Program verification

Nelma Moreira

Program verification
Lecture 8

Procedures

- Until now we consider a program as a sequence of commands
- The treatment of subrotines is challenging from the point 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.

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**
    $\{f = fact(i-1)$ and $i \leq n+1\}$
    $f \leftarrow f \times i;$
    $i \leftarrow i+1$

By the correction of the body we have:

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

Applying the above rule we have:

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

Modularity

- In verification it is useful that one can reuse correctness results;
- Let

  $$\texttt{fact} = f \leftarrow 1; i \leftarrow 1; \texttt{while}\ i \leq n\ \texttt{do}\ (f \leftarrow f \times i; i \leftarrow i+1)$$

  and $fact(n) = n!$, and we have a proof of

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

  we would like to use this result to prove a weaker specification:

  $$\{n = 10\}\texttt{fact}\{f = fact(n)\}$$

  This can be achieved using the consequence rule.
- However, if we have,

  $$\{n \geq 0 \wedge n = n_0\}\texttt{fact}\{f = fact(n) \wedge n = n_0\}$$

  we cannot derive the weaker triple.

Adaptation

The problem of matching a proved specification of a program with a weaker specification is called the *adaptation problem* (without the full proof of this last specification).

(Satisfiable specification) A specification (φ, ψ) is satisfiable if there is a program C such that $\models \{\varphi\}C\{\psi\}$.

(Adaptation completeness) Let (φ, ψ) satisfiable and for any program C we have $\models \{\varphi'\}C\{\psi'\}$ whenever $\models \{\varphi\}C\{\psi\}$. A deductive system of Hoare triples is *adaptation complete* iff for any program C the following rule is derivable.

$$\frac{\{\varphi\}C\{\psi\}}{\{\varphi'\}C\{\psi'\}}$$

Hoare logic is not adaptation complete, due to the presence of auxiliary variables.

- Informally, auxiliary variables are universally quantified over Hoare triples, connecting pre and post conditions. But, the side conditions in $cons_p$ rule do not take that in consideration.
- A solution was proposed by Kleymann, considering a stronger consequence rule, formalizing the difference between program and auxiliary variables.
- In the consequence rule

$$\frac{\{\varphi\}C\{\psi\}}{\{\varphi'\}C\{\psi'\}} \quad \text{if} \;\; \varphi' \to \varphi \wedge \psi \to \psi'$$

- The first side condition is interpreted in the pre-state, whereas the second is interpreted in the post-state. Both should communicate through the auxiliary variables.
- The auxiliary variables in ψ have to be interpreted in the pre-state and should be existentially quantified:in the factorial example $n = 10 \to n \geq 0 \wedge n = n_0$, does not hold, but $n = 10 \to \exists n_0 . n \geq 0 \wedge n = n_0$ does.

The adequate side condition suggested by Kleymann has the form

$$\varphi' \to (\varphi \wedge (\psi \to \psi'))$$

Let $\overline{y}$ be the auxiliary variables in $\{\varphi\}C\{\psi\}$, quantification is introduced as follows:

$$\varphi' \to \exists \overline{y_f} . (\varphi[\overline{y_f}/\overline{y}] \wedge (\psi[\overline{y_f}/\overline{y}] \to \psi'))$$

We interpret the auxiliary variables in φ' and ψ' and substituted program variables in the post-state by universally quantified fresh variables

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

where $\overline{y}$ are the auxiliary variables in $\{\varphi\}C\{\psi\}$, $\overline{x}$ the program variables in C, and $\overline{y_f}$, $\overline{x_f}$ are fresh variables.

- The previous rule works for total correctness.
- we have a weaker condition por partial correctness

$$\varphi' \to ((\varphi \to \psi) \to \psi')$$

The program variables are now universally quantified

$$\varphi' \to (\forall \overline{y}.(\varphi \to \psi) \to \psi')$$

The resulting rule is

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

where $\overline{y}$ are auxiliary variables $\{\varphi\}C\{\psi\}$, $\overline{x}$ are the program variables of C and $\overline{y_f}$ and $\overline{x_f}$ fresh.

