Supplement for the 3rd lecture

Department of Computer Science and Engineering
Isao Sasano

In this document we explain some supplementary explanation to Hoare triples and substitutions.

1 A derivation of the Hoare triple in p. 17

We show a derivation of the Hoare triple {true} **while** true **do** x:=1 {false} in p. 17 as follows.

$$\dfrac{\dfrac{\dfrac{\text{true} \Rightarrow \text{true} \wedge \text{true} \quad \overline{\{\text{true}\}\ \text{x:=1}\ \{\text{true}\}}\ (\text{assign})}{\{\text{true} \wedge \text{true}\}\ \text{x:=1}\ \{\text{true}\}}\ (\text{conseq})}{\{\text{true}\}\ \textbf{while}\ \text{true}\ \textbf{do}\ \text{x:=1}\ \{\text{true} \wedge \neg\ \text{true}\}}\ (\text{while}) \qquad \text{true} \wedge \neg\text{true} \Rightarrow \text{false}}{\{\text{true}\}\ \textbf{while}\ \text{true}\ \textbf{do}\ \text{x:=1}\ \{\text{false}\}}\ (\text{conseq})$$

In the above derivation tree, $\{\text{true}\}\ x := 1\ \{\text{true}\}$ holds from the assignment axiom since

$$\text{true}[1/\text{x}] = \text{true}$$

holds. In the above derivation we abbreviate the assignment axiom as assign, the consequence rule as conseq, and the while rule as while.

2 Definition of substitution

Here we define substitution for logical expression, which is used in the assignment axiom. Firstly, we assume that the logical expressions used in this lecture as follows, although it is not explicitly mentioned in the slides because

of the lack of space.

$$
\begin{array}{rcl}
P & := & \mathrm{true} \mid \mathrm{false} \\
& \mid & P \wedge P \mid P \vee P \mid \neg P \mid P \Rightarrow P \\
& \mid & E \leq E \mid E \geq E \mid E < E \mid E > E \mid E = E \\
E & := & N \\
& \mid & V \\
& \mid & E + E \\
& \mid & E - E \\
N & := & \cdots \mid -2 \mid -1 \mid 0 \mid 1 \mid 2 \mid \cdots \\
V & := & x \mid y \mid z \mid \cdots
\end{array}
$$

This kind of definition is called *inductive definition*, which is out of scope of this lecture. We inductively define substitutions for the logical expressions defined above as follows, which is also out of scope of this lecture.

$$
\begin{array}{rcll}
P[E/x] & = & \text{case } P \text{ of} & \\
& & \mathrm{true} & \rightarrow \quad \mathrm{true} \\
& & \mathrm{false} & \rightarrow \quad \mathrm{false} \\
& & P_1 \wedge P_2 & \rightarrow \quad P_1[E/x] \wedge P_2[E/x] \\
& & P_1 \vee P_2 & \rightarrow \quad P_1[E/x] \vee P_2[E/x] \\
& & \neg P & \rightarrow \quad \neg P[E/x] \\
& & P_1 \Rightarrow P_2 & \rightarrow \quad P_1[E/x] \Rightarrow P_2[E/x] \\
& & E_1 \leq E_2 & \rightarrow \quad E_1[E/x] \leq E_2[E/x] \\
& & E_1 \geq E_2 & \rightarrow \quad E_1[E/x] \geq E_2[E/x] \\
& & E_1 < E_2 & \rightarrow \quad E_1[E/x] < E_2[E/x] \\
& & E_1 > E_2 & \rightarrow \quad E_1[E/x] > E_2[E/x] \\
& & E_1 = E_2 & \rightarrow \quad E_1[E/x] = E_2[E/x] \\
E[E_0/x] & = & \text{case } E \text{ of} & \\
& & N & \rightarrow \quad N \\
& & E_1 + E_2 & \rightarrow \quad E_1[E_0/x] + E_2[E_0/x] \\
& & E_1 - E_2 & \rightarrow \quad E_1[E_0/x] - E_2[E_0/x] \\
& & V & \rightarrow \quad \text{if } V = x \text{ then } E_0 \text{ else } V
\end{array}
$$

3 Notation of Hoare triples

People use various notations for Hoare triples. In the slides we used the notation of the form $\{P_1\}\ S\ \{P_2\}$, while the original paper by Hoare [1] used

the notation of the form P_1 $\{S\}$ P_2, where the statements are surrounded by braces.

