.

Hoare Triples

We want to define a logical system that allows us to make formal statements about the working of programs and prove them by exact logical derivation. First of all, we define the notion of *specification*. A specification is a proposition that states something about the content of the store before and after the execution of a program. The form of a specification is a *Hoare Triple*:

$$\{P\}\ p\ \{Q\}$$

where P and Q are logical propositions, stating some property of the variables used in the program p. The above Hoare triple is interpreted as saying:

> *If initially the store is in a state that satisfies the proposition P, and we execute the program p, after the execution the store will be in a state satisfying the proposition Q.*

Here is an example of a true Hoare triple:

$$\{\mathsf{x} > 0 \wedge \mathsf{x} = x_0\}\ \mathsf{y} := 0; \mathsf{while}\ \mathsf{x} > 0\ \mathsf{do}\ (\mathsf{y} := \mathsf{x} + \mathsf{y}; \mathsf{x} := \mathsf{x} - 1)\ \{\mathsf{y} = x_0(x_0 + 1)/2\}.$$

and an example of a false Hoare triple:

$$\{\mathsf{x} > 0 \wedge \mathsf{x} = x_0\}\ \mathsf{y} := 0; \mathsf{while}\ \mathsf{x} > 0\ \mathsf{do}\ (\mathsf{x} := \mathsf{x} - 1; \mathsf{y} := \mathsf{x} * \mathsf{y})\ \{\mathsf{y} = x_0!\}.$$

Program and Ghost Variables

As you can see in the examples above, we use two kinds of variables, for which I use different typographical fonts.

Program Variables written x, y, etc. are the variables used in the program, they denote memory locations in the store, their value will change during the execution of the program.

Ghost Variables (also called *logical variables*), written x, y, etc. are not used in the program, they denote fixed values and don't change during the execution of the program; they are used to "remember" the value of a program variable at a specific point in the execution to refer to it later. In the examples above, the ghost variable x_0 is used to refer to the value that the program variable x has at the beginning of the computation (at the end x has value 0).

Rules of Hoare Logic

We give now the rules of derivation for Hoare Triples. The rules are formulated in *natural deduction* style: They are all of the form:

$$\frac{\mathcal{A}_1 \quad \cdots \quad \mathcal{A}_n}{\mathcal{B}}$$

meaning that if we proved the propositions $\mathcal{A}_1 \ldots \mathcal{A}_n$, then we can apply the rule to prove $\mathcal{B}$. Here are the rules:

$$\frac{}{\{P\}\ \mathsf{skip}\ \{P\}}\ \text{Skip} \qquad \frac{}{\{P[e/\mathsf{x}]\}\ \mathsf{x} := e\ \{P\}}\ \text{Assignment}$$

$$\frac{\{P\}\ p_1\ \{Q\} \quad \{Q\}\ p_2\ \{R\}}{\{P\}\ p_1; p_2\ \{R\}}\ \text{Composition}$$

$$\frac{\{P \wedge b = \mathsf{true}\}\ p_1\ \{Q\} \quad \{P \wedge b = \mathsf{false}\}\ p_2\ \{Q\}}{\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2\ \{Q\}}\ \text{Conditional}$$

$$\frac{\{I \wedge b = \mathsf{true}\}\ p\ \{I\}}{\{I\}\ \mathsf{while}\ b\ \mathsf{do}\ p\ \{I \wedge b = \mathsf{false}\}}\ \text{Loop}$$

$$\frac{P \to P_0 \quad \{P_0\}\ p\ \{Q_0\} \quad Q_0 \to Q}{\{P\}\ p\ \{Q\}}\ \text{Implied}$$

Proof Tableaux for Hoare Logic

A *proof tableau* is a way to represent a derivation in Hoare Logic. We write the program in the middle of the page, with every command occupying a separate line. We draw two vertical bars on the left and right of the program. On each side we write pre- and post-conditions for every instruction:

$$\begin{array}{l|l|l} & & \{P_0\} \\ \{P_1\} & c_1; & \{Q_1\} \\ \{P_2\} & c_2; & \{Q_2\} \\ & \vdots & \\ \{P_n\} & c_n & \{Q_n\} \end{array}$$

Here P_0 is the precondition of the program and is written by itself on the right on the first line, which is otherwise empty; Q_n is the postcondition of the program and is written on the right of the last line. Every line must be justified directly by one of the rules of Hoare Logic. The *passage* from one line to the next must be justified by the use of the implication rule, that is, we must prove that $P_0 \to P_1$, $Q_1 \to P_2$, etcetera.