We will only use these new consequence rules to deal with recursive procedures

Example

Given the assertion:

$$\{n \geq 0 \wedge n = n_0\}\texttt{fact}\{f = fact(n) \wedge n = n_0\}$$

To derive a weaker assertion:

$$\{n = 10\}\texttt{fact}\{f = fact(n)\}$$

we obtain the side condition

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

Adaptation

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

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

where

- $\overline{y}$ are the auxiliar variables in $\{\varphi\}C\{\psi\}$
- $\overline{x}$ program variables in C
- $\overline{x_f}$ and $\overline{y_f}$ fresh variables

to reuse the above deduction for a stronger precondition

$$\frac{\{n \geq 0 \wedge n = n_0\}\textbf{call } \textsc{fact}\{f = fact(n) \wedge n = n_0\}}{\{n = 10\}\textbf{call } \textsc{fact}\{f = fact(10)\}}$$

and we obtain the side condition

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

For the system $\mathcal{H}_g$ this is not possible because it lacks a consequence rule, but we may have a specific rule to deal 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 pre-state.

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

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

where $\overline{x}$ are program variables 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

$$\{n = 10\}\textbf{call } \textsc{fact}\{f = fact(10)\}$$

can be derived by consequence rule

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

and we obtain the side condition, considering n and f program variables,

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

To derive the triple with $\tilde{x}$

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

one needs to modify the **call** rule as follows

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

where $x_1, \ldots, x_n$ are the program variables of $\mathbf{body}(p)$.

In the example

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

Exerc. 8.1. *Given* $\{x > 0 \wedge y > 0\}\mathbf{body}(p)\{z = x + y \wedge x = \tilde{x}\}$ *for some procedure* p*, apply the deduction rules and obtain the verification conditions to ensure the derivation of*

$$\{\varphi\}\mathbf{call}\ p\{\psi\}$$

where

$$\varphi \textit{ is } x > 0 \wedge y > 0 \wedge x = \tilde{x} + 100$$
$$\psi \textit{ is } z = x + y \wedge x = \tilde{x} + 100$$

◇

Recursive procedures

- In recursive procedures, $\mathbf{body}(\mathbf{p})$ 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

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

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

- For total correctness new rules with variants need to be introduce.
- It is an axiomatic counterpart of fixpoint induction.

Example

Consider the procedure

proc FACTR =
  if $n == 0$ **then**
    $f \leftarrow 1$
  else
    $n \leftarrow n - 1$;
    call 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 an adapted consequence rule

Procedure calls in $\mathcal{H}_g$ (idea)

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. (For more [AFPMdS11] Chap. 8.1).

Contracts and mutual recursion

- we consider programs as a set of procedures and a set of global variables
- procedures communicate through the global variables and thus do not have parameters
- procedures can be mutually recursive
- this also models very simple object-oriented programming languages

We extend the syntax of the programming language:

- **PN** is a set of procedure names p, q, ...
- **Proc** are procedure definitions, Φ

- **Prog** are programs, Π
- **Pspec** are programs correctness formulas, S_p

$$\begin{aligned} \Phi &::= \mathbf{pre}\,\varphi\,\mathbf{post}\,\psi\,\mathbf{proc}\,p = C \\ \Pi &::= \Phi \mid \Pi\,\Phi \\ S_p &::= \{\Pi\} \end{aligned}$$

And let

- $\tilde{\mathbf{Var}} = \{\tilde{x} \mid x \in \mathbf{Var}\}$
- $\tilde{\mathbf{Var}}(\varphi) = \{x \mid \tilde{x} \text{ occurs in } \varphi\}$
- $\lfloor\theta\rfloor = \theta[x_1/\tilde{x_1}, \ldots, x_n/\tilde{x_n}]$, for any formula θ such that $\tilde{\mathbf{Var}}(\theta) = \{x_1, \ldots, x_n\}$.