4 A Hoare triple for linear search code

Here, we present a Hoare triple for linear search code, following the content of Section 3.5 in the reference book [2]. As explained in class, a statement in an imperative language where the control flow has a single entry and exit point can be characterized by the conditions that hold at the entry and exit points (preconditions and postconditions). As an example let us consider linear search, which can be performed efficiently using a sentinel. In Pascal, the following code fragment can be used.

```
A[0] := x;
i := n;
while x <> A[i] do i := i-1
```

This is a code fragment that performs a linear search to find the position in array `A` where the value is equal to `x`. The operator `<>` checks if the values of the expressions on the left and right are not equal. When this code fragment is executed, the value of variable `i` will be as follows immediately after execution:

- If `x` is within the range of `A[1]` to `A[n]`, the index of the rightmost occurrence among them (assuming `A[1]` ... `A[n]` is arranged from left to right).
- 0 if not found.

We would like to prove that this actually happens using Hoare logic (extended to handle arrays). To do so, it suffices to prove the following Hoare triple:

```
{n≥1}
A[0] := x;
i := n;
while x <> A[i] do i := i-1
{(i=0 ∧ x∉A[1...n]) ∨ (0<i≤n ∧ x=A[i] ∧ x∉A[i+1...n])}
```

Here, $x \notin a[i \ldots j]$ denotes that x is not within the range from $a[i]$ to $a[j]$. It is assumed to hold if $i > j$.

The following assumptions are made:

- `x` contains the value being searched for.
- The indices of array `A` include the indices `0` to `n`.
- The search is performed within the range of indices `1` to `n` in array `A`. `A[0]` is reserved as a sentinel.
- Elements of array `A` are of integer type, and the variables `i`, `x`, and `n` are of integer type.
- $\texttt{n} \geq 1$

The above Hoare triple holds if $\{P\}$ `while x<>A[i] do i=i-1` $\{Q\}$ holds, where P represents

$$\texttt{x} = \texttt{A[0]} \land \texttt{x} \notin \texttt{A[i+1} \ldots \texttt{n]} \land 0 \leq \texttt{i} \leq \texttt{n}$$

and Q represents

$$\texttt{x} = \texttt{A[i]} \land \texttt{x} \notin \texttt{A[i+1} \ldots \texttt{n]} \land 0 \leq \texttt{i} \leq \texttt{n}$$

(Demonstrating this requires extending Hoare logic to support array element assignment, which is omitted here.) The proof tree for Hoare triple $\{P\}$ `while x<>A[i] do i=i-1` $\{Q\}$ is as follows.

$$\dfrac{\dfrac{\dfrac{P \land \texttt{x<>A[i]} \Rightarrow P[\texttt{i-1/i}] \quad \overline{\{P[\texttt{i-1/i}]\}\ \texttt{i=i-1}\ \{P\}}\ \text{(assign)}}{\{P \land \texttt{x<>A[i]}\}\ \texttt{i=i-1}\ \{P\}}\ \text{(conseq)}}{\{P\}\ \texttt{while x<>A[i] do i=i-1}\ \{P \land \neg\texttt{x<>A[i]}\}}\ \text{(while)} \qquad P \land \neg\texttt{x<>A[i]} \Rightarrow Q}{\{P\}\ \texttt{while x<>A[i] do i=i-1}\ \{Q\}}\ \text{(conseq)}$$

References

[1] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, 1969.

[2] Ravi Sethi. *Programming Languages Concepts and Constructs, second edition.* Addison Wesley, 1996.