

Hoare Logic (II): Procedures

(Based on [Gries 1981; Slonneger and Kurtz 1995])

Yih-Kuen Tsay

Department of Information Management
National Taiwan University

Non-recursive Procedures

- We first consider procedures with *call-by-value* parameters (and *global variables*).
- Syntax:

proc $p(\text{in } x); S$

where x may be a list of variables, S does not contain p , and S does not change x .

- Inference rule:

$$\frac{\{P\} S \{Q\}}{\{P[a/x] \wedge I\} p(a) \{Q[a/x] \wedge I\}}$$

where a may not be a global variable changed by S and I does not refer to variables changed by S .

How It May Go Wrong

- Example: **proc** $p(\text{in } x); b := 2x;$
- Below is an incorrect usage of the rule

$$\frac{\{x = 1\} b := 2x \{b = 2 \wedge x = 1\}}{\{(x = 1)[b/x]\} p(b) \{(b = 2 \wedge x = 1)[b/x]\}}$$

since the conclusion is not valid

$$\{b = 1\} p(b) \{b = 2 \wedge b = 1\}.$$

- The inference rule cannot be applied, because the global variable b is changed by procedure p .
- The problem is that x becomes an alias of b in the invocation $p(b)$, while $\{x = 1\} b := 2x \{b = 2 \wedge x = 1\}$ does not take this into account.

Non-recursive Procedures (cont.)

🌐 We now consider procedures with *call-by-value*, *call-by-value-result*, and *call-by-result* parameters.

🌐 Syntax:

proc p(**in** x ; **in out** y ; **out** z); S


where x, y, z may be lists of variables, S does not contain p , and S does not change x .

🌐 Inference rule:

$$\frac{\{P\} S \{Q\}}{\{P[a, b/x, y] \wedge I\} p(a, b, c) \{Q[a, b, c/x, y, z] \wedge I\}}$$

where b, c are (lists of) distinct variables, a, b, c may not be global variables changed by S , and I does not refer to variables changed by S .

Non-recursive Procedures (cont.)

-  Using wp , one can justify the rule with the understanding that “ $p(a, b, c)$ ” is equivalent to “ $x, y := a, b; S; b, c := y, z$ ”.

Recursive Procedures

- 🌐 A rule for recursive procedures without parameters:

$$\frac{\{P\} p() \{Q\} \vdash \{P\} S \{Q\}}{\vdash \{P\} p() \{Q\}}$$

where p is defined as “**proc** $p()$; S ”.

- 🌐 A rule for recursive procedures with parameters:

$$\frac{\forall v(\{P[v/x]\} p(v) \{Q[v/x]\}) \vdash \{P\} S \{Q\}}{\vdash \{P[a/x]\} p(a) \{Q[a/x]\}}$$

where p is defined as “**proc** $p(\mathbf{in} x)$; S ” and a may not be a global variable changed by S .

An Example

```

proc nonzero();
begin
    read x;
    if x = 0 then nonzero() fi;
end

```

- 🌐 The semantics of “**read** x” is defined as follows:

$$\{IN = v \cdot L \wedge P[v/x]\} \text{ **read** } x \{IN = L \wedge P\}$$

where v is a single value and L is a stream of values.

- 🌐 We wish to prove the following:

$$\{IN = Z \cdot n \cdot L \wedge \text{“}Z \text{ contains only zeros”} \wedge n \neq 0\} // \{P\}$$

$$\text{nonzero()};$$

$$\{IN = L \wedge x = n \wedge n \neq 0\} // \{Q\}$$

An Example (cont.)

- It amounts to proving the following annotation:

```

proc nonzero();
begin
  { $IN = Z \cdot n \cdot L \wedge$  "Z contains only zeros"  $\wedge n \neq 0$ } // { $P$ }
  read x;
  if  $x = 0$  then nonzero() fi;
  { $IN = L \wedge x = n \wedge n \neq 0$ } // { $Q$ }
end

```

- The first step is to find a suitable assertion R between “**read** x ” and the “**if**” statement.
- For this, we consider two cases: (1) Z is empty and (2) Z is not empty.

An Example (cont.)