Given a program Π with a procedure `p`,

$$\mathbf{pre}\,\varphi\,\mathbf{post}\,\psi\,\mathbf{proc}\,p = C$$

we define

$$\begin{aligned} \mathbf{pre}(p) &= \varphi \\ \mathbf{post}(p) &= \psi \\ \mathbf{body}(p) &= C \end{aligned}$$

And

- $\mathbf{pre}(p)$ and $\mathbf{post}(p)$ contain no auxiliar variables (only either program variables or quantified logical variables)
- $\mathbf{pre}(p)$ has no occurrence of $\tilde{}$ variables.

Contract triple

Given a program Π a *contract triple* for a procedure p is

$$\{\mathbf{pre}(p)\}\mathbf{call}\ p\{\mathbf{post}(p)\}$$

A program Π is *correct*, denoted by $\{\Pi\}$, if all procedures are correct with respect to their specifications

$$\models \{\Pi\} \iff \ \models \{\mathbf{pre}(p) \wedge x_1 = \tilde{x_1} \wedge \cdots \wedge x_k = \tilde{x_k}\}\mathbf{call}\ p\{\mathbf{post}(p)\}, \ \ \forall p \in \mathbf{PN}(\Pi).$$

where we suppose $\mathbf{PN}(\Pi) = \{p_1, \ldots, p_n\}$.

Deductive System for Parameterless Procedures, $\mathcal{H}_g$

[mutual recursion parameterless]

$$\frac{\begin{array}{cc} [\{\Pi\}] & [\{\Pi\}] \\ \vdots & \vdots \\ \{\widetilde{\mathbf{pre}(p_1)}\}\mathbf{body}(p_1)\,\{\mathbf{post}(p_1)\} & \{\widetilde{\mathbf{pre}(p_n)}\}\mathbf{body}(p_n)\,\{\mathbf{post}(p_n)\} \end{array}}{\{\Pi\}}$$

$$\begin{array}{r} \text{where } \widetilde{\mathbf{pre}(p_i)} = \mathbf{pre}(p_i) \wedge x_1 = \tilde{x_1} \wedge \cdots \wedge x_k = \tilde{x_k} \\ \tilde{\mathbf{Var}}(\mathbf{post}(p_i)) = \{x_1, \ldots, x_k\} \end{array}$$

[procedure call parameterless]

$$\frac{\{\Pi\}}{\{\varphi\}\mathbf{call}\ p\{\psi\}} \text{ if } \ \varphi \rightarrow \forall \overline{x_f}.((\mathbf{pre}(p) \rightarrow \lfloor \mathbf{post}(p)[\overline{x_f}/\overline{x}] \rfloor) \rightarrow \psi[\overline{x_f}/\overline{x}])$$

$$\begin{array}{r} \text{where } \overline{x} = \mathbf{Var}(\mathbf{post}(p)) \cup \mathbf{Var}(\psi) \\ \overline{x_f} \text{ are fresh variables} \end{array}$$

Example

Let Π be the program below with $\mathbf{PN}(\Pi) = \{p_1, p_2\}$

pre $x > 0 \wedge y > 0$
post $x = \tilde{x} \wedge y = 2 \times \tilde{y} \wedge z = x + y \wedge z > 2$
proc $p_1 =$
  $y \leftarrow 2 \times y;$
  $z \leftarrow x + y$
pre $x > 0$
post $x = \tilde{x} \wedge y = 2 \times \tilde{y} \wedge z = 3 \times \tilde{x} + 200$
proc $p_2 =$
  $y \leftarrow x + 100;$
  call p_1

Verification Conditions Generator

We can extend the VCG algorithm to cope with procedures. For the **call** command we have

$$wp(\mathbf{call}\ p,\psi) = \forall\overline{x_f}.((\mathbf{pre}(p) \to \lfloor \mathbf{post}(p)[\overline{x_f}/\overline{x}]\rfloor) \to \psi[\overline{x_f}/\overline{x}])$$
$$VC(\mathbf{call}\ p,\psi) = \emptyset$$

The set of verification conditions for a program correctness formula $\{\Pi\}$ can be computed by

$$VCG(\{\Pi\}) = \bigcup_{p\in\mathbf{PN}(\Pi)} VCG(\{\widetilde{\mathbf{pre}(p)}\}\mathbf{body}(p)\,\{\mathbf{post}(p)\})$$

