Program verification

Nelma Moreira

Program verification
Lecture 5

Procedures

- Until now we consider a program as a sequence of commands
- The treatment of subrotines is challenging from the oint of view of verification: procedures or functions)
- The treatment of procedures and functions includes the following aspects:
  - recursive calls (that can lead to non termination in the evaluation of expressions);
  - parameters;
- A program will be a set of procedures annotated with contracts.
- We will not consider here an operational semantics for procedures but assume that there exists one and the program logic will be adequate.
- We start with procedures without parameters.

ma

Procedures and Recursion

We suppose that procedures have no parameters.

- **proc** $p = C_p$ defines a procedure p;
- the command C_p is the body of the procedure p ($\mathbf{body}(p)$);
- the new command **call** p invokes the procedure, transfering execution to the body of p;
- A natural semantics rule could be:

$$\frac{\langle \mathbf{body}(p), s\rangle \quad\longrightarrow\quad s'}{\langle \mathbf{call}\ p, s\rangle \quad\longrightarrow\quad s'}$$

- for non recursive procedures the rule of Hoare logic is

$$\frac{\{\varphi\}\mathbf{body}(p)\{\psi\}}{\{\varphi\}\mathbf{call}\ p\{\psi\}}$$

Example

Consider the procedure

proc FACT =
  f ← 1;
  i ← 1;
  while $i \leq n$ **do**
    $\{f = fact(i-1) \text{ and } i \leq n+1\}$
    $f \leftarrow f \times i;$
    $i \leftarrow i + 1$

By the correction of the body we have:

$$\{n \geq 0 \wedge n = n_0\}\mathbf{body}(\text{FACT})\{f = fact(n) \wedge n = n_0\}$$

Allying the above rule we have:

$$\{n \geq 0 \wedge n = n_0\}\mathbf{call}\ \text{FACT}\{f = fact(n) \wedge n = n_0\}$$

Adaptation

We can use the adapted consequence rule for sistem $\mathcal{H}$

$$\frac{\{\varphi\}C\{\psi\}}{\{\varphi'\}C\{\psi'\}} \quad \text{se} \;\; \varphi' \rightarrow \forall \overline{x_f}.(\forall \overline{y_f}.(\varphi[\overline{y_f}/\overline{y}] \rightarrow \psi[\overline{y_f}/\overline{y}, \overline{x_f}/\overline{x}]) \rightarrow \psi'[\overline{x_f}/\overline{x}])$$

to reuse the above deduction for a stronger precondition

$$\frac{\{n \geq 0 \wedge n = n_0\}\mathbf{call}\ \text{FACT}\{f = fact(n) \wedge n = n_0\}}{\{n = 10\}\mathbf{call}\ \text{FACT}\{f = fact(10)\}}$$

and we obtain the side condition

$$\begin{aligned} n = \;\; & 10 \rightarrow \forall n_f, f_f.((\forall n_{0f}.n \geq 0 \wedge n = n_{0f} \rightarrow f_f = fact(n_f) \wedge n_f = n_{0f}) \\ & \rightarrow f_f = fact(10)) \end{aligned}$$

For the system $\mathcal{H}_g$ this is not possible because it lacks a consequence rule, but we may have a specific rule to dela with recursive procedures.

Notation $\tilde{}$

In practice specification languages avoid the generality allowed by auxiliary variables and forbid their use in the procedure specifications.

Given a variable x we denote $\tilde{x}$ its value in the prestate.

For the previous example we have

$$\{n \geq 0\}\textbf{call } \textsc{fact}\{f = fact(n) \wedge n = \tilde{n}\}$$

The new consequence rule is

$$\frac{\{\varphi\}C\{\psi\}}{\{\varphi'\}C\{\psi'\}} \quad \text{se} \quad \varphi' \to \forall \overline{x_f}.((\varphi \to \lfloor \psi[\overline{x_f}/\overline{x}] \rfloor) \to \psi'[\overline{x_f}/\overline{x}])$$

and $\lfloor \psi[\overline{x_f}/\overline{x}] \rfloor$ denotes the result of substituting in $\psi[\overline{x_f}/\overline{x}]$ every variable $\tilde{x}$ by the corresponding x.

The triple can be derived by

$$\frac{\{n \geq 0\}\textbf{call } \textsc{fact}\{f = fact(n) \wedge n = \tilde{n}\}}{\{n = 10\}\textbf{call } \textsc{fact}\{f = fact(10)\}}$$

and we obtain the side condition

$$n = 10 \to \forall n_f, f_f.((n \geq 0 \to f_f = fact(n_f) \wedge n_f = n) \to f_f = fact(10))$$

To derive the triple with $\tilde{x}$ one needs to modify call rule as follows

$$\frac{\{\varphi \wedge x = \tilde{x_1} \wedge \ldots \wedge x_n = \tilde{x_n}\}\textbf{body}(p)\{\psi\}}{\{\varphi'\}\textbf{call } p\{\psi'\}}$$

where $x_1, \ldots, x_n$ are the program variables

Recursive procedures

- In recursive procedures, `body(p)` can contain commands `call p`
- The application of the rule for procedures given above can lead to infinite derivations.
- The following rule was proposed by Hoare

$$\frac{\begin{array}{c}[\{\varphi\}\textbf{call } p\{\psi\}] \\ \vdots \\ \{\varphi\}\textbf{body}(p)\{\psi\}\end{array}}{\{\varphi\}\textbf{call p}\{\psi\}}$$

Assuming $\{\varphi\}\textbf{call}\, p\{\psi\}$ we can derive $\{\varphi\}\textbf{body}(p)\{\psi\}$, then $\{\varphi\}\textbf{call}\, p\{\psi\}$ can be derived without hypotheses (and that is why the hypothesis had square brackets).

- It is an axiomatic counterpart of fixpoint induction.

Example

Consider the procedure

$\textbf{proc}\;\; \text{FACTR} =$
  $\textbf{if}\;\; n == 0\; \textbf{then}$
    $f \leftarrow 1$
  $\textbf{else}$
    $n \leftarrow n - 1;$
    $\textbf{call}\; \text{FACTR};$
    $n \leftarrow n + 1;$
    $f \leftarrow n \times f$

then

$$\{n \geq 0 \wedge n = n_0\}\textbf{call}\; \text{FACTR}\{f = fact(n) \wedge n = n_0\}$$

can be derived using the adapted consequence rule.

Procedure calls in $\mathcal{H}_g$

In this case the side conditions of the rule for procedures should include an adaptation condition

$$\frac{\{\varphi\}\textbf{body}(p)\{\psi\}}{\{\varphi'\}\textbf{call}\; p\{\psi'\}} \text{ if } \;\varphi' \to \forall\overline{x_f}.(\forall\overline{y_f}.\varphi[\overline{y_f}/\overline{y}] \to \psi[\overline{y_f}/\overline{y}, \overline{x_f}/\overline{x}]) \to \psi'[\overline{x_f}/\overline{x}])$$

where $\overline{y}$ are the auxiliary variables of $\{\varphi\}\textbf{body}(p)\{\psi\}$, $\overline{x}$ are the program variables of $\textbf{body}(p)$, and $\overline{y_f}$, $\overline{x_f}$ fresh. The idea of the rule is that the body of p is proved correct with respect to (φ, ψ), then this specification should be strong enough to adapt the procedure to weaker specifications.

[AFPMdS11] Chap. 8.1

References

[AFPMdS11] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simão Melo de Sousa. *Rigorous Software Development: An Introduction to Program Verification.* Springer, 2011.