- 🌐 Case 1: Z is empty
 $\{IN = n \cdot L \wedge n \neq 0\}$
read x
 $\{IN = L \wedge x = n \wedge n \neq 0\}$
- 🌐 Case 2: Z is not empty
 $\{IN = 0 \cdot Z' \cdot n \cdot L \wedge \text{"Z' contains only zeros"} \wedge n \neq 0\}$
read x
 $\{IN = Z' \cdot n \cdot L \wedge \text{"Z' contains only zeros"} \wedge n \neq 0 \wedge x = 0\}$
- 🌐 Applying the **Disjunction** rule, we get a suitable R :

$$(IN = L \wedge x = n \wedge n \neq 0) \vee$$

$$(IN = Z' \cdot n \cdot L \wedge \text{"Z' contains only zeros"} \wedge n \neq 0 \wedge x = 0)$$

An Example (cont.)

- 🌐 We now have to prove the following:

$$\{R\} \text{ if } x = 0 \text{ then nonzero() fi } \{IN = L \wedge x = n \wedge n \neq 0\}$$

- 🌐 From the **Conditional** rule, this breaks down to

- ☀️ $\{R \wedge x = 0\} \text{ nonzero() } \{IN = L \wedge x = n \wedge n \neq 0\}$

- ☀️ $(R \wedge x \neq 0) \rightarrow (IN = L \wedge x = n \wedge n \neq 0)$ (obvious)

- 🌐 The first case involving the recursive call simplifies to

$$\begin{aligned} &\{IN = Z' \cdot n \cdot L \wedge \text{"Z' contains only zeros"} \wedge n \neq 0 \wedge x = 0\} \\ &\text{nonzero()} \\ &\{IN = L \wedge x = n \wedge n \neq 0\} \end{aligned}$$

- 🌐 The precondition is stronger than we need and $x = 0$ can be removed.

An Example (cont.)

- Finally, we are left with the following proof obligation:

$$\{IN = Z' \cdot n \cdot L \wedge \text{"Z' contains only zeros"} \wedge n \neq 0\}$$

nonzero()

$$\{IN = L \wedge x = n \wedge n \neq 0\}$$

- The induction hypothesis gives us exactly the above.
- And, this completes the proof.

Termination of Recursive Procedures

- Consider the previous recursive procedure again.

```
proc nonzero();  
begin  
    read  $x$ ;  
    if  $x = 0$  then nonzero() fi;  
end
```

- Given an input of the form $IN = L_1 \cdot n \cdot L_2$, where L_1 contains only zero values and $n \neq 0$, the command “nonzero()” will halt.
- We prove this *by induction* on the length of L_1 .

Proving Termination by Induction

- 🌐 Basis: $\text{length}(L_1) = 0$
 - ☀️ The input has the form $IN = n \cdot L_2$, where $n \neq 0$.
 - ☀️ After “**read** x ”, $x \neq 0$.
 - ☀️ The boolean test $x = 0$ does not pass and the procedure call terminates.
- 🌐 Induction step: $\text{length}(L_1) = k > 0$
 - ☀️ Hypothesis: $\text{nonzero}()$ halts when $\text{length}(L_1) = k - 1 \geq 0$.
 - ☀️ Let $L_1 = 0 \cdot L'_1$.
 - ☀️ The call $\text{nonzero}()$ is invoked with $IN = 0 \cdot L'_1 \cdot n \cdot L_2$, where L'_1 contains only zero values and $n \neq 0$.

Induction step (cont.)

- ☀ After “**read** x ”, $x = 0$.
- ☀ This boolean test $x = 0$ passes and a second call `nonzero()` is invoked inside the **if** statement.
- ☀ The second `nonzero()` is invoked with $L'_1 \cdot n \cdot L_2$, where L'_1 contains only zero values and $n \neq 0$
- ☀ Since $\text{length}(L'_1) = k - 1$, termination is guaranteed by the hypothesis.



🌐 A rule for proving termination of recursive procedures:

$$\frac{\{\exists u \in W (u < Z \wedge P(u))\} p() \{Q\} \vdash \{P(Z)\} S \{Q\}}{\vdash \{\exists t \in W (P(t))\} p() \{Q\}}$$

where

- ☀️ $(W, <)$ is a well-founded set,
- ☀️ p is defined as “**proc** $p()$; S ”, and
- ☀️ Z is a “rigid” variable that ranges over W and does not occur in P , Q , or S .

References

-  D. Gries. *The Science of Programming*, Springer-Verlag, 1981.
-  K. Slonneger and B.L. Kurtz. *Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach*, Addison-Wesley, 1995.