Exerc. 8.2. *Considering the program* Π *given above compute* $VCG(\{\Pi\})$ $\diamond$

$$VCG(\{\widetilde{\mathbf{pre}(p_1)}\}\mathbf{body}(p_1)\,\{\mathbf{post}(p_1)\})$$
$$= VCG(\{x > 0 \wedge y > 0 \wedge\ x = \tilde{x} \wedge\ y = \tilde{y} \wedge\ z = \tilde{z}\}$$
$$y \leftarrow 2\times y; z \leftarrow x + y$$
$$\{x = \tilde{x} \wedge y = 2\times\tilde{y} \wedge z = x + y \wedge z > 2\})$$
$$= \{x > 0 \wedge y > 0 \wedge\ x = \tilde{x} \wedge\ y = \tilde{y} \wedge\ z = \tilde{z} \to$$
$$x = \tilde{x} \wedge 2\times y = 2\times\tilde{y} \wedge x + 2\times y = x + 2\times y \wedge 2\times y + x > 2\}$$

$$VCG(\{\widetilde{\mathbf{pre}(p_2)}\}\mathbf{body}(p_2)\,\{\mathbf{post}(p_2)\})$$
$$= VCG(\{x > 0\ \wedge\ x = \tilde{x} \wedge\ y = \tilde{y} \wedge\ z = \tilde{z}\}$$
$$y \leftarrow x + 100; \mathbf{call}\ p_1\{z = 3\times\tilde{x} + 200\})$$
$$= \{x > 0\ \wedge\ x = \tilde{x} \wedge\ y = \tilde{y} \wedge\ z = \tilde{z} \to$$
$$wp(\mathbf{call}\ p_1, z = 3\times\tilde{x} + 200)[x + 100/y]\}$$
$$= \{x > 0\ \wedge\ x = \tilde{x} \wedge\ y = \tilde{y} \wedge\ z = \tilde{z} \to$$
$$\forall x_f, y_f, z_f.((x > 0 \wedge x + 100 > 0 \to x_f = x \wedge y_f = 2\times(x + 100) \wedge z_f = x_f + y_f \wedge z_f > 2$$
$$\to z_f = 3\times\tilde{x} + 200)$$

where

$$wp(\mathbf{call}\ p_1, z = 3\times\tilde{x} + 200)$$
$$= \forall x_f, y_f, z_f.((x > 0 \wedge y > 0 \to \lfloor x_f = \tilde{x} \wedge y_f = 2\times\tilde{y} \wedge z_f = x_f + y_f \wedge z_f > 2\rfloor)$$
$$\to z_f = 3\times\tilde{x} + 200)$$
$$= \forall x_f, y_f, z_f.((x > 0 \wedge y > 0 \to x_f = x \wedge y_f = 2\times y \wedge z_f = x_f + y_f \wedge z_f > 2)$$
$$\to z_f = 3\times\tilde{x} + 200)$$

articleAll verification conditions are valid thus the program is correct.

Frame conditions

After the execution of a call command **call** p nothing is assumed about the value of the variables that do not occur in $\mathbf{Var}(\mathbf{post}(p)) \cup \mathbf{Var}(\psi)$ according to the correctness rule:

$$\frac{\{\Pi\}}{\{\varphi\}\mathbf{call}\ p\{\psi\}} \text{ if } \ \varphi \to \forall \overline{x_f}.((\mathbf{pre}(p) \to \lfloor \mathbf{post}(p)[\overline{x_f}/\overline{x}]\rfloor) \to \psi[\overline{x_f}/\overline{x}])$$

Thus if one wants to connect the value of any variable between the pre-state and the post-state this must be expressed in $\mathbf{post}(p)$.

If one knows which variables p modifies then one could have

$$\overline{x} = \mathbf{frame}(p)$$

where $\mathbf{frame}(p)$ denotes the set of variables possibly assigned by p. In this way the value of a variable not assigned in p and occurring in ψ is considered in the pre-state. It is the same as $\mathbf{post}(p)$ contains $x = \tilde{x}$.