In general, the method to complete a proof tableau is to work from the postcondition up: from a proposition written on the right of a line, apply the appropriate Hoare logic rule to obtain the proposition on the left; then try to simplify it and use the simplified proposition at the right side of the line above.

This method is called the *weakest precondition* calculation: always use the *tightest* preconditions that fit the rules.

Empty Steps

At some point in the derivation we may want to do some purely logical steps to change the proposition we are working with. We can do this by leaving the line empty and proving the implication of the proposition on the right-hand side:

$$\begin{array}{c|c|l} \cdots & \cdots & \{P\} \\ & & \{Q\} \\ \cdots & \cdots & \cdots \end{array}$$

and proving that $P \to Q$.

Assignment

For an assignment instruction, you obtain the left-hand side proposition by making the substitution as in the Hoare logic rule.

$$\begin{array}{c|c|l} \cdots & \cdots & \{?\} \\ \{?\} & \mathsf{x} := e & \{Q\} \end{array} \implies \begin{array}{r|c|l} \cdots & \cdots & \{?\} \\ \{Q[e/\mathsf{x}]\} & \mathsf{x} := e & \{Q\} \end{array} \implies \begin{array}{r|c|l} \cdots & \cdots & \{P\} \\ \{Q[e/\mathsf{x}]\} & \mathsf{x} := e & \{Q\} \end{array}$$

Where P is some proposition such that $P \to Q[e/\mathsf{x}]$ is provable; usually it is just a simplification of $Q[e/\mathsf{x}]$.

Here is a derivation of the Hoare triple:

$$\{\mathsf{x} = x_0 \wedge \mathsf{y} = y_0\}\ \mathsf{x} := \mathsf{x} + \mathsf{y}; \mathsf{y} := \mathsf{x} - \mathsf{y}; \mathsf{x} := \mathsf{x} - \mathsf{y}\ \{\mathsf{x} = y_0 \wedge \mathsf{y} = x_0\}$$

using the tableaux method:

$$\begin{array}{r|l|l} \{\mathsf{x} + \mathsf{y} = x_0 + y_0 \wedge \mathsf{y} = y_0\} & \mathsf{x} := \mathsf{x} + \mathsf{y}; & \{\mathsf{x} = x_0 \wedge \mathsf{y} = y_0\} \\ \{\mathsf{x} = x_0 + y_0 \wedge \mathsf{x} - \mathsf{y} = x_0\} & \mathsf{y} := \mathsf{x} - \mathsf{y}; & \{\mathsf{x} = x_0 + y_0 \wedge \mathsf{y} = x_0\} \\ \{\mathsf{x} - \mathsf{y} = y_0 \wedge \mathsf{y} = x_0\} & \mathsf{x} := \mathsf{x} - \mathsf{y} & \{\mathsf{x} = y_0 \wedge \mathsf{y} = x_0\} \end{array}$$

with additional proofs of the implications:

$$\begin{aligned} \mathsf{x} = x_0 \wedge \mathsf{y} = y_0 &\to \mathsf{x} + \mathsf{y} = x_0 + y_0 \wedge \mathsf{y} = y_0 \\ \mathsf{x} = x_0 + y_0 \wedge \mathsf{y} = y_0 &\to \mathsf{x} = x_0 + y_0 \wedge \mathsf{x} - \mathsf{y} = x_0 \\ \mathsf{x} = x_0 + y_0 \wedge \mathsf{y} = x_0 &\to \mathsf{x} - \mathsf{y} = y_0 \wedge \mathsf{y} = x_0. \end{aligned}$$

Conditional Command

If we want to prove a Hoare triple for a conditional instruction:

$$\{P\}\ \mathsf{if}\ b\ \mathsf{then}\ p_1\ \mathsf{else}\ p_2\ \{Q\}$$

We start by writing the statement in four lines, with the precondition on the right-hand side of the line above the statement and the postcondition on the

right-hand side of the empty line under the statement:

	$\{P\}$
if b then	
p_1	
else	
p_2	
	$\{Q\}$

We fill in some of the missing proposition according to the rule of Hoare logic for conditionals: We write the precondition on the left side just before the if statement; Immediately after the then statement, on the right, we fill in the conjunction of the precondition and the boolean expression in the conditional; Similarly after the else statement we fill in the conjunction of the precondition and the negation of the expression; In both cases, we fill in the postcondition after the end of the corresponding subprograms:

$\{P\}$		$\{P\}$
	if b then	$\{P \wedge b = \mathsf{true}\}$
	p_1	$\{Q\}$
	else	$\{P \wedge b = \mathsf{false}\}$
	p_2	$\{Q\}$
		$\{Q\}$

We complete the tableau by filling in the left-hand sides according to the rules pertaining to the instructions present in p_1 and p_2:

$\{P\}$		$\{P\}$
	if b then	$\{P \wedge b = \mathsf{true}\}$
$\{R_1\}$	p_1	$\{Q\}$
	else	$\{P \wedge b = \mathsf{false}\}$
$\{R_2\}$	p_2	$\{Q\}$
		$\{Q\}$

Finally, we must prove that these propositions on the left side follow from statement on the right of then and else, respectively:

$$P \wedge b = \mathsf{true} \to R_1$$
$$P \wedge b = \mathsf{false} \to R_2$$

As an example, here is the proof of correctness of a program that computes the absolute value (as it is common, we identify a boolean expression b with the proposition $b = \mathsf{false}$):

$\{\top\}$		$\{\top\}$
	if $\mathsf{x} \leq 0$ then	$\{\mathsf{x} \leq 0\}$
$\{-\mathsf{x} = \lvert\mathsf{x}\rvert\}$	$\mathsf{y} := -\mathsf{x}$	$\{\mathsf{y} = \lvert\mathsf{x}\rvert\}$
	else	$\{\neg\mathsf{x} \leq 0\}$
$\{\mathsf{x} = \lvert\mathsf{x}\rvert\}$	$\mathsf{y} := \mathsf{x}$	$\{\mathsf{y} = \lvert\mathsf{x}\rvert\}$
		$\{\mathsf{y} = \lvert\mathsf{x}\rvert\}$

with additional proofs of the implications:

$$\mathsf{x} \leq 0 \rightarrow -\mathsf{x} = |\mathsf{x}|$$
$$\neg \mathsf{x} \leq 0 \rightarrow \mathsf{x} = |\mathsf{x}|.$$

Skip

The skip statement has a very simple rule: just copy the proposition on its right to its left:

$$\{?\} \;\Big|\; \vdots \\ \mathsf{skip} \\ \vdots \;\Big|\; \{Q\} \qquad \Longrightarrow \qquad \{Q\} \;\Big|\; \vdots \\ \mathsf{skip} \\ \vdots \;\Big|\; \{Q\}$$

Here is an example of a proof of a specification for a program that uses the skip command:

$$\begin{array}{r|l|l} \{\mathsf{x} = x_0\} & \mathsf{if}\ \mathsf{x} < 0\ \mathsf{then} & \{\mathsf{x} = x_0\} \\ \{-\mathsf{x} = |x_0|\} & \quad \mathsf{x} := -\mathsf{x} & \{\mathsf{x} = x_0 \wedge \mathsf{x} < 0\} \\ & \mathsf{else} & \{\mathsf{x} = |x_0|\} \\ \{\mathsf{x} = |x_0|\} & \quad \mathsf{skip} & \{\mathsf{x} = x_0 \wedge \neg \mathsf{x} < 0\} \\ & & \{\mathsf{x} = |x_0|\} \\ & & \{\mathsf{x} = |x_0|\} \end{array}$$

with additional proofs of:

$$\mathsf{x} = x_0 \wedge \mathsf{x} < 0 \rightarrow -\mathsf{x} = |x_0|$$
$$\mathsf{x} = x_0 \wedge \neg \mathsf{x} < 0 \rightarrow \mathsf{x} = |x_0|.$$