For instance

pre $x > 0 \land y > 0$
post $z = x + y$
frame z
proc $p =$
...

Instead for explicitly state that the value of x is preserved by the execution of p, the contract just says that only z will be modified. If

$$\varphi = x > 0 \land y > x \land x = \tilde{x} + 100$$

and after the execution of **call** p the post-condition ψ is true,

$$\psi = z = x + y \land x = \tilde{x} + 100$$

then the side condition for the **call** rule would be

$$\begin{aligned} x > 0 \land y > x \land x = \tilde{x} + 100 \to & \forall z_f.((x > 0 \land y > 0 \to z_f = x + y) \to \\ & z_f = x + y \land x = \tilde{x} + 100) \end{aligned}$$

Procedures with Parameters

We now have to consider a list of formal arguments for procedure definitions and a list of expressions in the **call** command.

We only consider parameters passed by *value*. Parameters passed by *reference* could easily be considered in the syntax but their axiomatic semantics is much more complicated due to aliases (as for arrays).

$$\begin{array}{rl} \textbf{Arglist} & \lambda ::= a, \lambda \mid \varepsilon \\ \textbf{Proc} & \Phi ::= \textbf{pre}\,\varphi\,\textbf{post}\,\psi\,\textbf{proc}\,p(\lambda) = C \\ \textbf{Comm} & C ::= \ldots \mid \textbf{call}\ p(\overline{E}) \end{array}$$

For $p \in \mathbf{PN}(\Pi)$

- $\mathbf{param}(p) = \lambda$, i.e., list of formal parameters passed by value
- we have now global variables and parameter variables, which have local scope
- $\mathbf{pre}(p)$ and $\mathbf{post}(p)$ can contain occurrences of parameters
- any variable occurring in the body of a procedure and not in its parameter list is a global variable

For instance,

pre θ
post ρ
proc p(x,z)=
    C

We have $\mathbf{param}(p) = \{x, z\}$, but parameters may be substituted by fresh variables in the procedure's body and contract. The following definition is equivalent to the above if x' and z' do not occur free in C, θ, or ρ.

pre $\theta[x'/x, z'/z]$
post $\rho[x'/x, z'/z]$
proc p(x',z')=
    $C[x'/x, z'/z]$

If a variable is both global and a parameter, the global one would not be visible inside the procedure. But we will assume that *this* cannot occur: global variables cannot occur as parameters of a procedure $p \in \mathbf{PN}(\Pi)$.

We assume *static scope*: when a procedure is called the values of the caller's local variables do not affect the callee.

Parameters Passed by Value

Suppose first that a procedure p has a single formal parameter a. A procedure call rule without adaptation could be

$$\frac{\{\Pi\}}{\{\varphi\}\textbf{call}\ p(E)\{\psi\}} \text{ if } \varphi \rightarrow \mathbf{pre}(p)[E/a] \text{ and } \mathbf{post}(p)[E/a] \rightarrow \psi$$

- if a occurs in φ (or in E) its value is the value in the caller procedure

- if a occurs in **pre**(p) or **post**(p) its value is substituted by the one in the pre-state (caller).
- if a is not assigned in p, it is called a *constant value*; in this case, the mutual recursion rule is the same as for parameterless procedures.
- if a is assigned in p, as the internal value in p is irrelevant for the caller, in **post**(p) and ψ the value of a is the one in the pre-sate

However, if a is not a constant value, the mutual recursion rule must change. Consider just one branch

$$\frac{\begin{array}{c}[\{\Pi\}] \\ \vdots \\ \{\mathbf{pre}(p) \wedge a = \tilde{a}\}\mathbf{body}(p)\,\{\mathbf{post}(p)[\tilde{a}/a]\}\end{array}}{\{\Pi\}}$$

Example

Consider again the factorial as a one-parameter procedure.

pre $n \geq 0$
post $f = fact(n)$
proc FACTR (n)=
  $f \leftarrow 1$;$i \leftarrow 1$;
  while $i \leq n$ **do**
    $\{f = fact(i-1) \wedge i \leq n+1\}$
    $f \leftarrow f \times i$;
    $i \leftarrow i + 1$

The following triple can be derived

$$\{x \geq -10\}\mathbf{call}\ \text{FACT}(x+20)\{f = fact(x+20)\}$$

with the following side conditions:

$$x \geq -10 \rightarrow (n \geq 0)[x+20/n]$$
$$f = fact(x+20) \rightarrow f = fact(n)[x+20/n]$$

Deductive system for mutually recursive procedures with parameters passed by value

[mutual recursion pbv]

$$\frac{\begin{array}{cc} [\{\Pi\}] & [\{\Pi\}] \\ \vdots & \vdots \\ \{\widetilde{\mathbf{pre}(p_1)}\}\mathbf{body}(p_1)\,\{\widetilde{\mathbf{post}(p_1)}\} & \{\widetilde{\mathbf{pre}(p_n)}\}\mathbf{body}(p_n)\,\{\widetilde{\mathbf{post}(p_n)}\} \end{array}}{\{\Pi\}}$$

where

$$\widetilde{\mathbf{pre}(p_i)} = \mathbf{pre}(p_i) \wedge x_1 = \tilde{x_1} \wedge \cdots \wedge x_k = \tilde{x_k},$$

with $\tilde{\mathbf{Var}}(\mathbf{post}(p_i)) \cup \mathbf{param}(p_i) = \{x_1, \ldots, x_k\}$, and

$$\widetilde{\mathbf{post}(p_i)} = \mathbf{post}(p_i)[\tilde{a_1}/a_1, \ldots, \tilde{a_m}/a_m],$$

with $\mathbf{param}(p_i) = \{a_1, \ldots, a_m\}$.

[procedure call pbv]

$$\frac{\{\Pi\}}{\{\varphi\}\mathbf{call}\ p(\overline{E})\{\psi\}} \text{ if } \varphi \to \forall \overline{x_f}.((\mathbf{pre}(p)[\overline{E}/\overline{a}] \to \lfloor \mathbf{post}(p)[\overline{E}/\overline{a}, \overline{x_f}/\overline{x}] \rfloor) \to \psi[\overline{x_f}/\overline{x}])$$

where

$$\begin{aligned} \overline{a} &= \mathbf{param}(p) \\ \overline{x} &= \mathbf{Var}(\mathbf{post}(p)) \cup \mathbf{Var}(\psi) \\ \overline{x_f} &\text{ are fresh variables} \end{aligned}$$

- global variables that occur in $\overline{E}$ are not substituted by fresh variables in $\mathbf{post}(p)$ as they must be interpreted in the pre-state (thus the simultaneous substitution of $\overline{a}$ and $\overline{x}$).
- $\overline{x}$ has no parameter variables (of any procedure)
- parameters are not substituted by fresh variables

Example

Consider

pre $a > 0 \wedge y > 0$
post $z = a + y$
proc p(a)=
...

The variable y must be a global variable and suppose we onde to prove the following triple:

$$\{x > 0 \wedge y > x\}\mathbf{call}p(2 \times x + 1)\{z = 2 \times x + 1 + y\}$$

If x is global the side conditions is

$$(x > 0 \land y > x) \to \forall x_f, y_f, z_f.(2 \times x + 1 + y > 0 \land y > 0 \to z_f = 2 \times x + 1 + y_f) \\ \to z_f = 2 \times x_f + 1 + y_f$$

which is not valid:

- z_f is the final value of z and in the contract is given in terms of the value of x in the pre-state while the postcondition of the triple uses the value of x in the post-state.
- this can be fixed if we include in the contract $\textbf{post} z = a + y \land x = \tilde{x}$

But if x is a local parameter of the caller then the side condition is

$$(x > 0 \land y > x) \to \forall y_f, z_f.(2 \times x + 1 + y > 0 \land y > 0 \to z_f = 2 \times x + 1 + y_f) \\ \to z_f = 2 \times x + 1 + y_f$$

which is valid, as p cannot modify the value of a parameter.

Other features of procedures

- Parameters passed by value/reference
- Aliasing
- Return values of procedures
- Pure functions

For more see: [AFPMdS11] Chap. 8.2

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.